Carole Lamontagne

découvrir et maîtriser
Visual Basic 6.0

gestion de projet / typographie Eveline Martel
correction Sophie Mongrain

ISBN 978-2-923565-41-5

copyright © 2002 Loze-Dion éditeur inc.

Loze-Dion éditeur inc.
95, Saint-Sylvestre
Longueuil (Québec) J4H 2W1

Téléphone :	(450) 679-1955	www.lozedion.com
télécopieur :	(450) 679-6339	lde@videotron.ca

Tous droits réservés. On ne peut reproduire, enregistrer, ni diffuser aucune partie du présent ouvrage sous quelque forme ou par quelque procédé que ce soit sans avoir une autorisation écrite de l'éditeur.

À Guy,

Avec toute ma reconnaissance pour son soutien de chaque instant.

PRÉFACE

Découvrir et maîtriser Visual Basic 6.0 s'adresse à toute personne qui désire s'initier à la programmation avec le langage Visual Basic et concevoir rapidement des applications Windows. Ce volume est conçu pour les autodidactes et pour les étudiants sous la supervision d'un enseignant. Avant d'aborder la programmation d'applications Windows, vous devez connaître l'environnement Windows et plus particulièrement l'Explorateur de Windows.

Approche pédagogique

Dans les exercices proposés, vous **découvrirez** de nouvelles notions dont le code de programmation vous sera fourni intégralement. Ce code est toujours suivi d'un encadré dans lequel vous trouverez des détails et des renvois à des sections théoriques du volume. Toute nouvelle notion est reprise au moins une seconde fois, dans un même exercice ou dans un exercice subséquent, afin que vous puissiez la **maîtriser**. Le titre du volume *Découvrir et maîtriser Visual Basic 6.0* résume bien son approche pédagogique.

Structure du volume

Le Chapitre 1 propose une introduction au langage Visual Basic à la fois théorique et pratique. Deux exercices vous permettent de vous familiariser avec l'environnement de développement de Visual Basic et de créer rapidement de petites applications.

Le Chapitre 2 vous permet de travailler efficacement dans l'environnement de développement de Visual Basic 6.0. Il est important de le consulter avant d'aborder les exercices des chapitres suivants.

Le Chapitre 3 est entièrement pratique et compte une quinzaine d'exercices dans lesquels vous découvrirez graduellement les contrôles de la boîte à outils ainsi que les principaux éléments de programmation qui sont décrits dans les chapitres 4 et 5.

Dans le Chapitre 6, vous apprendrez comment utiliser les boîtes de dialogue standard de Windows dans vos projets et vous serez en mesure de programmer un éditeur de texte semblable à WordPad, comprenant un menu standard, une barre d'outils, une barre d'état et un menu contextuel.

Le Chapitre 7 traite de l'utilisation des fichiers de données afin de conserver les informations fournies par les utilisateurs lors de l'exécution de vos applications. Vous découvrirez comment valider des données, les enregistrer, les protéger et les imprimer dans des rapports.

Le Chapitre 8 permet de créer votre propre contrôle ActiveX. Tout au long de votre apprentissage vous manipulerez constamment des objets et dans ce dernier chapitre, vous découvrirez plus spécifiquement comment créer des propriétés, des méthodes et des évènements propres à une classe d'objets. Vous terminerez ainsi votre apprentissage avec une excellente base des concepts généraux de la programmation orientée objet.

Dans l'ensemble des exercices, l'accent est mis sur le respect des standards Windows concernant les interfaces graphiques utilisateur, la logique de programmation, l'utilisation de normes de programmation et la robustesse des programmes. Plusieurs techniques de validation sont utilisées pour vérifier les données au moment de la saisie et les erreurs d'exécution sont interceptées afin d'afficher des messages significatifs aux utilisateurs.

Le langage Visual Basic comportant un très grand nombre d'éléments, ce volume a été conçu pour accélérer l'accès aux informations qu'il contient. Un super index permet de retrouver un élément à la page où il est décrit et dans les exercices où il est utilisé.

Pour vous guider, repérez les symboles suivants :

Dans un exercice, ce symbole spécifie un renvoi à une section théorique du volume où un élément est détaillé.

Ce symbole indique qu'une alternative ou un truc est proposé.

Dans une section théorique, ce symbole identifie les exercices pratiques dans lesquels l'élément traité est utilisé.

Syntaxe des éléments du langage Visual Basic

Lors de la description des éléments du langage Visual Basic, des conventions syntaxiques sont utilisées. Par exemple, la syntaxe de l'instruction **Type** est illustrée comme suit :

Syntaxe : [**Private** | **Public**] **Type** *nomtype*
nomélément [([*indices*])] **As** *type*...
End Type

- Les mots en **gras** sont des mots qui appartiennent au langage Visual Basic et sont obligatoires. Exemple : **Type** et **End Type**.
- Les mots en *italique* indiquent que vous devez fournir un élément. Par exemple, il peut s'agir d'une instruction, du nom d'une variable ou encore d'un type de données. Exemple : *nomtype* spécifie qu'un type de données doit être cité à cet emplacement.

- Les éléments entre [crochets] sont facultatifs. Exemple : l'élément [*indices*] est optionnel ainsi que les mots **Private** et **Public**.
- Les éléments séparés par une barre verticale | indiquent qu'un seul des éléments cités doit être choisi. Exemple : les mots **Private** et **Public** sont optionnels mais si vous désirez utiliser ces mots, un seul des deux mots peut être cité.
- Des points de suspension (…) indiquent que l'élément qui précède peut être cité plusieurs fois. Exemple : *nomélément* [([*indices*])] **As** *type*… peut être énuméré plusieurs fois, l'instruction Type pouvant comporter plusieurs éléments.

  Exemple de codification d'une instruction **Type** :

```
Private Type Etudiant
  Nom As String * 30
  NotesObtenues(5) As Integer
End Type
```

Les images

Lors de la conception des interfaces graphiques, plusieurs images sont nécessaires. Il peut s'agir d'icônes (.ico), de curseurs (.cur) ou encore d'images bitmaps (.bmp). Ces images sont stockées dans des fichiers image qui sont fournis avec Visual Basic 6.0. Lors de l'installation de Visual Basic 6.0, ces fichiers sont généralement placés sous le dossier suivant :

C:\Program Files \Microsoft Visual Studio \Common \Graphics \…

Arborescence des fichiers

Plus de 300 fichiers sont fournis avec le volume*. Ces fichiers sont structurés de la façon suivante : Chapitre/Exercice/Debut et Chapitre/Exercice/Fin. Les chapitres correspondent aux numéros des chapitres du volume et les exercices sont numérotés de 1 à 28. Dans un sous-dossier Debut, vous trouverez les fichiers nécessaires à la réalisation d'un exercice. Tant qu'aux sous-dossiers Fin, ils contiennent les fichiers solution de chacun des exercices.

*Les fichiers ne sont pas inclus avec les volumes utilisés dans le cadre d'un cours. Ils sont fournis seulement à l'enseignant.

TABLE DES MATIÈRES

Chapitre 1 Introduction 1

1.1 Historique du langage de programmation Visual Basic 1
1.2 Paramètres spécifiques de configuration 2
1.3 Exercice 1 Découvrir Visual Basic 6
1.4 Exercice 2 14
1.5 La programmation orientée objet (OO) pilotée par des événements 20
1.6 L'Explorateur d'objets 22
1.7 Syntaxe de la programmation des objets 23
1.8 Convention d'affectation de noms aux objets (propriété *Name*) 24

Chapitre 2 L'environnement de développement 28

2.1 La fenêtre principale 28
2.2 La fenêtre Explorateur de projets 29
2.3 La boîte à outils et les composants 30
2.4 La fenêtre de conception de feuille 33
2.5 La fenêtre Présentation des feuilles 37
2.6 La fenêtre Propriétés 38
2.7 La fenêtre de code 40
2.8 La barre de menus 45

Chapitre 3 Découvrir et maîtriser Visual Basic par la pratique 69

3.1 Exercice 3 Créer le projet *Inscription* 69
3.2 Exercice 4 Créer le projet *Cadran* 75
3.3 Exercice 5 Créer le projet *Édition de texte* 81
3.4 Exercice 6 Modifier le projet *Groupe de contrôles* 85
3.5 Exercice 7 Valider et convertir avant de calculer 87
3.6 Exercice 8 Créer le projet *Calcul de factures* 96
3.7 Exercice 9 Créer le projet *Financement automobile* 103
3.8 Exercice 10 111
3.9 Exercice 11 Créer le projet *Emplois en informatique* 115
3.10 Exercice 12 Modifier le projet *Édition de texte* 122
3.11 Exercice 13 Modifier le projet *Calcul de moyennes* 124
3.12 Exercice 14 130
3.13 Exercice 15 Les différentes structures répétitives 137
3.14 Exercice 16 142
3.15 Exercice 17 Compléter le projet *Regroupement de projets* 152

Chapitre 4 Les objets, leurs propriétés, méthodes et événements 158

4.1 L'objet *Form* 158
4.2 Les contrôles intégrés de la boîte à outils 165
4.3 Les groupes de contrôles (propriété *Index*) 186
4.4 Les propriétés communes des objets 189
4.5 Les méthodes communes des objets 197
4.6 Les événements communs des objets 199
4.7 Les menus 214
4.8 Les masques de saisie (contrôle *MaskEdBox*) 221
4.9 Les contrôles FlatScrollBar, Slider et UpDown 224
4.10 L'objet *App* 227
4.11 L'objet *ClipBoard* (le Presse-papiers de Windows) 228
4.12 Ajouter et supprimer dynamiquement un contrôle 228

Chapitre 5 Les éléments du langage de programmation 231

5.1 Les modules 231
5.2 Les types de fichier d'un projet Visual Basic 232
5.3 L'objet de démarrage 233
5.4 Les procédures 234
5.5 Les instructions d'arrêt (End, Unload et Stop) 241
5.6 Les éléments de syntaxe 241
5.7 Les variables et les types de données 243
5.8 Les constantes 249
5.9 L'affectation par l'opérateur = 251
5.10 Les expressions mathématiques 253
5.11 La concaténation des chaînes de caractères 255
5.12 Affichage de boîtes de dialogue prédéfinies 255
5.13 Les structures de décision 261
5.14 Les structures répétitives 267
5.15 La structure *With* 273
5.16 L'instruction *Beep* 274
5.17 La gestion des erreurs 274
5.18 Les tableaux en mémoire 280
5.19 Les types de données personnalisés (instruction *Type*) 284
5.20 Les fonctions intégrées 286

Chapitre 6 Découvrir des contrôles utiles par la pratique 308

6.1 Les contrôles du système de fichiers 308
Exercice 18 Créer le projet *Sélection d'un fichier* 308

6.2 Les boîtes de dialogue standard de Windows 312
Exercice 19 Créer le projet *Boîtes de dialogue standard* 317
Exercice 20 Programmer un éditeur de texte 323
6.3 Le contrôle *RichTextBox* 332
Exercice 21 Modifier le projet *Boîtes de dialogue standard* 334
6.4 Les barres d'outils et les barres d'état 337
Exercice 22 Compléter le projet *Barre d'outils et barre d'état* 343

Chapitre 7 Les fichiers de données et les rapports 353

7.1 Les fichiers de données 353
Exercice 23 Modifier le projet *Inscription* 365
7.2 Production de rapports à l'aide de l'objet *Printer* 367
Exercice 24 Imprimer le fichier de données du projet *Inscription* 373
Exercice 25 Modifier le projet *Regroupement de projets* 377
Exercice 26 Créer le projet *Sondage* 384
Exercice 27 Créer le projet *Validation de données* 391

Chapitre 8 Création d'un contrôle ActiveX 405

8.1 Introduction à la technologie ActiveX 405
Exercice 28 Créer le projet *Contrôle Pointage* 405
8.2 L'Assistant Interface de contrôle ActiveX VB 6 423

Annexe A 425

Annexe B 428

Bibliographie 430

Index 431

1 Introduction

1.1 HISTORIQUE DU LANGAGE DE PROGRAMMATION VISUAL BASIC

Le langage de programmation BASIC (***B****eginner's* ***A****ll purpose* ***S****ymbolic* ***I****nstruction* ***C****ode*) fut créé dans les années 60 par des chercheurs de l'Université de Darmouth en Nouvelle-Écosse. Lors de sa création, le langage BASIC était destiné aux *débutants* en programmation, d'où le mot *beginner*. Avec l'apparition des premiers ordinateurs domestiques au début des années 80, Bill Gates et Paul Allen adaptèrent le langage BASIC aux micro-ordinateurs de type PC. Ils fondèrent alors, la société *Microsoft* afin de commercialiser ce langage.

Les premiers PC incluaient une version du langage Basic dont la première fût GW-BASIC, suivie de QuickBasic et finalement QBasic, toutes ces versions étant utilisables sous DOS.

Au début des années 90, avec la plate-forme Windows, est apparu le langage de programmation **Visual Basic** (VB) qui permettait aux programmeurs de créer rapidement les interfaces graphiques de Windows. Aujourd'hui, ce langage ne présente plus que quelques similitudes avec son prédécesseur BASIC. C'est un langage orienté objet (OO) piloté par des événements, facile à utiliser, même pour un amateur, mais qui au fil des années est devenu de plus en plus sophistiqué et volumineux. Il est actuellement un outil de développement couramment utilisé par des programmeurs professionnels. Il demeure toujours un excellent outil d'apprentissage de la programmation et il permet aux débutants de se familiariser avec les divers éléments de la programmation objet (classes, objets, propriétés, méthodes).

On retrouve sur le marché 3 éditions de Visual Basic 6.0 : Initiation, Professionnelle et Entreprise. La version *Professionnelle* est celle utilisée dans ce volume. Cette version permet aux professionnels du domaine de l'informatique de développer des projets et d'en faire la distribution. Elle compte trois CD : un pour VB et deux pour MSDN (*Microsoft Developer Network*), l'aide en ligne de Visual Basic sous le format HTML. La bibliothèque MSDN fait partie de la famille de produits de développement

d'applications *Microsoft Visual Studio 6.0* qui comprend entre autres, Visual C++ et Visual FoxPro.

On retrouve deux dérivés de Visual Basic : VBA et VBScript. VBA (***V**isual **B**asic for **A**pplications*) est une variante réduite de Visual Basic, utilisée pour l'automatisation de tâches dans les applications *Microsoft Office* (Word, Excell, Access). VBScript est un langage de script surtout utilisé sur les serveurs Web et interprété par le navigateur des clients.

Le dernier produit d'outils de développement de Microsoft, Visual Studio.NET est conçu pour les applications et les services Web distribués. On y retrouve le sous-produit Visual Basic.NET qui comporte des changements importants par rapport à Visual Basic 6.0, au niveau du langage et de l'environnement de développement. La version anglaise de ce produit est apparue au début de l'année 2002 et la version française éducationnelle n'est pas encore disponible en magasin (juin 2002).

Le principal concurrent de Visual Basic est l'outil de développement Delphi, un produit de la compagnie *Borland*. Ces deux outils de type RAD (*Rapid Application Development*) sont similaires. Cependant, en terme de programmation orientée objet, dans Visual Basic, l'absence du concept d'héritage (création d'une nouvelle classe à partir d'une classe existante), constitue une faiblesse. Cette lacune est cependant corrigée dans la version Visual Basic.NET.

1.2 PARAMÈTRES SPÉCIFIQUES DE CONFIGURATION

Avant de débuter les exercices de ce volume, il est nécessaire d'apporter quelques changements dans les paramètres de configuration de votre logiciel Visual Basic 6.0 ainsi que dans les paramètres régionaux de Windows, afin d'éviter certaines ambiguïtés qui pourraient survenir.

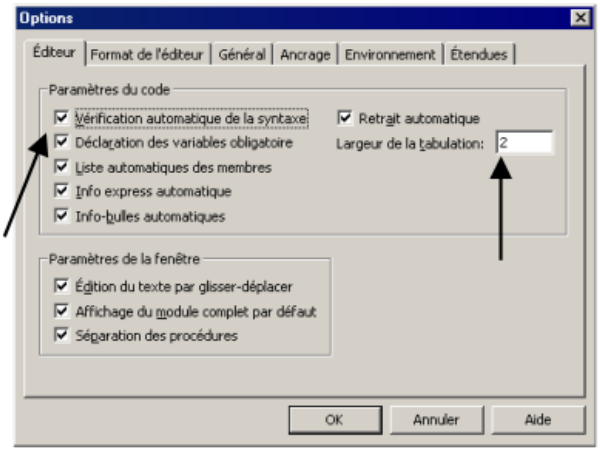
- Dans Visual Basic, sélectionnez le menu Outils et le choix Options. Dans la boîte de dialogue Options, sélectionnez l'onglet Éditeur.

  La case à cocher **Déclaration des variables obligatoire** doit être activée.

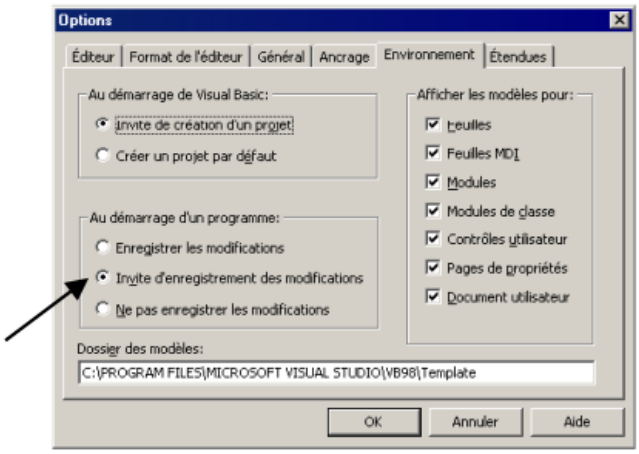
  Cette option ajoute automatiquement la ligne "Option Explicit" au début des modules de code. Cette ligne spécifie d'afficher un message d'erreur lorsqu'une variable utilisée dans le programme n'a pas été déclarée. Il n'est pas obligatoire de déclarer les variables en VB et en l'absence d'une déclaration, VB procède à une déclaration qui correspond rarement à nos besoins. Dans plusieurs cas, l'affichage d'un message aide le programmeur à résoudre rapidement une erreur de déclaration de variables.

La valeur dans la zone "Largeur de la tabulation" doit être fixée à 2.

Dans le code de programmation, plusieurs *retraits* successifs doivent être effectués pour en améliorer la lisibilité. Un retrait est généré en pressant la touche Tab et il occupe une largeur de 4 caractères, ce qui est beaucoup trop pour les programmes qui seront réalisés à l'aide de ce volume.



- Sélectionnez l'onglet Environnement et le choix "Invite d'enregistrement des modifications".

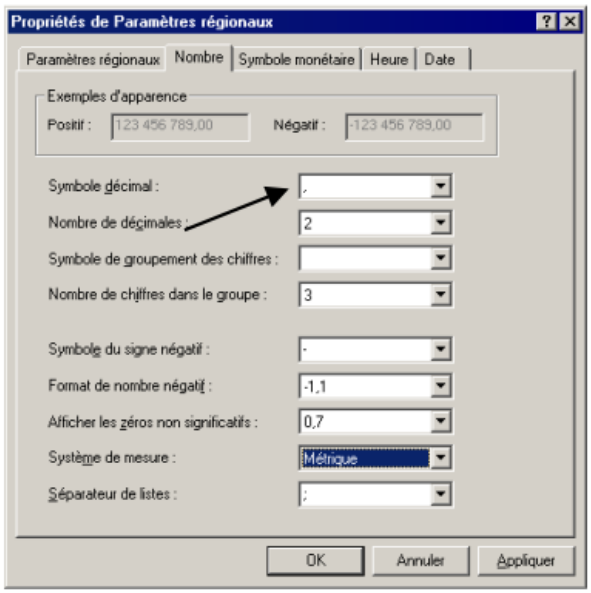


Lorsqu'un projet est testé en mode exécution, l'option "Invite d'enregistrement des modifications" provoque l'affichage d'une confirmation d'enregistrement des dernières modifications effectuées. Il est alors possible de répondre *oui* ou *non* ou encore d'*annuler* l'exécution.

Le choix par défaut "Ne pas enregistrer les modifications" est risqué. Un programme en développement provoque parfois un gel du système et des modifications peuvent ainsi être perdues.

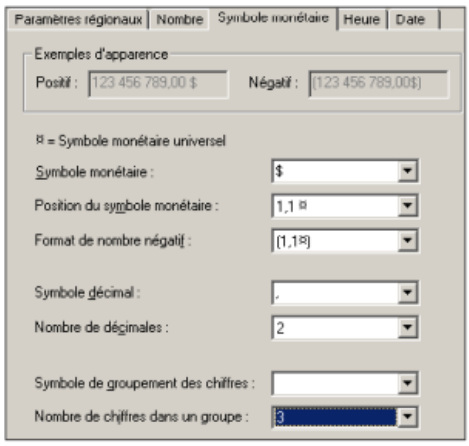
Le troisième choix "Enregistrer les modifications" est restrictif parce qu'il est possible qu'à certains moments, le programmeur ne désire pas enregistrer ses derniers changements.

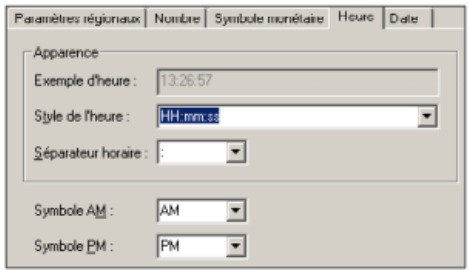
- Accédez au Panneau de configuration de Windows, sélectionnez "Paramètres régionaux" et par la suite, l'onglet Nombre.

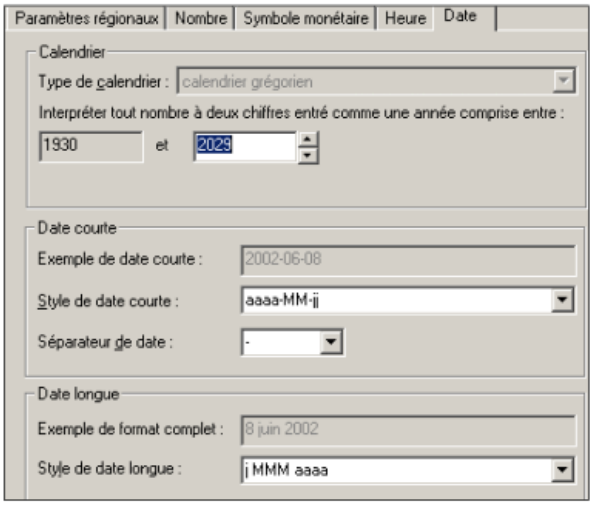


Le **symbole décimal** utilisé dans les nombres doit être la **virgule**. Cette modification est essentielle parce que le choix du symbole décimal du système a une implication directe sur l'utilisation des fonctions numériques et des fonctions de dates et heures.

- Modifiez le symbole *décimal* si nécessaire et vérifiez la concordance des autres éléments de l'onglet Nombre. Par la suite, comparez le contenu des onglets Symbole monétaire, Heure et Date avec les trois figures qui suivent et apportez des modifications au besoin.







1.3 EXERCICE 1 DÉCOUVRIR VISUAL BASIC

Partie 1 Explorer l'environnement de Visual Basic

1. À partir de **l'Explorateur de Windows**, créez un nouveau dossier **.../Visual Basic**.

   Le dossier **.../Visual Basic** contiendra tous les fichiers de l'ensemble des exercices du volume. L'unité de disque et le chemin d'accès sont laissés à votre discrétion. Pour chaque exercice, un nouveau sous-dossier sera créé.

2. Exécutez le fichier Chapitre1/Exercice1/Debut/Question.exe du volume. Cliquez sur Oui et sur Non et observez le résultat. L'exercice 1 vise à programmer cette petite application.

3. Pour démarrer Visual Basic 6.0 :

   Cliquez sur le bouton **Démarrer** de la barre des tâches et sélectionnez les choix **Programmes / Microsoft Visual Studio 6.0 / Microsoft Visual Basic 6.0 /** Onglet **Nouveau** de la fenêtre Nouveau Projet / **EXE standard** / Bouton **Ouvrir**. À cette étape, vous êtes en mode **[création]** d'un nouveau projet nommé **Projet1**. Le nom du projet et le mode de travail sont affichés dans la barre de titre du projet.

4. Observez et identifiez les différentes fenêtres de l'environnement de conception :
   - La fenêtre **Boîte à outils** (à gauche)
   - La fenêtre **Conception de feuilles** (au centre, Projet1 – Form1)
   - La fenêtre **Explorateur de projets** (en haut, à droite)
   - La fenêtre **Propriétés** (à droite, au centre)
   - La fenêtre **Présentation des feuilles** (en bas, à droite)

Si l'une des fenêtres n'est pas affichée, sélectionnez la fenêtre absente dans le menu Affichage.

Observez la figure illustrée à la page 28 du Chapitre 2, afin de bien identifier chacune des fenêtres de l'environnement de Visual Basic 6.0.

Le Chapitre 2 décrit en détails, les fenêtres, menus et barres d'outils de l'environnement de développement.

Partie 2 Créer un premier projet *Question truquée*

Étape 1 Nommer et enregistrer les composantes d'un nouveau projet

Tout projet Visual Basic compte au moins 2 fichiers :

`.vbp` (fichier Visual Basic Project)
`.frm` (fichier qui contient les spécifications d'une feuille ou de l'objet Form)

Il est fortement conseillé de créer un nouveau dossier pour chacun de vos projets Visual Basic afin de regrouper toutes les composantes de ceux-ci.

1. Nommez le projet :

   Dans la fenêtre Explorateur de projets, sélectionnez **Projet1**. Dans la fenêtre Propriétés, remplacez le nom **Projet1** par **Question** (à droite de la propriété **Name**). Observez le nom du projet qui apparaît maintenant sur la barre des tâches et dans la fenêtre Explorateur de projets.

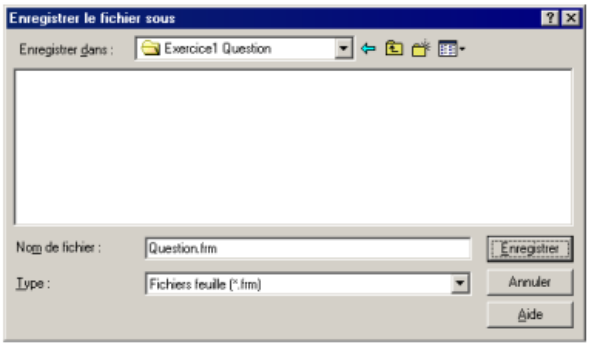
2. Nommez la feuille (objet Form) :

   Dans la fenêtre Explorateur de projets, sélectionnez **Form1**. Dans la fenêtre Propriétés, remplacez le nom **Form1** par **frmQuestion** (propriété **Name**). Respectez les minuscules et les majuscules dans les noms. Observez le nom qui apparaît maintenant dans la fenêtre Explorateur de projets.

3. Enregistrez la feuille :

> Il peut sembler prématuré d'enregistrer des fichiers avant même d'avoir effectué d'autres tâches. C'est une bonne habitude à prendre pour éviter toute perte éventuelle de votre travail. La forme et le projet sont sauvegardés une première fois et par la suite, vous n'aurez qu'à cliquer sur la disquette de la barre d'outils pour enregistrer en bloc tous les fichiers modifiés qui sont rattachés au projet.

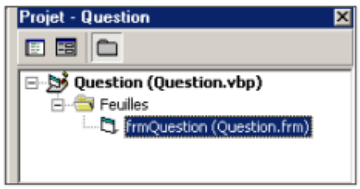
- Sélectionnez **Fichier** au menu et **Enregistrez frmQuestion sous...**
- Dans la boîte de dialogue "Enregistrer le fichier sous", sélectionnez votre dossier Visual Basic et cliquez sur le bouton "Créer un nouveau dossier" .
- Nommez le nouveau dossier **Exercice1 Question**. Cliquez sur le bouton Ouvrir.
- Modifiez le nom de fichier pour **Question.frm** et cliquez sur le bouton Enregistrer.



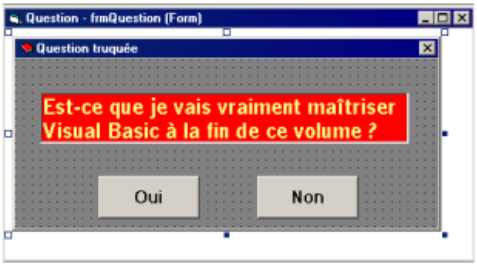
4. Enregistrez le projet :
   - Sélectionnez **Fichier** au menu et **Enregistrer le projet sous...**
   - Dans la boîte de dialogue "Enregistrer le projet sous", assurez-vous que le dossier actif est bien **Exercice1 Question** et cliquez sur le bouton Enregistrer afin de sauvegarder le fichier **Question.vbp**.
5. Observez la fenêtre Explorateur de projets.

Question est le nom du projet et (**Question.vbp**) est le nom du fichier qui contient les spécifications du projet.

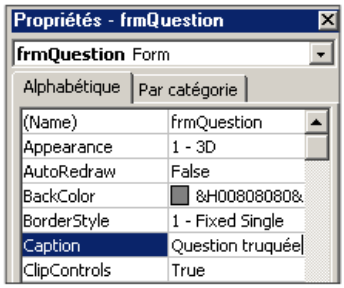
frmQuestion est le nom de l'objet Form (nom de la feuille) et (Question.frm) est le nom du fichier qui contient les spécifications de la feuille.



Étape 2 Conception de l'interface graphique du projet



1. Dans la fenêtre Form1 (au centre de la fenêtre principale), ajustez la taille de la feuille de façon approximative avec les poignées de dimension, en vous guidant sur la figure qui précède.
2. Dans la fenêtre Propriétés, sélectionnez la propriété **Caption** (texte de la barre de titre) et entrez le titre "Question truquée".



Notez que les propriétés sont en *ordre alphabétique* sauf la propriété (**Name**) qui apparaît toujours en premier.

3. Sélectionnez la propriété **Icon** (icône de la barre de titre). Cliquez sur le bouton à droite de la propriété afin d'ouvrir la boîte de dialogue "Charger une icône". Sélectionnez le fichier Chapitre1/Exercice1/Debut/BOOK01A.ICO du volume et cliquez sur le bouton Ouvrir. Observez l'icône dans le coin supérieur gauche de la feuille.
4. Fixez d'autres propriétés à l'objet frmQuestion.

   BackColor : Sélectionnez une couleur d'arrière-plan dans l'onglet Palette

   BorderStyle : 1-Fixed single (boutons Réduire et Agrandir enlevés, taille non modifiable par l'utilisateur)

   StartUpPosition : 2-Center screen (fenêtre affichée au centre de l'écran)
5. Dans la boîte à outils, double-cliquez sur l'objet **CommandButton** (bouton de commande) .
6. Dans la fenêtre Propriétés, modifiez la propriété **Name** pour **cmdOui**.

Pour modifier une propriété, sélectionnez d'abord l'objet concerné pour accéder à sa fenêtre Propriétés.

Selon la *convention d'affectation des noms des objets* (propriété **Name**, réf. 1.8), un nom d'objet est précédé d'une abréviation minuscule qui identifie l'objet (exemple : cmd pour CommandButton). Le nom de l'objet débute par une majuscule (exemple : cmdOui).

Les **objets** de la boîte à outils se nomment aussi **contrôles**.

7. Modifiez les propriétés suivantes de l'objet cmdOui :

   Caption : Oui (titre du bouton)
   Font : Taille 12, style gras (police de caractères)
8. Si nécessaire, ajustez la taille du bouton cmdOui et glissez-le à l'endroit désiré sur la feuille.
9. Ajoutez le bouton de commande Non sur la feuille et fixez les propriétés suivantes :

   Name : cmdNon
   Caption : Non
   Font : taille 12, style gras

10. Dans la boîte à outils, sélectionnez un contrôle **Label** (**étiquette** qui contient généralement un titre) et dessinez-le sur la feuille. Fixez les propriétés qui suivent et ajustez la taille et la position de l'étiquette.

    Name : lblTitre
    Caption : Est-ce que je vais vraiment maîtriser Visual Basic à la fin de ce volume ?
    BackColor : Sélectionnez une couleur d'arrière-plan dans l'onglet Palette
    BorderStyle : 1-Fixed single (style de la bordure de l'étiquette)
    Font : Fixez une police et une taille
    ForeColor : Sélectionnez une couleur de texte dans l'onglet Palette

11. Enregistrez votre travail en cliquant sur la disquette de la barre d'outils.

12. Exécutez le projet en cliquant sur le bouton Exécuter de la barre d'outils ou en pressant la touche F5.

13. Observez la fenêtre qui s'affiche au centre de l'écran. Pour l'instant, les boutons Oui et Non ne sont pas opérationnels. La prochaine étape vise à programmer ces deux boutons. Quittez le mode [exécution] par le bouton Fermer , afin de revenir au mode [création].

14. Attribuez la valeur 1680 à la propriété **Top** des boutons Oui et Non. Réajustez la taille de la feuille si nécessaire.

> La propriété **Top** est la distance entre le haut d'un objet et le haut de la feuille, exprimée en **twips**. (567 twips = 1 cm.)

Étape 3 Programmation des procédures événementielles

> Une **procédure événementielle** est un ensemble d'instructions propres à un objet qui sont exécutées lors du déclenchement d'un événement. L'événement le plus courant est l'événement **Click** qui survient lorsque l'utilisateur clique sur un objet.
>
> Le nom d'une procédure événementielle est formé du nom de l'objet, du caractère _ et du nom de l'événement. Par exemple, cmdOui_Click.
>
> Une procédure débute par le mot **Sub** et se termine par les mots **End Sub**.
>
> Les procédures sont codées dans la **fenêtre de code**.

1. Codez la procédure événementielle du bouton Oui :

    - Double-cliquez sur le bouton Oui en mode [création] afin d'accéder directement à la **fenêtre de code**.

- Entrez les lignes de code illustrées en caractère gras. Notez que la première et la dernière ligne de la procédure ont été insérées automatiquement. Avant de taper les lignes de code, lisez attentivement le texte de l'encadré qui suit le code.

```
Private Sub cmdOui_Click()
  'Message affiché si l'utilisateur choisit le bouton Oui
  MsgBox "...à la condition de faire TOUS les exercices!", _
                vbExclamation, "Oui..."
End Sub
```

La combinaison des touches **Ctrl + Espace** permet de compléter un mot automatiquement. Tapez les premiers caractères d'un mot, et pressez Ctrl + Espace. L'éditeur de Visual Basic tentera de compléter lui-même le mot. Cette combinaison de touches est un élément *très utile* de l'éditeur de VB. Il vous permettra de réduire le texte à taper de façon appréciable. **À retenir !**

Lors de la saisie du code de programmation, tapez *tous les caractères en minuscules*. L'éditeur de Visual Basic convertit automatiquement les caractères en majuscules lors d'un changement d'instruction.

Une ligne qui débute par une *apostrophe* est un *commentaire* (affiché en vert).

Utilisez la touche **Tab** pour mettre en retrait les instructions à l'intérieur d'une procédure. Le retrait par défaut est d'une largeur de 4 caractères. Je suggère d'utiliser seulement *2 caractères*. Si la tabulation est de 4 caractères, modifiez cette option en sélectionnant au menu, les choix Outils / Options / onglet Éditeur / Largeur de la tabulation = 2.

La fonction **MsgBox** permet d'afficher un message à l'utilisateur dans une boîte de dialogue (réf. La fonction *MsgBox*, page 256). Le premier argument est le titre du message, le deuxième spécifie une icône (vbExclamation affiche un point d'exclamation jaune) et le dernier est le titre de la barre de titre.

Le **caractère de continuité de ligne** _ , à la fin d'une ligne de code, indique que l'instruction continue sur la ligne suivante. On peut ainsi répartir une instruction sur plusieurs lignes. Le caractère de continuité doit être *précédé d'une espace*.

Erreurs de syntaxe dans le code

Lors de la saisie d'une instruction, l'interpréteur de Visual Basic vérifie s'il y a des *erreurs de syntaxe*. En présence d'une erreur, l'instruction est réaffichée en *rouge*, un message est affiché et le curseur est placé quelque part sur la ligne en erreur (l'erreur n'est pas forcément sur le caractère où se trouve le curseur...). Toute erreur de syntaxe doit être corrigée avant d'exécuter le projet.

- Exécutez le projet et testez le bouton Oui. Observez les différents éléments du message affiché : texte du message, icône et barre de titre.

2. Codez la procédure événementielle du bouton Non :

- Dans le haut de la fenêtre de code, 2 zones de liste sont présentes : la liste **Objet** (gauche) et la liste **Procédure** (droite). Sélectionnez **cmdNon** dans la liste Objet et **MouseMove** dans la liste Procédure.

Visual Basic insère la procédure de l'événement le plus courant de l'objet, sans laisser le temps au programmeur de sélectionner un événement dans la liste Procédure. Dans le cas présent, l'événement **Click** a été ajouté.

- Sélectionnez l'événement **MouseMove** dans la liste Procédure. La procédure Click sera supprimée lors de la prochaine exécution.

Évitez de taper la première ligne d'une procédure !

Utilisez les listes Objet et Procédure de la fenêtre de code afin ***d'éviter des fautes de frappe inutiles***.

- Entrez les lignes de code illustrées en gras de la procédure cmdNon_MouseMove :

```
'Procédure exécutée lorsque le pointeur de la souris
'atteint le bouton Non
Private Sub cmdNon_MouseMove(Button As Integer, _
                Shift As Integer, X As Single, Y As Single)
  If cmdNon.Top = 1680 Then
    cmdNon.Top = 0
  Else
    cmdNon.Top = 1680
  End If
End Sub
```

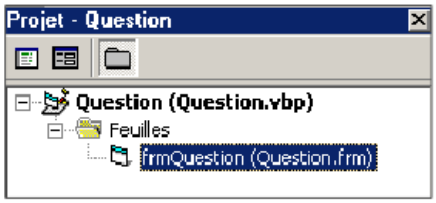
L'événement **MouseMove** est déclenché lorsque le pointeur de la souris atteint un objet. Lorsque l'utilisateur pointe le bouton Non, la propriété Top du bouton est modifiée, provoquant ainsi un déplacement du bouton. La valeur 0 de la propriété Top du bouton indique que celui-ci est situé à l'extrémité supérieure de la feuille.

Consultez la rubrique 5.13.1 qui traite de la structure de décision de l'instruction If.

- Enregistrez tous les fichiers du projet, exécutez et testez le bouton Non.

Important !

Pour passer de la fenêtre Conception de feuille à la fenêtre de code et vice-versa, utilisez les boutons **Code** (bouton de gauche) et **Afficher l'objet** (bouton du centre) dans le haut de la fenêtre de l'Explorateur de projets.



- Cliquez plusieurs fois sur les boutons **Code** et **Afficher l'objet** et observez le passage d'une fenêtre à l'autre.

Étape 4 Créer le fichier exécutable (.exe)

- Sélectionnez **Fichier** au menu et le choix **Créer Question.exe...**
- Dans la boîte de dialogue Créer le projet, assurez-vous que le dossier actif est bien **Exercice1 Question** et pressez le bouton OK.
- Enregistrez tous les fichiers du projet et fermez Visual Basic.
- Accédez à l'Explorateur de Windows et observez le contenu de votre dossier .../Visual Basic/Exercice1 Question. Exécutez le fichier **Question.exe** par un double-clic sur son nom.

1.4 EXERCICE 2

Partie 1 Créer le projet *Jean qui rit et Jean qui pleure*

Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre1 /Exercice2 /Debut Jean dans un nouveau dossier .../Visual Basic /Exercice 2 Jean.

2. Exécutez le fichier .../Visual Basic /Exercice 2 Jean /**Jean.exe** et cliquez plusieurs fois sur le bouton de gauche.

> Deux images ont été superposées lors de la conception de l'interface graphique. Lorsque l'une des images est visible, l'autre est dissimulée (propriété Visible). Aussi, le texte d'une étiquette est modifié selon l'image affichée.

Étape 2 Nommer et sauvegarder les composantes d'un nouveau projet

1. Dans Visual Basic, créez un nouveau projet de type Exe standard.
2. Nommez le projet **Jean** et la forme **frmJean** (propriété Name). Enregistrez la forme dans le fichier .../Visual Basic /Exercice 2 Jean / **Jean.frm** et le projet dans le fichier **Jean.vbp**.

> Pour enregistrer les composantes du projet qui ont été modifiées, cliquez sur la disquette de la barre d'outils.

Étape 3 Conception de l'interface graphique du projet

1. Fixez les propriétés de la feuille frmJean :
   - La barre de titre contient le titre "Jean qui rit et Jean qui pleure" ;
   - Seulement le bouton Fermer est disponible sur la barre de titre et la taille de la fenêtre ne peut être modifiée par l'utilisateur ;

- L'icône de la barre de titre est .../Visual Basic/Exercice 2 Jean /FACE03.ICO ;
- Le blanc est utilisé comme couleur d'arrière-plan ;
- La feuille doit être affichée au centre de l'écran.

2. Dessinez une étiquette (contrôle Label) au-dessus de l'image et fixez les propriétés suivantes :

Name	lblTexte
BackStyle	0-Transparent
couleur du texte	rouge
titre de l'étiquette	Jean qui rit
taille des caractères	14

3. Ajoutez un contrôle **Image** sur l'interface graphique .

Name	imgRit
Picture	Sélectionnez .../Visual Basic /Exercice 2 Jean /Jean qui rit.bmp
Visible	True

4. Ajoutez un second contrôle Image et superposez-le au premier.

Name	imgPleure
Picture	.../Visual Basic /Exercice 2 Jean /Jean qui pleure.bmp
Visible	False

> Lors du démarrage, l'image qui rit sera affichée et l'image qui pleure sera dissimulée.

5. Ajoutez et nommez les 2 boutons de commande :

Bouton **cmdCliquer** :

Style = 1- Graphical, arrière-plan : jaune, taille = 10

Bouton **cmdQuitter** :

Style = 1- Graphical, arrière-plan : jaune

Picture = .../Visual Basic /Exercice 2 Jean /TRFFC14.ICO

Étape 4 Programmation des procédures événementielles

1. Codez la procédure événementielle du bouton cmdQuitter :

```
Private Sub cmdQuitter_Click()
  Unload Me
End Sub
```

L'instruction **Unload Me** indique de décharger la feuille de la mémoire. Puisque le projet compte une seule feuille, le déchargement de celle-ci terminera l'exécution.

Consultez la section 5.5 Les instructions d'arrêt.

2. Codez la procédure événementielle du bouton cmdCliquer en traduisant le pseudo-code qui suit en Visual Basic :

```
Private Sub cmdCliquer_Click()
  Si l'image qui rit est visible
    - dissimuler l'image qui rit
    - rendre l'image qui pleure visible
    - afficher le texte "Jean qui pleure" dans l'étiquette
  sinon
    - dissimuler l'image qui pleure
    - rendre l'image qui rit visible
    - afficher le texte "Jean qui rit" dans l'étiquette
  Fin Si
End Sub
```

Consultez la section L'instruction If, page 261.

3. Testez votre projet jusqu'à ce qu'il fonctionne parfaitement.

Étape 5 Définir un bouton par défaut et un bouton d'annulation

1. Fixez la propriété suivante au bouton cmdCliquer : **Default = True**.

La propriété **Default** permet de définir un **bouton par défaut**. Au démarrage, celui-ci obtient le **focus** qui est représenté par une *bordure pointillée* encadrant le bouton.

Cette bordure indique à l'utilisateur qu'il peut presser la touche **Enter** ou la touche **Espace** au lieu de cliquer sur le bouton pour déclencher un événement. *Un seul bouton par défaut* peut être défini sur une feuille.

2. Fixez la propriété suivante au bouton cmdQuitter : **Cancel = True**.

> La propriété **Cancel** permet de définir un **bouton d'annulation**. Celui-ci permet à l'utilisateur de presser la touche **Esc** au lieu de cliquer sur le bouton pour déclencher un événement. *Un seul bouton d'annulation* peut être défini sur une feuille.

3. Testez le projet et pressez les touches **Enter**, **Espace** et **Esc** au lieu d'utiliser la souris.
4. Testez de nouveau le projet et pressez plusieurs fois la touche **Tab**.

> La touche **Tab** permet le déplacement entre les différents contrôles accessibles d'une feuille. Dans le cas présent, elle permet de passer d'un bouton à l'autre. Lorsque le bouton Quitter obtient le focus, les touches Enter et Esc sont alors équivalentes.

5. Créez et enregistrez le fichier exécutable Jean.exe.

Partie 2 Modifier le projet *Lumières de circulation*

Étape 1 Préparation, observation et ouverture d'un projet existant

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre1 /Exercice2 /Debut Lumieres dans un nouveau dossier .../Visual Basic /Exercice 2 Lumieres.
2. Exécutez le fichier .../Visual Basic /Exercice 2 Lumieres /**Lumieres.exe**.

> Comme dans la première partie de l'exercice, le changement des lumières est effectué par la *superposition d'images*. Dans le cas présent, trois images sont superposées : une lumière verte, une lumière jaune et une lumière rouge.

3. Dans Visual Basic, ouvrez le projet .../Visual Basic /Exercice 2 Lumieres /Lumieres.vbp en sélectionnant au menu Fichier /Ouvrir un projet /Onglet Existant /Rechercher le fichier /Bouton Ouvrir.

Étape 2 Modifier l'interface graphique

1. Le bouton cmdChanger doit être le *bouton par défaut* et le bouton cmdFermer doit être le *bouton d'annulation*.
2. Observez le contenu de la propriété **Caption** des 2 boutons de commandes. On y retrouve le caractère & dans le texte.

Le caractère **&** (*et commercial* ou *esperluette*) utilisé dans la propriété Caption permet de créer un **raccourci clavier** en utilisant la touche **Alt** et la lettre qui suit le caractère &. Cette lettre est *soulignée* à l'écran indiquant ainsi à l'utilisateur le raccourci clavier.

Exemple

La lettre *F soulignée* indique à l'utilisateur qu'il peut presser la combinaison des touches Alt + F pour fermer la fenêtre.

Étape 3 Coder les procédures

1. Codez la procédure du bouton Fermer. Lors d'un clic sur ce bouton, la feuille est déchargée de la mémoire et l'exécution est terminée.
2. Codez la procédure du bouton "Changer Lumières" :
   - Lorsque l'utilisateur sélectionne le bouton cmdChanger, les images doivent s'afficher dans l'ordre suivant : vert/jaune/rouge/vert/jaune/rouge. Une image visible doit être dissimulée afin que la suivante soit affichée.
   - Utilisez la structure suivante de l'instruction IF : (If/EsleIf/Else/End If)

```
If condition1 Then
   instruction(s)
ElseIf condition2 Then
   instruction(s)
Else
   instruction(s)
End If
```

Consultez la section L'instruction If, page 261.

3. Exécutez le projet, cliquez plusieurs fois sur le bouton "Changer Lumières" et assurez-vous de son bon fonctionnement.
4. Codez une procédure pour chacune des images qui sera exécutée lors d'un clic sur chacune d'entre elles. Par exemple, si l'utilisateur clique sur l'image verte, la procédure suivante sera exécutée :

```
Private Sub imgVert_Click()
  Call cmdChanger_Click
End Sub
```

L'instruction **Call** permet d'exécuter une procédure déjà codée. Cette instruction permet souvent de réduire le code de programmation en évitant des répétitions inutiles et illogiques.

Consultez la section Déclaration et appel d'une procédure événementielle Sub (instruction Call), page 234.

5. Exécutez le projet et cliquez plusieurs fois directement sur les lumières. Assurez-vous du bon fonctionnement des procédures *Click* de chacune des images.
6. Testez le fonctionnement des touches **Enter**, **Espace**, **Esc**, **Alt+L** et **Alt+F**.
7. Créez et enregistrez le fichier exécutable Lumieres.exe.

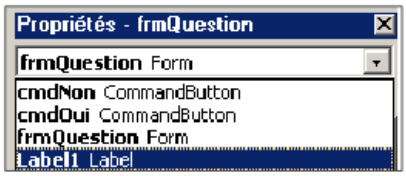
1.5 LA PROGRAMMATION ORIENTÉE OBJET (OO) PILOTÉE PAR DES ÉVÉNEMENTS

Visual Basic permet de créer rapidement une **interface graphique utilisateur** (GUI, Graphical User Interface). L'interface utilisateur qu'on appelle aussi **feuille** ou **fenêtre**, est un **objet Form** sur lequel d'autres **objets visuels** ou **graphiques** sont ajoutés, comme des boutons de commande, des images, des étiquettes, des zones de texte, des cases à cocher, ou encore des boutons d'option. Ces objets sont appelés **contrôles** ou **Composants ActiveX** qui est une norme de programmation Microsoft qui définit les caractéristiques des objets.

Chaque objet a des caractéristiques qui le décrit que l'on appelle ses **propriétés** (terme VB) ou ses **attributs** (terme OO). Ces caractéristiques peuvent être attribuées en mode [création] ou en mode [exécution].

Exemple : Observation du projet Question.vbp (Exercice 1)

On retrouve tous les **objets** du projet dans la **liste de sélection des objets** dans la fenêtre Propriétés.



La propriété **Name** de chacun des objets a été fixée dans la fenêtre des Propriétés en mode [création]. Par exemple, l'objet **Form** porte le nom **frmQuestion** et le **bouton Non** porte le nom **cmdNon**.

D'autres valeurs de propriétés ont été attribuées aux objets en mode [création] : **Caption**, **Icon**, **BackColor**, **ForeColor**, **BorderStyle**, **Font**, **StartUpPosition** et **Top**.

La propriété **Top** de l'objet **cmdNon** est modifiée en mode [exécution], ce qui permet le déplacement de l'objet.

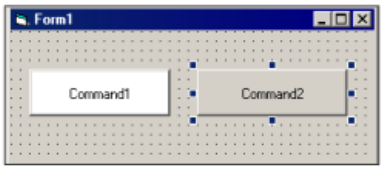
On peut agir sur un objet avec des **méthodes** en mode [exécution]. Une méthode (terme VB) ou un **comportement** (terme OO) d'un objet est représenté par un *verbe d'action*. Par exemple, la méthode **SetFocus** (placer le focus) indique de placer le curseur sur un objet et la méthode **Clear** (effacer) spécifie d'effacer le contenu d'un objet.

Les instructions Visual Basic réagissent à des **événements** qui peuvent être déclenchés, par exemple, lors d'un clic sur un objet (événement **Click**) ou encore, lorsque le pointeur de la souris atteint un objet (événement **MouseMove**). Les événements peuvent survenir en tout temps et les instructions sont exécutées lorsque les événements se produisent. Un programme Visual Basic est ainsi subdivisé en **procédures événementielles**, chaque procédure étant formée d'instructions, exécutables à la suite du déclenchement d'un événement propre à un objet. Dans le projet Question.vbp, on retrouve deux procédures événementielles : cmdOui_Click et cmdNon_MouseMove.

La principale différence entre un *langage procédural* (exemple : COBOL, BASIC, Pascal et C) et Visual Basic est la *nature événementielle* de ce dernier. Dans un langage procédural, les instructions sont exécutées séquentiellement *du début à la fin* du programme. Avec les événements, *il n'y a plus de début ni de fin*.

En terme de programmation orientée objet, on parle de **classe d'objet**. Une classe représente le modèle d'un objet. Une classe d'objet définit *l'objet* lui-même avec l'ensemble de ses *propriétés*, *méthodes* et *événements*. Un objet est une occurrence d'une classe, appelée **instance**, dont les caractéristiques varient d'un objet à l'autre. Lorsqu'un objet est ajouté sur une feuille, une *nouvelle instance de la classe* est alors créée.

Exemple

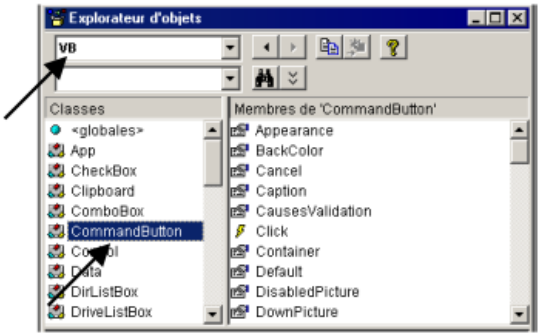


Les objets Command1 et Command2 sont des instances de la classe d'objet CommandButton. Leurs caractéristiques sont différentes puisque l'un est blanc et l'autre est gris. Chacun de ces 2 objets issus de la même classe, possède ainsi ses propres propriétés.

1.6 L'EXPLORATEUR D'OBJETS

L'**Explorateur d'objets** répertorie les *classes d'objets* existantes ainsi que les *constantes* et *fonctions* de Visual Basic.

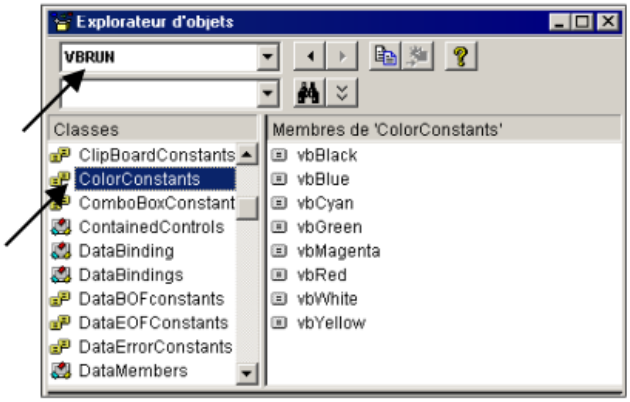
- Pressez la touche F2 ou sélectionnez au menu Affichage /Explorateur d'objets. Choisissez la bibliothèque VB et la classe CommandButton.



Des **icônes** sont utilisées pour différencier la nature des choix qui apparaissent dans les listes.

	Classe d'objet
	Propriété
	Événement
	Méthode ou fonction
	Groupe de constantes
	Constante Visual Basic

- Sélectionnez la bibliothèque VBRUN et la classe ColorConstants.



On peut visualiser tous les membres de la classe ColorConstants. Chacune des constantes de cette classe peut être utilisée dans le code de programmation pour représenter une couleur.

- Sélectionnez < Toutes bibliothèques > et la classe < Globales >. Vous y retrouverez la plupart des fonctions préprogrammées de Visual Basic. Une courte description de chaque fonction apparaît dans le bas de la fenêtre de l'Explorateur d'objets.

- Pour fermer l'Explorateur d'objets, cliquez sur le bouton Fermer de sa barre de titre. En l'absence de celle-ci, cliquez sur l'une des icônes de l'Explorateur de projets (Code) ou (Afficher l'objet).

En Visual Basic, on peut utiliser les objets existants ou encore créer de nouvelles classes d'objets par des modules de classe. On peut aussi créer de nouveaux composants ActiveX avec leurs propriétés, méthodes et événements. Cette technique de conception de nouveaux contrôles fait l'objet du Chapitre 8.

1.7 SYNTAXE DE LA PROGRAMMATION DES OBJETS

- Pour modifier la valeur d'une **propriété**, utilisez la syntaxe suivante :

Objet.Propriété = Valeur

Le nom de l'*objet* est suivi d'un *point*, lui-même suivi du nom de la *propriété*. Le caractère = permet d'affecter une *valeur* à la propriété. Cette valeur peut prendre plusieurs formes.

Exemple

```
cmdNon.Top = 3                      valeur numérique
txtNom.Text = "Tapez ici"           chaîne de caractères
imgRit.Visible = True               valeur booléenne
cmdBouton.BackColor = vbWhite       constante Visual Basic
chkFeminin.Value = vbChecked        constante Visual Basic
```

- Pour agir sur un objet avec une **méthode**, utilisez la syntaxe suivante :

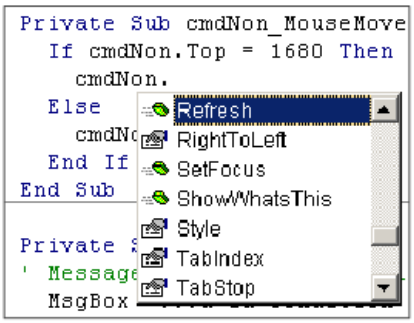
Objet.Méthode

Le nom de l'*objet* est suivi d'un *point*, lui-même suivi du nom de la *méthode*.

Exemple

```
cmdNon.SetFocus                     donner le focus au bouton cmdNon
cboProvinces.Clear                  effacer les provinces de la liste
```

Lorsque le point est tapé à la suite du nom de l'objet, l'éditeur de Visual Basic affiche une liste des propriétés et méthodes de l'objet. Une *icône* permet d'identifier s'il s'agit d'une *propriété* ou d'une *méthode*.



1.8 CONVENTION D'AFFECTATION DE NOMS AUX OBJETS (PROPRIÉTÉ NAME)

Les noms d'objets doivent respecter des *règles de syntaxe* et des *normes de programmation*. Les règles syntaxiques sont propres au langage de programmation et sont *obligatoires*, tandis que les normes de programmation sont *nécessaires* à une compréhension rapide du code de programmation. Ces normes sont suggérées dans l'aide de Visual Basic.

Normes de programmation et syntaxe de la propriété *Name* d'un objet

Normes de programmation :

- Le nom commence par un *préfixe* généralement formé de 3 lettres *minuscules* formant une abréviation de l'objet. Ce préfixe permet d'identifier rapidement la classe de l'objet.

  Exemple : Le préfixe cmd identifie un objet de la classe CommandButton.

- Le préfixe est suivi d'une identification qui commence par une lettre *majuscule*.

  Exemple : Le nom cmdEffacer identifie le bouton de commande Effacer et est plus facile à lire que cmdeffacer.

- L'identification peut comporter plusieurs mots. Dans ce cas, chacun des mots doit débuter par une lettre *majuscule*.

  Exemple : Le nom cmdEffacerResultats identifie le bouton de commande "Effacer les résultats".

Visual Basic ne tient pas compte des minuscules et des majuscules. Leur utilisation a pour but d'améliorer la lisibilité du code.

Règles de syntaxe :

- Un nom doit commencer par une lettre, sa longueur ne doit pas dépasser 40 caractères et il doit être formé seulement de lettres, de chiffres et du caractère de soulignement _.

  Exemple : lblNombre1, lblNombre2, lblNombre_Etudiants

- Évitez l'emploi de caractères accentués (exemple : é, É, ç, etc.).

Il n'est pas nécessaire de nommer tous les contrôles ajoutés sur une feuille. *Seulement les contrôles qui figurent dans le code de programmation doivent être nommés.*

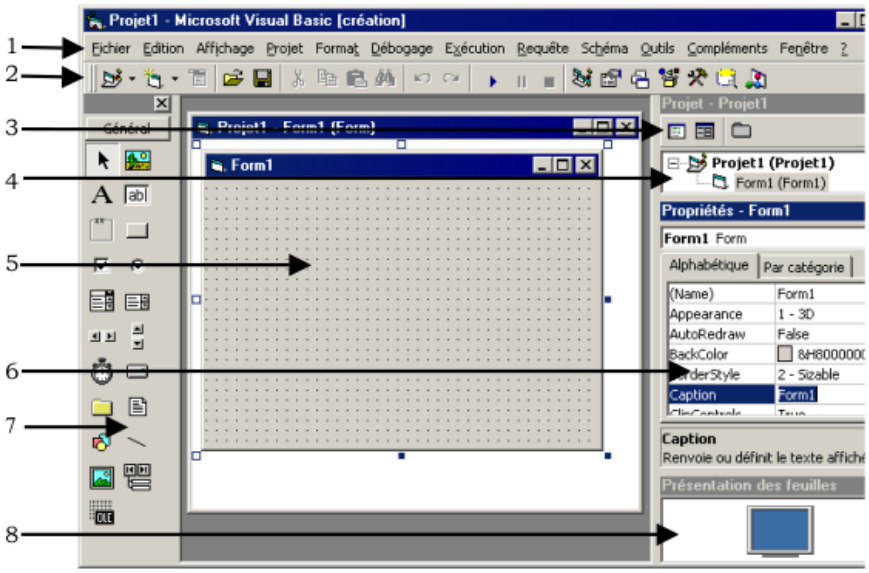
Tableau partiel des préfixes suggérés dans la rubrique *Conventions, Attribution de nom à un objet de l'index* de l'Aide MSDN

Objet	Préfixe	Objet	Préfixe
ADO (contrôle de données)	ado	Menu	mnu
Animation	ani	MMControl (MCI multimédia)	mci
CheckBox	chk	MonthView	mvw
ComboBox	cbo	MSChart (graphique)	cht
Command (modèle objet ADO)	cmd	MSFlexGrid	fgd
CommandButton	cmd	MSHFlexGrid	flex
CommonDialog	dlg	OLE	ole
CoolBar	clb	OptionButton	opt
Connection (modèle objet ADO)	cnn	PictureBox	pic
Data (contrôle de données)	dat	ProgressBar	prg
DataCombo	dbc	RecordSet (avec ADO)	rs
DataEnvironment (concepteur)	DE	ReportImage	rptimg
DataGrid	dgd	ReportFucntion	rptfnc
DataList	dbl	ReportLabel	rptlbl
DataRepeater	drp	ReportLine	rptlin
DataReport (concepteur)	rpt	ReportShape	rptshp
DirListBox	dir	ReportText	rpttxt
DriveListBox	drv	RichTextBox	rtf
DTPicker	dtp	Shape	shp
FileListBox	fil	Slider	sld
FlatScrollbar	fsb	SSTab	stab

Objet	Préfixe	Objet	Préfixe
Form	frm	StatusBar	sta
Frame	fra	SysInfo	sys
HScrollBar	hsb	TabStrip	tab
Image	img	TextBox	txt
ImageCombo	icbo	Timer	tmr
ImageList	ils	Toolbar	tlb
Label	lbl	TreeView	tre
Line	lin	UpDown	upd
ListView	lvw	VScrollBar	vsb
MaskEditBox	msk	Winsock	wsk
MDI (feuille)	mdi		

2 L'environnement de développement

2.1 LA FENÊTRE PRINCIPALE



1 Barre de menus
2 Barre d'outils Standard
3 Icône d'accès à la fenêtre de code
4 Fenêtre Explorateur de projets
5 Fenêtre de conception de feuille
6 Fenêtre Propriétés
7 Boîte à outils
8 Fenêtre Présentation des feuilles

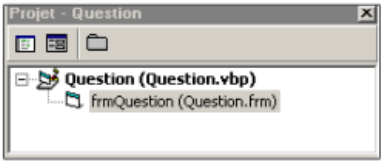
La *fenêtre principale* est celle qui apparaît au démarrage de Visual Basic. Elle regroupe la *barre de menus*, la *barre d'outils Standard* ainsi que 5 *fenêtres distinctes* : Explorateur de projets, Conception de feuille, Présentation des feuilles, Boîte à outils et

Propriétés. La fenêtre principale permet l'accès à une *sixième fenêtre* appelée Fenêtre de code. Chacune des fenêtres de l'environnement de Visual Basic possède *son propre menu contextuel*, accessible avec le bouton droit de la souris ou la touche correspondante du clavier.

L'environnement par défaut est de type MDI (Multiple Document Interface) qui permet de créer plusieurs feuilles, alors qu'un environnement de type SDI (Simple Document Interface) ne peut créer qu'une seule feuille à la fois. Il est possible de modifier l'environnement par défaut en sélectionnant au menu, Outils /Options /onglet Étendues.

La barre de titre affiche le nom du projet **Projet1** ainsi que le mode **[création]** qui signifie que le projet est à l'étape de **conception**. Pour tester un projet en cours de conception, on passe du mode [création] au mode **[exécution]** par la touche F5 ou le bouton Exécuter de la barre d'outils Standard . Le bouton Fin permet de revenir au mode [création].

2.2 LA FENÊTRE EXPLORATEUR DE PROJETS

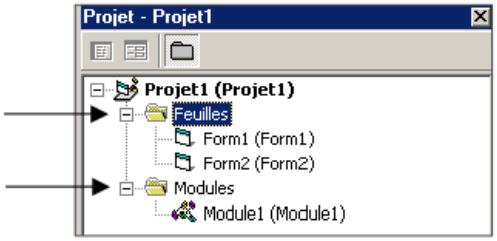


La fenêtre Explorateur de projets est affichée, en sélectionnant au menu Affichage /Explorateur de projets (Ctrl+R) ou par l'icône correspondante de la barre d'outils Standard .

Cette fenêtre utilise une structure arborescente pour illustrer les différentes composantes d'un ou plusieurs projets. Une *icône* spécifie le type de la composante. Elle est suivie de son *nom* et du *nom du fichier* entre parenthèses, dans lequel la composante a été enregistrée. L'absence de suffixe dans le nom du fichier indique que la composante *n'a pas encore été enregistrée*. On y retrouve aussi trois boutons fort utiles :

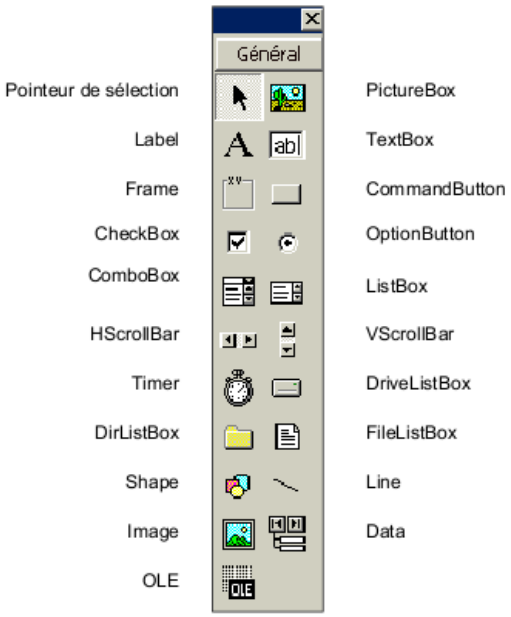
Icône	Nom du bouton	Fonction du bouton	Autres accès
	Code	Affiche la fenêtre de code.	menu Affichage/Code (F7)
	Afficher l'objet	Affiche la fenêtre de conception de feuille.	menu Affichage/Objet (Maj+F7)
	Basculer les dossiers	Affiche ou masque les types de composantes.	

Exemple : Le bouton "Basculer les dossiers" affiche les types Feuilles et Modules.



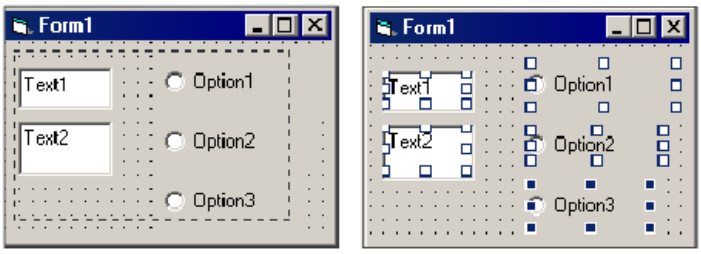
2.3 LA BOÎTE À OUTILS ET LES COMPOSANTS

La fenêtre Boîte à outils est affichée, en sélectionnant au menu Affichage /Boîte à outils ou par l'icône correspondante de la barre d'outils standard .



Le pointeur de sélection

La Boîte à outils contient un **pointeur de sélection** , qui sert à sélectionner des contrôles déjà en place sur la feuille de conception. Pour utiliser le pointeur de sélection, cliquez sur son bouton dans la boîte à outils et glissez le pointeur de la souris dans la fenêtre de conception, de façon à encadrer les éléments à sélectionner. Dans les figures qui suivent, cinq objets ont été sélectionnés.



Les contrôles intégrés

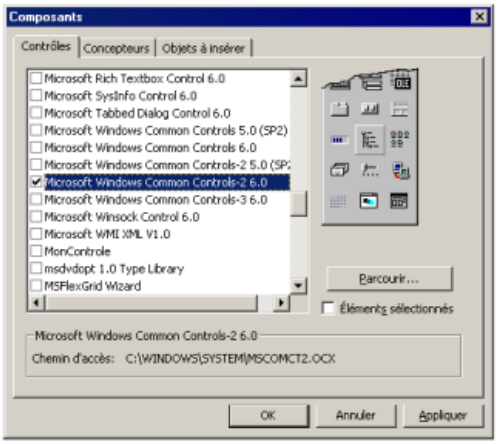
Dans la Boîte à outils, on retrouve aussi un bouton pour chacun des 20 objets appelés **contrôles intégrés** qui peuvent être ajoutés sur la feuille de conception. Lorsqu'un objet est ajouté, une nouvelle instance de la classe d'objet est alors créée.

Chacun des objets inclus dans la boîte à outils est détaillé dans la section 4.2 Les contrôles intégrés de la boîte à outils.

Ajout de composants

D'autres **composants**, appelés aussi **contrôles ActiveX**, peuvent être ajoutés dans la boîte à outils, selon les besoins du projet. Pour ajouter un composant :

- Sélectionnez le choix Composants du menu contextuel de la boîte à outils. On sélectionne le menu Projet /Composants, Ctrl+T.
- Cochez un ou plusieurs choix dans la liste et pressez OK.



- Observez le ou les nouveaux composants dans le bas de la boîte à outils.

La plupart des fichiers de type **Contrôle ActiveX** sont stockés dans le dossier C:\Windows\System et portent le suffixe .OCX. Le bouton Parcourir permet de sélectionner un autre dossier.

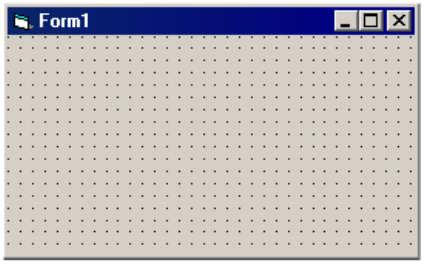
Plusieurs contrôles ActiveX sont utilisés dans les exercices de ce volume. Chacun est décrit lors de son utilisation pratique. La plupart d'entre eux sont décrits dans le Chapitre 4.

Les onglets de la boîte à outils

La boîte à outils peut contenir plusieurs onglets en plus de l'onglet

qui apparaît dans le haut de la boîte à outils. L'ajout de nouveaux onglets permet de regrouper des contrôles et ainsi de désencombrer l'onglet Général, lorsque celui-ci contient plusieurs composants ajoutés. Pour ajouter un onglet à la boîte à outils, sélectionnez le choix "Ajouter un onglet" du menu contextuel.

2.4 LA FENÊTRE DE CONCEPTION DE FEUILLE

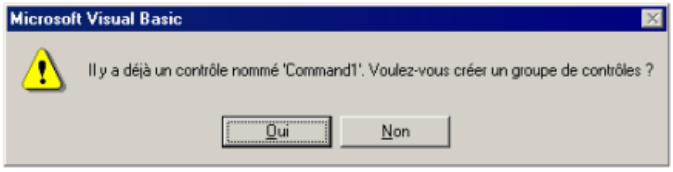


La fenêtre de conception de feuille permet de créer *l'interfaœ utilisateur*. La feuille ou fenêtre créée est une instance de la classe Form de Visual Basic et un objet dont le nom par défaut est **Form1**.

Ajout d'un contrôle

Double-cliquez sur le contrôle dans la boîte à outils ou cliquez sur le contrôle dans la boîte à outils pour le sélectionner et dessinez-le sur la feuille de conception en glissant avec la souris ; relâchez le bouton de la souris lorsque la taille désirée est atteinte.

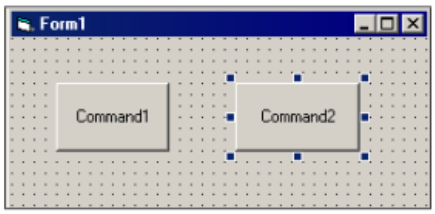
Il est possible de copier un contrôle existant par Copier/Coller afin d'ajouter des contrôles semblables (exemple : 4 boutons de commande de la même taille). Sélectionnez le contrôle à copier, pressez Ctrl+C et Ctrl+V. Pour ne pas créer un **groupe de contrôles**, répondez **Non** au message suivant :



Les groupes de contrôles sont décrits dans la section 4.3 Les groupes de contrôles.

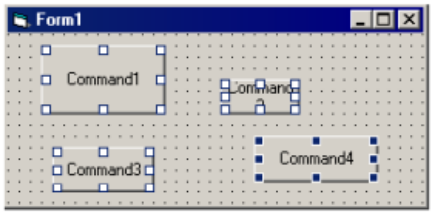
Sélection d'objets

Avant d'apporter des modifications à un ou plusieurs objets, ceux-ci doivent d'abord être *sélectionnés*. On identifie un objet sélectionné par les *poignées de dimension* (petits carrés) qui l'entourent. Dans la feuille qui suit, le bouton Command2 est sélectionné.



- Pour *annuler une sélection*, cliquez à l'extérieur de l'objet sélectionné.
- Pour sélectionner *un seul contrôle ou une feuille*, cliquez sur l'objet ou utilisez la liste de sélection des objets dans le haut de la fenêtre Propriétés.
- Pour sélectionner plusieurs objets :
  - Utilisez le "Pointeur de sélection" de la boîte à outils et encadrez les objets à sélectionner **ou**
  - Sélectionnez le premier objet et enfoncez la touche Maj ou Ctrl pour sélectionner les autres.

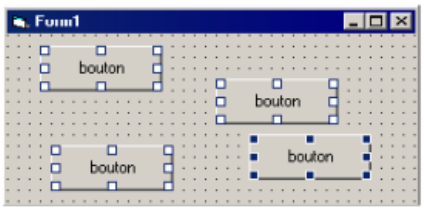
  Lorsque plusieurs contrôles sont sélectionnés, celui dont les poignées de dimension sont en couleur (le dernier sélectionné), spécifie les caractéristiques des autres lors d'une opération commune.



Exemple : Le bouton Command4 a des poignées de couleur.

La propriété Caption de Command4 est modifiée et l'opération suivante est effectuée : menu Format /Uniformiser la taille, /Les deux. Observez le nouvel

aspect des boutons dans la fenêtre qui suit. Ils ont le *même titre* et ils sont tous de la *même taille*, fixée par celle du bouton Command4.



En utilisant la sélection multiple, il est possible de modifier rapidement une propriété commune aux contrôles sélectionnés. Par exemple, les polices de caractères de plusieurs objets (propriété Font) peuvent être modifiées en une seule opération.

Suppression d'un contrôle

Pour supprimer un ou plusieurs contrôles, sélectionnez-les et pressez la touche *Suppr*. Les contrôles effacés sont récupérables par Édition /Annuler (Ctrl+Z). *Après avoir exécuté le projet, aucune récupération n'est possible.*

Déplacement et ajustement de la taille des contrôles

Pour *déplacer* un ou plusieurs contrôles sélectionnés, glissez-les sur la feuille avec la souris ou, enfoncez la touche Ctrl et utilisez les flèches de direction du clavier pour déplacer dans la direction désirée.

Pour *ajuster la taille* d'un contrôle ou d'une feuille, placez le pointeur de la souris sur l'une des poignés de dimension de l'objet. Lorsque le pointeur prend l'apparence d'une flèche double (↔), glissez jusqu'à la largeur ou la hauteur souhaitée. Les poignées situées dans les coins ajustent la taille en largeur et en hauteur.

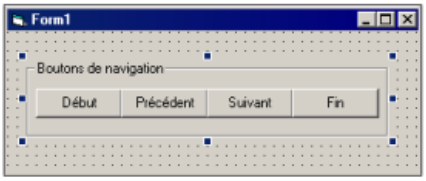
Pour *ajuster la taille de plusieurs contrôles sélectionnés*, enfoncez la touche *Maj* et utilisez les flèches de direction du clavier pour ajuster la largeur ou la hauteur.

Le menu Format, décrit dans ce chapitre, offre d'autres options pour déplacer et ajuster la taille des contrôles (section Le menu Format, page 56).

Regroupement de contrôles

Pour regrouper des contrôles, on utilise un **conteneur**. Un conteneur est un objet qui en contient d'autres. Le conteneur le plus utilisé est le contrôle **Frame** de la boîte à outils. L'objet Form ainsi que les contrôles à onglets (TabStrip, SSTab) sont aussi des conteneurs, puisqu'ils contiennent d'autres contrôles.

Le regroupement de contrôles permet d'améliorer la présentation des objets dans une interface utilisateur. Une fois regroupés, les contrôles se déplacent facilement, en sélectionnant seulement le conteneur. Dans la feuille qui suit, un cadre contient 4 boutons de commande. En déplaçant le cadre sélectionné, tous les boutons seront déplacés.

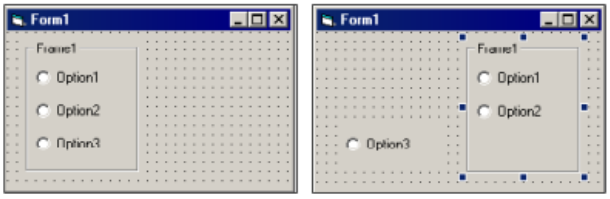


- Pour ajouter *un nouveau contrôle* dans un conteneur, cliquez sur le contrôle dans la boîte à outils et dessinez-le dans le conteneur. *Il ne faut pas double-cliquer dans la boîte à outils, car le contrôle ne sera pas inclus dans le conteneur.* L'utilisation de cette méthode est une source d'erreur courante.
- Pour ajouter un contrôle existant dans un conteneur, coupez-le, sélectionnez le conteneur et collez le contrôle dans le conteneur.

Afin de vous assurer que des contrôles sont bien inclus dans un conteneur, déplacez le conteneur. Les contrôles qui ne sont pas inclus, ne seront pas déplacés.

Exemple d'une erreur courante

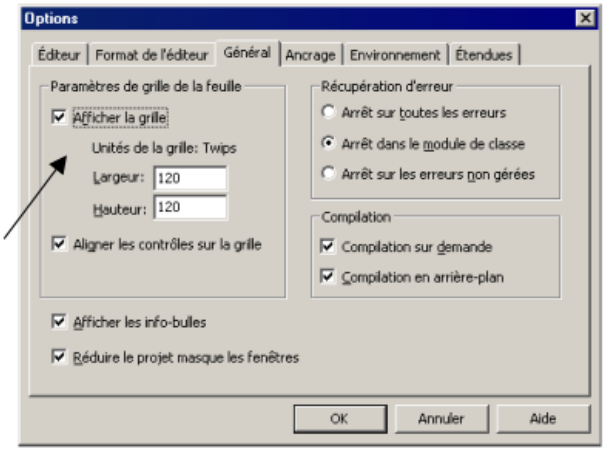
Dans la feuille de droite, illustrée ci-après, le déplacement du conteneur Frame1, nous permet de constater que le bouton Option3 n'est pas inclus dans le cadre.



Ce genre d'erreur dans la conception d'une interface graphique peut causer quelques désagréments lors de la programmation. Par exemple, par une seule instruction (`Frame1.Visible = False`), tous les contrôles peuvent être masqués à l'utilisateur. Les contrôles non inclus dans le conteneur ne seront pas masqués.

La grille de points

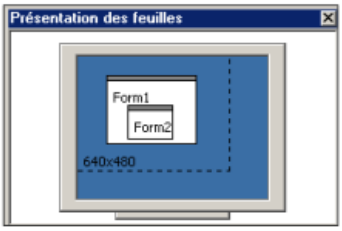
Une **grille de points** apparaît dans la fenêtre de conception pour faciliter l'alignement des contrôles qui y seront ajoutés en mode [création]. Celle-ci peut être modifiée ou supprimée, en sélectionnant au menu, Outils /Options /onglet Général.



2.5 LA FENÊTRE PRÉSENTATION DES FEUILLES

La fenêtre Présentation des feuilles permet de placer visuellement les feuilles d'un projet, les unes par rapport aux autres, fixant ainsi la position de chacune au démarrage du projet. Pour afficher cette fenêtre, sélectionnez au menu

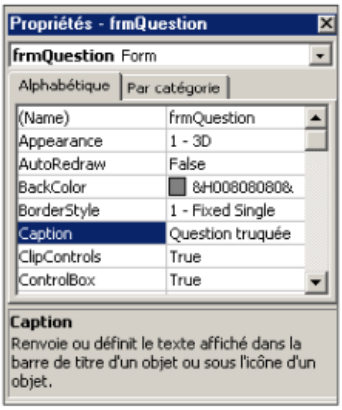
Affichage/Fenêtre Présentation des feuilles ou l'icône correspondante de la barre d'outils Standard .



Malgré l'utilité de cette fenêtre, elle n'est pas souvent utilisée dans la conception d'un projet. Il est donc judicieux de fermer cette fenêtre, afin de bénéficier de plus d'espace de travail pour la Fenêtre des propriétés. Dans la plupart des projets de ce volume, la propriété **StartUpPosition** de l'objet Form est suffisante pour indiquer la position d'une fenêtre au démarrage.

2.6 LA FENÊTRE PROPRIÉTÉS

La fenêtre Propriétés est affichée en sélectionnant au menu Affichage /Fenêtre Propriétés (F4) ou par l'icône correspondante de la barre d'outils Standard .

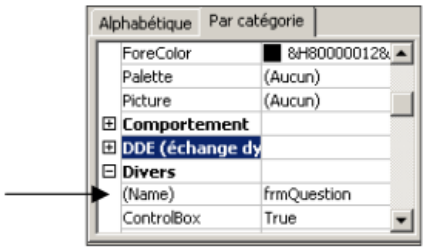


La fenêtre Propriétés permet, en mode [création], d'observer et de modifier les propriétés d'un *objet sélectionné*. Un objet peut être sélectionné directement dans la

fenêtre de conception de feuille ou à partir de la **Liste de sélection des objets**, située dans le haut de la fenêtre Propriétés.

Dans la colonne de *gauche*, on retrouve le nom des propriétés et dans la colonne de *droite*, la valeur de chacune. Les propriétés sont affichées en ordre "Alphabétique" ou "Par catégorie" selon l'onglet sélectionné.

La *liste par catégorie* est généralement plus pratique parce qu'elle permet de masquer certaines catégories de propriétés qui ne sont pas utilisées, réduisant ainsi le nombre de lignes affichées dans la fenêtre. Par exemple, dans la figure qui suit, les propriétés des catégories Comportement et DDE sont masquées temporairement. Notez que la propriété (Name) n'apparaît plus en premier, mais qu'elle fait partie de la catégorie Divers.



Modification d'une propriété

- Tapez la nouvelle valeur dans la colonne de droite, ou

- Cliquez sur une flèche à droite de la propriété pour faire un choix, ou

- Cliquez sur … pour accéder à une boîte de dialogue.

  Pour la plupart des contrôles ActiveX qui sont ajoutés dans la boîte à outils, la propriété (Personnalisé) permet d'accéder à une fenêtre comprenant l'ensemble des propriétés du contrôle, facilitant ainsi la modification de celles-ci.

Dans le *bas* de le fenêtre Propriétés, une courte description de la propriété sélectionnée est affichée. Pour masquer cette description, décochez le choix Description du menu contextuel.

Le choix **Ancrable** du menu contextuel des fenêtres Propriétés, Boîte à outils, Explorateur de projets et Présentation des feuilles spécifie que la position de la fenêtre est *fixe* à l'intérieur de la fenêtre principale. Pour placer l'une de ces fenêtres à un endroit qui vous convient mieux, décochez le choix Ancrable.

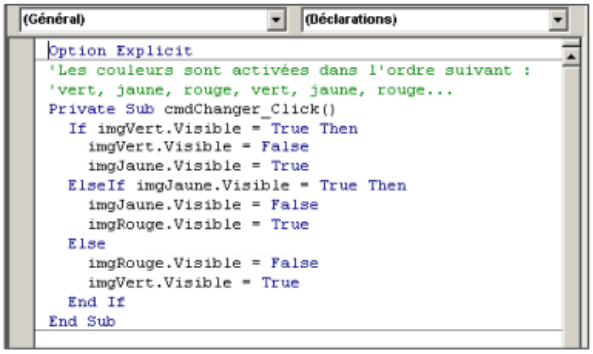
2.7 LA FENÊTRE DE CODE

Il existe plusieurs façons d'accéder à la fenêtre de code, passant ainsi du mode de conception graphique au **mode de programmation** :

- Sélectionnez au menu Affichage /Code, ou
- Pressez la touche F7, ou
- Double-cliquez sur un objet à programmer (feuille ou contrôle) dans la fenêtre de conception de feuille, ou
- Utilisez le bouton Code de la fenêtre Explorateur de projets.

Pour basculer de la fenêtre de code à la fenêtre de conception de feuille, pressez Maj+F7 ou utilisez le bouton "Afficher l'objet" de la fenêtre Explorateur de projets.

La fenêtre de code, appelée aussi **module de code**, contient le code de programmation lié à une feuille du projet.



Les listes Objet et Procédure

Dans le *haut* de la fenêtre de code, on retrouve 2 zones de liste, la liste **Objet** à gauche et la liste **Procédure** à droite.

Liste Objet Liste Procédure

La liste **Objet** permet de *sélectionner l'un des objets du projet* et la liste **Procédure** permet de *sélectionner un événement associé à l'objet*, dans le but de **créer** ou de **retrouver** une *procédure événementielle*. Lorsque le choix (Général) est sélectionné dans la liste de gauche, le choix (Déclarations) est disponible dans la liste de droite, ainsi que les *procédures personnalisées* qui ont été ajoutées au projet.

Pour plus de détails, consultez la section Déclaration et appel d'une procédure personnalisée *Sub*, page 235.

Le choix (Déclarations) permet de déclarer ou de retrouver des éléments dans la section (Général) du module de code située au début de celui-ci.

Dans un module de code plus ou moins volumineux, l'utilisation des listes Objet et Procédure permet d'accéder directement à une procédure ou à une déclaration déjà codée, vous permettant de gagner un temps appréciable.

Lignes graphiques et couleurs

Les déclarations apparaissent au *début* de la fenêtre de code (exemple : Option Explicit). Elles sont suivies de l'ensemble des procédures. Pour plus de clarté, une

ligne graphique est automatiquement ajoutée pour séparer les déclarations des procédures et chacune des procédures. L'éditeur de Visual Basic utilise le *bleu* pour les mots clé (If, Private, True), le *noir* pour le texte normal, le *vert* pour les commentaires et le *rouge* pour indiquer une erreur de syntaxe. Notez que les couleurs peuvent être modifiées en sélectionnant au menu, Outils /Options /onglet Format de l'éditeur.

Les minuscules et les majuscules

Dans le code de programmation, l'éditeur de Visual Basic ne fait aucune distinction entre les *minuscules* et les *majuscules*. Par exemple, les mots PRiVATe et priVaTE sont équivalents. Une fois que l'entrée d'une ligne ou d'une instruction est complétée, les mots sont réaffichés comme ils ont été déclarés ou nommés.

Exemple cmdeffacer devient cmdEffacer ;

PRIVATE devient Private ;

mstrnomconjoint devient mstrNomConjoint.

Pour plus d'efficacité, tapez entièrement le code de programmation en *minuscules*, sauf les déclarations.

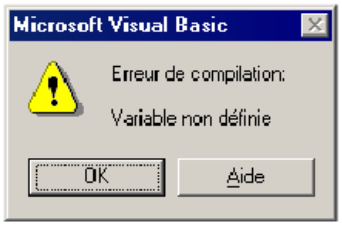
Les erreurs de syntaxe

Lors de la saisie du code, *l'interpréteur* de Visual Basic vérifie chacune des lignes saisies et signale la présence **d'erreurs de syntaxe**. Une erreur de syntaxe est un manquement aux règles de syntaxe du langage Visual Basic (exemple : oubli du mot Then dans une instruction If). En présence d'une erreur, la ligne est réaffichée en *rouge* et un message est affiché.

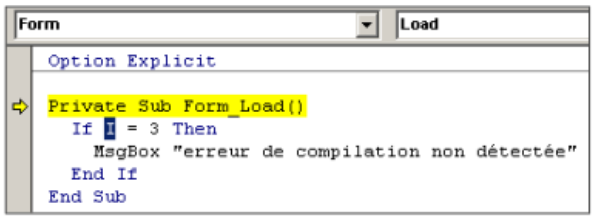
Le bouton Aide (F1) pointe directement à une rubrique de l'aide MSDN. En cliquant sur le bouton OK, le curseur est placé sur la ligne en erreur. Toute erreur de syntaxe doit être corrigée avant d'exécuter un projet.

Les erreurs de compilation

Lorsqu'un programme est exécuté (bouton Exécuter ou F5), celui-ci est d'abord compilé. La **compilation** d'un programme consiste à traduire le code source en code exécutable. Toutes les lignes de code sont alors vérifiées, les unes par rapport aux autres et des **erreurs de compilation** peuvent être détectées. Par exemple, une variable utilisée dans le programme n'a pas été déclarée. Comme les erreurs de syntaxe, les erreurs de compilation sont aussi signalées par un message.

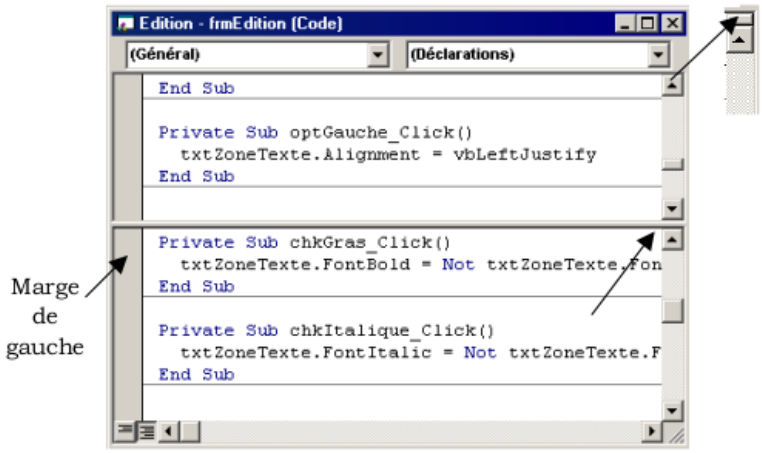


Une flèche dans la marge de gauche pointe la procédure en erreur et la variable I est sélectionnée. L'erreur doit être corrigée avant de continuer l'exécution (bouton Continuer ou F5).



Fractionnement de la fenêtre de code

Il est possible de fractionner la fenêtre de code en deux parties horizontales, en glissant le **Séparateur de fenêtre** vers le bas. Le séparateur de fenêtre est situé dans le *haut* de la barre de défilement verticale, à *droite* de la fenêtre de code.



Les icônes dans la marge de gauche

Dans la *marge de gauche* de la fenêtre de code, des icônes peuvent s'y trouver :

Icône	Fonction	Description
	Point d'arrêt	En exécution, un point d'arrêt permet d'interrompre celle-ci, à une ligne de code précise. Cliquez dans la marge pour ajouter un point d'arrêt et cliquez de nouveau pour le supprimer.
	Signet	Les signets sont des points de repère qui permettent de se déplacer plus rapidement dans une fenêtre de code volumineuse.
	Ligne en erreur Instruction suivante	Une flèche jaune signale une erreur de compilation ou une erreur d'exécution. En mode Pas à pas détaillé (F8), il est possible de définir la prochaine instruction à être exécutée.

Les boutons Module et Procédure

Dans le *coin inférieur gauche* de la fenêtre de code, on retrouve deux boutons très utiles :

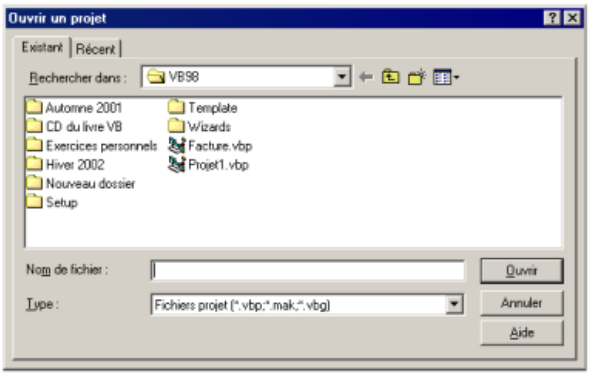
Icône	Nom du bouton	Fonction du bouton
	Affichage complet du module	Affiche tout le code du module.
	Affichage de procédure	Affiche le code d'une seule procédure sélectionnée.

2.8 LA BARRE DE MENUS

Le menu Fichier

Le choix **Nouveau projet** (Ctrl+N) affiche la boîte de dialogue "Nouveau projet", qui permet de débuter un autre projet au cours d'une session de travail. Il existe plusieurs types de projets, mais le choix le plus courant est un projet de type EXE Standard. Le projet en cours doit être enregistré avant d'en créer un nouveau, puisqu'il sera déchargé de la mémoire.

Le choix **Ouvrir un projet** (Ctrl+O) affiche la boîte de dialogue "Ouvrir un projet", qui permet de charger en mémoire un projet enregistré.

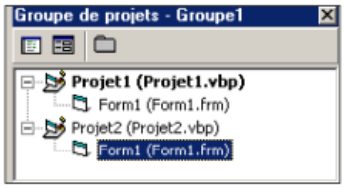


L'onglet **Existant** permet de sélectionner tout projet Visual Basic tandis que l'onglet **Récent** affiche les derniers projets ouverts. La zone "Rechercher dans" permet d'accéder à la structure arborescente des dossiers, ainsi que les boutons suivants :

Icône	Nom du bouton	Fonction du bouton
	Remonter d'un niveau	Affiche le dossier situé au niveau supérieur du dossier courant.
	Créer un nouveau dossier	Crée un nouveau dossier sous le dossier courant.
	Menu Affichage	Affiche un menu contextuel qui permet de modifier les paramètres d'affichage des dossiers et fichiers.

- Bouton équivalent de la barre d'outils Standard :

Le choix **Ajouter un projet** affiche la boîte de dialogue "Ajouter un projet" qui comporte trois onglets : Nouveau, Existant et Récent. En ajoutant un projet, un **groupe de projets** sera alors créé (exemple : Groupe1.vbg).



- Bouton équivalent de la barre d'outils Standard :

Le choix **Supprimer le projet** décharge de la mémoire un projet sélectionné après une confirmation d'enregistrement des dernières modifications.

Le choix **Enregistrer le projet (.vbp)** enregistre le projet en cours et toutes ses composantes, dans un dossier sélectionné avec un nom de fichier spécifié dans la boîte de dialogue "Enregistrer le fichier sous". Cette boîte de dialogue est affichée lors du premier enregistrement du projet ou lorsque le choix **Enregistrer le projet sous (.vbp)** est sélectionné. Lorsqu'un *groupe de projets est ouvert*, les choix suivants sont disponibles : **Enregistrer le groupe de projets (.vbg)**, **Enregistrer le groupe de projets sous (.vbg)**.

- Bouton équivalent de la barre d'outils Standard :

Le choix **Enregistrer *nom de la feuille* (.frm)** (Ctrl+S) enregistre la feuille sélectionnée, dans un dossier nommé avec un nom de fichier spécifié dans la boîte de dialogue "Enregistrer le fichier sous". Cette boîte de dialogue est affichée lors du premier enregistrement de la feuille ou lorsque le choix **Enregistrer *nom de la feuille* sous (.frm)** est sélectionné.

Créez un *nouveau dossier* pour chaque projet Visual Basic.

Pour tout nouveau projet, *nommez chacune des composantes* de celui-ci avec la propriété *Name* (projet et feuille). Par la suite, *enregistrez individuellement* la feuille et le projet.

Une fois qu'un projet a été enregistré, utilisez le bouton Enregistrer de la barre d'outils Standard pour enregistrer les dernières modifications de toutes les composantes du projet.

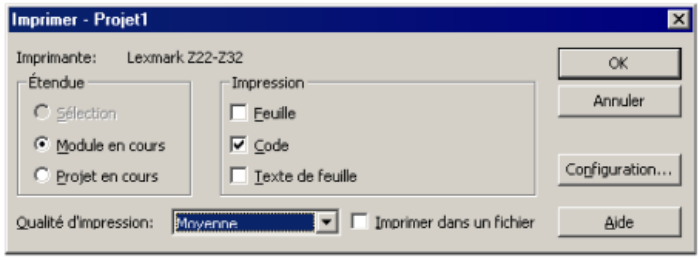
Attention !

Visual Basic enregistre dans le fichier **.vbp** du projet, le chemin d'accès du fichier dans lequel les feuilles ont été enregistrées (**.frm**). Lorsqu'un projet est déplacé dans un autre dossier ou qu'un dossier est renommé ou encore en transportant un projet sur un autre ordinateur (école/maison), il arrive que lors de l'ouverture du projet, le message d'erreur suivant soit affiché :

La forme n'a pas été trouvée, continuer à charger le projet ?

Dans ce cas, répondez **Oui** pour continuer à charger le projet, et par la suite, sélectionnez au menu **Projet/Ajouter une feuille /onglet Existant**. Spécifiez le nouveau chemin d'accès du fichier dans lequel la forme est enregistrée.

Le choix **Imprimer** (Ctrl+P) ouvre la boîte de dialogue Imprimer.



La zone **Étendue** spécifie s'il s'agit d'une partie du code sélectionnée (Sélection), ou de la feuille en cours (Module en cours) ou de toutes les feuilles du projet (Projet en cours).

La zone **Impression** permet de fixer jusqu'à trois choix :

Feuille	Impression de l'interface graphique utilisateur.
Code	Impression du code de programmation.
Texte de feuille	Impression des propriétés de la feuille et des propriétés des objets qu'elle contient.

Le choix **Configuration de l'impression** ouvre la boîte de dialogue de Windows "Configuration de l'impression", dans laquelle des paramètres peuvent être modifiés.

Le choix **Créer *nom du projet*.exe** ouvre la boîte dialogue "Créer le projet" qui permet de spécifier un dossier et un nom de fichier. En cliquant sur le bouton OK, un **fichier exécutable (.exe)** du projet est créé. Le bouton Options permet de fixer certaines propriétés du projet comme par exemple, le numéro de la version, le titre de l'application et l'icône associée.

Dans le bas du menu Fichier, les derniers projets ouverts sont affichés. Le choix Quitter (Alt+Q) ferme l'environnement de développement de Visual Basic après une confirmation d'enregistrement des dernières modifications.

Le menu Édition

Le menu **Édition** offre des choix utilisables lors de la conception de l'interface utilisateur ou lors de l'édition du code de programmation.

Les choix **Annuler/Refaire** permettent d'annuler ou de refaire les dernières opérations qui ont été effectuées. Utilisez le raccourci clavier Ctrl+Z pour annuler.

- Boutons Annuler et Refaire de la barre d'outils Standard :

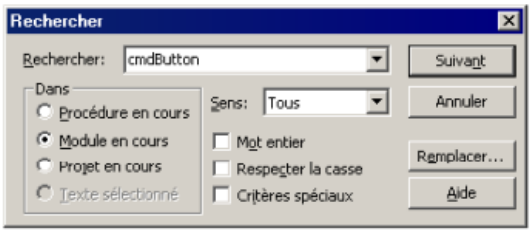
Les choix **Couper/Copier/Coller** permettent les opérations correspondantes par l'intermédiaire du Presse-Papiers de Windows. Utilisez les raccourcis clavier Ctrl+X, Ctrl+C et Ctrl+V.

Le choix **Supprimer** (touche Suppr) permet de supprimer un contrôle dans la fenêtre de conception de feuille ou un texte sélectionné dans la fenêtre de code.

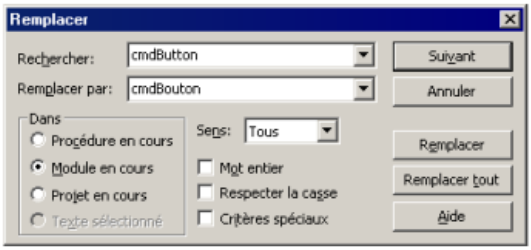
La combinaison des touches Ctrl+Y *supprime une ligne* dans la fenêtre de code.

Le choix **Tout sélectionner** sélectionne tous les contrôles de la fenêtre de conception de feuille ou tout le texte du module de code. Utilisez le raccourci clavier Ctrl+A.

Les choix **Rechercher/Suivant/Remplacer** permettent la recherche et le remplacement de chaînes de caractères dans le code de programmation.



La chaîne cmdButton est recherchée dans le module de code en cours. Le bouton Suivant spécifie de rechercher l'occurrence suivante de la chaîne cmdButton. Le bouton Remplacer affiche la boîte de dialogue Remplacer.



La chaîne cmdBouton est spécifiée comme *chaîne de remplacement* de la chaîne cmdButton.

- Bouton équivalent de la barre d'outils Standard :

Les choix **Mettre en retrait/Retrait négatif** permettent la mise en retrait de lignes de code sélectionnées, *afin d'améliorer la lisibilité du code de programmation*. Utilisez les touches correspondantes Tab et Maj+Tab.

Quelques normes de programmation concernant les retraits

- Les déclarations et instructions d'une procédure doivent être mises en retrait et les mots End Sub doivent être alignés avec le début de la procédure.

Exemple

```
Private Sub Form_Load()
  déclaration
  instrution
  instrution
End Sub
```

- Dans une structure comme l'instruction If, les instructions doivent être mises en retrait et les mots clés comme If, Else, EsleIF et End If doivent être alignés. Quelques autres structures : Select Case, With, For/Next, Do/Loop et While/Wend.

Exemple

```
Private Sub cmdChanger_Click()
  If imgVert.Visible = True Then
     imgVert.Visible = False
     imgJaune.Visible = True
  ElseIf imgJaune.Visible = True Then
     imgJaune.Visible = False
     imgRouge.Visible = True
  Else
     imgRouge.Visible = False
     imgVert.Visible = True
  End If
End Sub
```

Pour d'autres exemples de mises en retrait dans les structures, consultez les exemples des sections L'instruction If page 261, L'instruction Select Case page 265, 5.14 (Les structures répétitives) et 5.15 (With).

- Lorsqu'une instruction est répartie sur plusieurs lignes, décalez le texte de plusieurs retraits, à partir de la deuxième ligne, afin de dégager le mot clé de l'instruction ou la fonction utilisée.

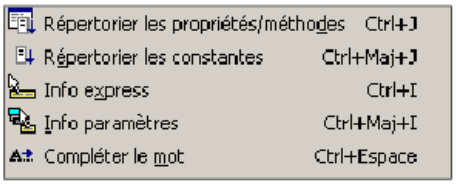
Exemple : dégagement du mot clé MsgBox

```
Private Sub cmdOui_Click()
  MsgBox "...à la condition de faire TOUS les exercices!", _
            vbExclamation, "Oui..."
End Sub
```

- Boutons Mettre en retrait et Retrait négatif de la barre d'outils Édition :

Le choix **Insérer un fichier** ajoute le texte du fichier spécifié, à la position courante du curseur de la fenêtre de code.

Les 5 choix suivants sont très utiles lors de *l'édition* du code de programmation :



- **Répertorier les propriétés/méthodes** (Ctrl+J) affiche une liste des propriétés et des méthodes d'un objet. Cette liste est la même que celle qui est affichée, au moment où un *point* est tapé à la suite du nom d'un objet.
- **Répertorier les constantes** (Ctrl+Maj+J) affiche une liste des constantes disponibles pour une propriété ou les arguments possibles pour une fonction dans une info-bulle. Les mêmes informations sont affichées lorsque le *signe* = est tapé à la suite d'une propriété ou lorsque le nom d'une fonction est saisi.
- **Info express** (Ctrl+I) affiche dans une info-bulle, la syntaxe d'une variable, fonction, instruction, méthode ou procédure sélectionnée.

- **Info paramètres** (Ctrl+Maj+I) donne des informations sur l'argument courant (illustré en gras), à l'intérieur d'une instruction ou d'une fonction.
- **Compléter le mot** (Ctrl+Espace) affiche automatiquement la suite d'un mot que Visual Basic reconnaît. Cette fonction de l'éditeur de code est *l'une des plus utiles*.

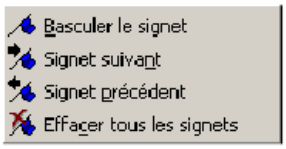
Comment entrer une ligne de code rapidement

Exemple : imgFigure.Visible = True

- Tapez img et pressez Ctrl+Espace ;
- Sélectionnez le choix imgFigure dans la liste par ↓ ;
- Tapez le point et sélectionnez le choix Visible dans la liste ;
- Tapez le signe = et sélectionnez le choix True ;
- Pressez la touche Enter.

- Les boutons équivalents sur la barre d'outils Édition :

Le choix **Signets** permet une navigation plus rapide à l'aide de points de repère dans la fenêtre de code d'un programme volumineux. La présence d'un signet est signalée par l'icône affichée dans la marge de gauche de la fenêtre de code.



Basculer le signet	Ajout ou suppression d'un signet.
Signet suivant	Déplacement vers le signet suivant.
Signet précédent	Déplacement vers le signet précédent.
Effacer tous les signets	Suppression de tous les signets.

- Les boutons équivalents sur la barre d'outils Édition :

Sur la barre d'outils Édition, on retrouve deux boutons *très utiles* qui étrangement, n'apparaissent pas au menu Édition :

	Le bouton **Commenter le bloc** permet de mettre en commentaire des lignes de code sélectionnées. Une apostrophe est ajoutée au début de chaque ligne.
	Le bouton **Ne pas commenter le bloc** supprime l'apostrophe au début des lignes de code sélectionnées.

Le menu Affichage

Plusieurs choix du menu Affichage servent à afficher ou activer des fenêtres de l'environnement de développement :

	Code	F7
	Objet	Maj+F7
	Explorateur d'objets	F2
	Explorateur de projets	Ctrl+R
	Propriétés	F4
	Présentation des feuilles	
	Boîte à outils	

La plupart de ces fenêtres ont été traitées au début de ce chapitre. L'Explorateur d'objets est détaillé dans la section 1.6 L'Explorateur d'objets.

Le choix **Définition** (Maj+F2) affiche l'emplacement de la fenêtre de code où est déclarée une variable ou une procédure sur laquelle le curseur pointe son nom.

Le choix **Dernier emplacement** (Ctrl+Maj+F2) donne la possibilité de revenir sur une ligne de code qui a été consultée ou modifiée. Les 8 dernières consultations ou modifications sont accessibles.

Le choix **Fenêtre Exécution** (Ctrl+G) affiche la fenêtre Exécution utilisée pour tester des parties de code. À partir de cette fenêtre, il est possible de consulter et de modifier les valeurs des variables et des propriétés. Il est aussi possible de forcer l'exécution d'instructions en les tapant directement dans la fenêtre Exécution.

- Bouton équivalent de la barre d'outils Débogage :

Le choix **Fenêtre Variables locales** affiche la fenêtre "Variables locales", qui contient une liste de toutes les variables de la procédure en cours, avec leurs valeurs.

- Bouton équivalent de la barre d'outils Débogage:

Le choix **Fenêtre Espions** affiche la fenêtre Espions, qui contient les espions qui ont été définis par Débogage/Ajouter un espion. Un **espion** permet de surveiller étroitement une variable, une propriété ou une expression VB valide. Par exemple, en définissant la variable I comme espion, à chaque fois que sa valeur change, sa nouvelle valeur s'affiche dans la fenêtre Espions.

- Bouton équivalent de la barre d'outils Débogage :

Le choix **Pile des appels** (Ctrl+L) affiche la boîte de dialogue Appels, qui énumère les procédures appelées du projet en cours, qui n'ont pas encore été exécutées. Les dernières procédures appelées apparaissent dans le haut de la liste, et les premières dans le bas de la liste.

- Bouton équivalent de la barre d'outils Débogage :

Le choix **Pages de propriétés** (Maj+F4) affiche la fenêtre "Pages de propriétés" d'un contrôle ActiveX ayant la propriété (Personnalisé). Cette option équivaut à sélectionner la propriété dans la fenêtre Propriétés du contrôle.

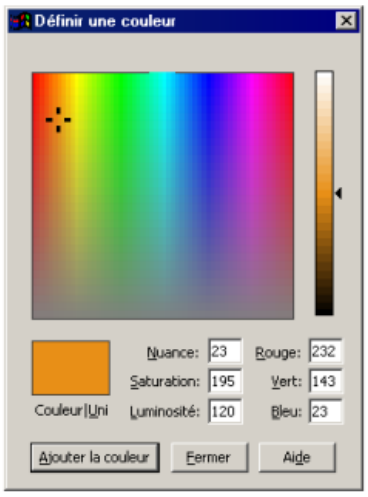
Le choix **Palette de couleurs** affiche la fenêtre suivante :

Le carré, dans le coin supérieur gauche, permet de modifier la couleur de l'arrière-plan d'un objet en cliquant sur la partie *grise* du carré. La couleur du texte d'un objet est modifiée en cliquant sur la partie *noire* du carré. La *partie externe* du carré représente donc la *couleur de l'arrière-plan* et la *partie interne* représente la *couleur du texte* d'un objet sélectionné.

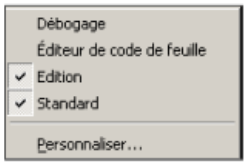
Le bouton "Par défaut" permet d'attribuer les *couleurs système* de Windows à un objet. Les couleurs choisies de cette façon seront harmonisées avec celles du choix Affichage du Panneau de configuration.

Pour *créer une nouvelle couleur*, sélectionnez un *carré blanc* dans le bas de la palette de couleurs et pressez le bouton "Sél. couleurs", pour afficher la fenêtre

"Définir une couleur". Cliquez dans la zone des couleurs sur le ton désiré et par la suite, sur le bouton "Ajouter la couleur" pour que celle-ci devienne disponible dans la palette de couleurs. Observez les valeurs 0 à 255 des zones Rouge, Vert et Bleu. La combinaison de ses trois couleurs permet de définir toutes les couleurs.



Le choix **Barre d'outils** permet d'afficher d'autres barres d'outils utiles, en plus de la barre d'outils Standard. Le choix Personnaliser affiche la boîte de dialogue Personnaliser qui permet de créer et de modifier des barres d'outils et de changer la barre de menus.



Ajoutez la barre d'outils Édition à votre environnement ; elle vous sera très utile pour commenter ou ne pas commenter des blocs de lignes de code.

Le menu illustré précédemment est aussi disponible en cliquant avec le bouton droit de la souris dans une zone libre de la barre d'outils.

Le menu Projet

Le choix **Ajouter une feuille** affiche la boîte de dialogue "Ajouter une feuille", qui permet d'ajouter une nouvelle feuille ou une feuille existante au projet.

- Le bouton "Ajouter une feuille" de la barre d'outils Standard : permet d'ajouter les huit premiers choix du menu Projet.

Le choix **Ajouter une feuille MDI** affiche la boîte de dialogue "Ajouter une feuille MDI", qui permet d'ajouter une nouvelle feuille MDI ou une feuille MDI existante. Une seule feuille MDI peut être ajoutée à un projet.

Le choix **Ajouter un module** affiche la boîte de dialogue "Ajouter un module", qui permet d'ajouter un nouveau *module standard* ou existant au projet.

Le choix **Ajouter un module de classe** affiche la boîte de dialogue "Ajouter un module de classe", qui permet d'ajouter un nouveau *module de classe* ou existant au projet.

Consultez la section 5.1 Les modules.

Le choix **Ajouter un contrôle utilisateur** affiche la boîte de dialogue "Ajouter un contrôle utilisateur", qui permet d'ajouter un nouveau contrôle utilisateur ou existant au projet.

Consultez le Chapitre 8 Création d'un contrôle utilisateur ActiveX.

Le choix **Ajouter un fichier** (Ctrl+D) affiche la boîte de dialogue "Ajouter un fichier", qui permet d'ajouter plusieurs types de fichiers au projet. Par exemple, il peut s'agir d'une feuille (.frm) ou d'un module standard (.bas).

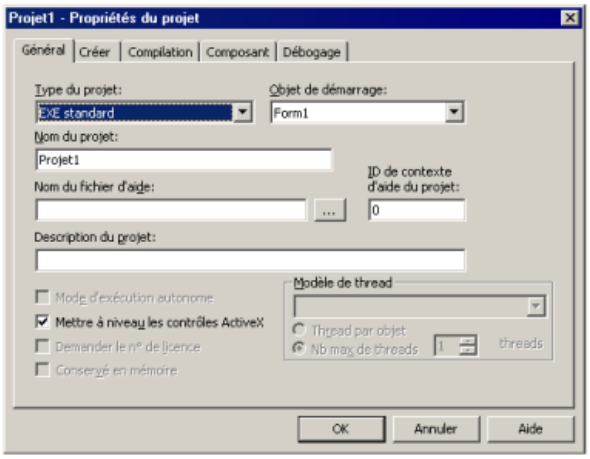
Le choix **Supprimer *nom d'un élément*** supprime du projet un élément sélectionné dans l'Explorateur de projets. Si cet élément a été modifié depuis le dernier enregistrement, un message de confirmation sera affiché. Le fichier n'est pas supprimé de son dossier, il est simplement enlevé du projet en cours.

Le choix **Références** affiche la boîte de dialogue Références, qui permet d'établir des références dans des bibliothèques d'objets de façon à rendre disponibles les objets appartenant à d'autres applications, dans un projet personnel. Une fois référencés, les objets sont affichés dans l'Explorateur d'objets et ils peuvent être utilisés dans le code de programmation.

Le choix **Composants** affiche la boîte de dialogue Composants, qui permet d'ajouter trois catégories de composants dans la boîte à outils : des **contrôles ActiveX**, des **concepteurs** qui seront ajoutés dans les choix du menu Projet (exemple : le concepteur Data Report) et des **objets insérables** (exemple : un document Microsoft Word).

Consultez la section 2.3 La boîte à outils et les composants.

Le choix **Propriétés de *nom du projet*** affiche la boîte de dialogue Propriétés, qui permet de modifier certaines caractéristiques du projet dont le type du projet, son nom, son icône, son numéro de version ainsi que **l'objet de démarrage**.



Consultez la section 5.3 L'objet de démarrage.

Le menu Format

Le menu **Format** offre un choix d'options utilisables lors de la conception d'une feuille, pour aligner les contrôles, les déplacer, les verrouiller, modifier leur taille et ajuster l'espacement entre eux.

Le choix **Aligner** aligne les contrôles sélectionnés en utilisant les caractéristiques du contrôle ayant les poignées de dimension de couleur. Ce choix offre plusieurs sous-choix.

- Le bouton équivalent de la barre d'outils Éditeur de code de feuille

Le choix **Uniformiser la taille** fixe la *même taille* aux contrôles sélectionnés, soit en largeur, soit en hauteur ou les deux, en utilisant les caractéristiques du contrôle ayant les poignées de dimension de couleur.

- Le bouton équivalent de la barre d'outils Éditeur de code de feuille

Le choix **Ajuster à la grille** ajuste la largeur et la hauteur du contrôle sélectionné sur les points de la grille les plus proches.

Les choix **Espacement horizontal** et **Espacement vertical** modifie l'espacement entre les contrôles sélectionnés, horizontalement ou verticalement. Ces choix offrent les sous-choix suivants : Égaliser, Augmenter, Diminuer et Supprimer.

Le choix **Centrer sur la feuille** centre les contrôles sélectionnés par rapport à la feuille, horizontalement ou verticalement.

- Le bouton équivalent de la barre d'outils Éditeur de code de feuille

Le choix **Plan** place les contrôles sélectionnés au premier plan de la feuille (Ctrl+J, devant les autres contrôles) ou à l'arrière-plan de la feuille (Ctrl+K, derrière les autres contrôles).

- Les boutons "Mettre au premier plan" et "Envoyer à l'arrière-plan" de la barre d'outils Éditeur de code de feuille

Le choix **Verrouiller les contrôles** verrouille tous les contrôles d'une feuille à leur position respective afin de ne pas les déplacer par inadvertance. Pour déverrouiller, sélectionnez ce choix de nouveau.

- Le bouton équivalent de la barre d'outils Éditeur de code de feuille

Le menu Exécution

Le choix **Exécuter** (F5) lance l'exécution du projet à partir du mode [création]. Une fois que le mode [exécution] est activé, les choix Arrêt et Fin deviennent alors disponibles. En mode [arrêt], le choix Exécuter du menu devient le choix **Continuer** (F5).

- Bouton équivalent des barres d'outils Standard et Débogage

Le choix **Arrêt** (Ctrl+Arrêt) permet d'interrompre l'exécution du projet ; l'exécution peut reprendre avec le choix Continuer.

- Bouton équivalent des barres d'outils Standard et Débogage

Le choix **Fin** arrête définitivement l'exécution du programme. Le mode [création] est de nouveau activé.

- Bouton équivalent des barres d'outils Standard et Débogage

Le choix **Exécuter avec compilation complète** (Ctrl+F5) force la compilation de toutes les lignes de code avant l'exécution.

Le choix **Exécuter de nouveau** (Maj+F5) relance l'exécution du projet après une interruption causée par une erreur d'exécution, un point d'arrêt ou le choix Arrêt. L'exécution est reprise à partir du début.

Le menu Débogage

Le choix **Pas à pas détaillé** (F8) exécute le programme instruction par instruction, donnant ainsi la possibilité au programmeur de suivre d'une façon très détaillée l'exécution du projet. Il peut alors observer le contenu de chacune des variables au moyen des info-bulles qui sont affichées, lorsque le pointeur de la souris est placé sur l'une d'elles. En mode **Pas à pas**, la prochaine ligne à être exécutée est affichée en jaune et précédée d'une flèche dans la marge de gauche.

- Bouton équivalent de la barre d'outils Débogage

Le choix **Pas à pas principal** (Maj+F8) empêche l'exécution détaillée d'une procédure appelée. La procédure est exécutée mais non détaillée. Le mode "Pas à pas" étant parfois long, cette option permet de gagner un temps appréciable.

- Bouton équivalent de la barre d'outils Débogage

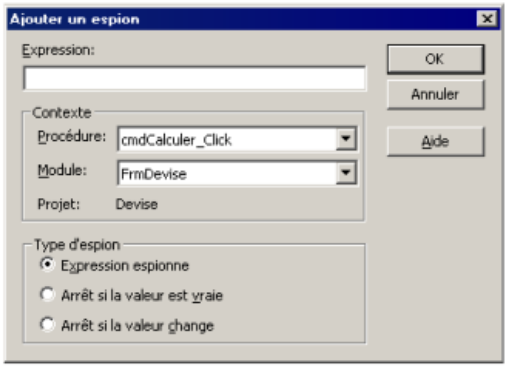
Le choix **Pas à pas sortant** (Ctrl+Maj+F8) permet d'interrompre le mode "Pas à pas" par une sortie de la procédure. Le reste de la procédure est exécuté mais non détaillé.

- Bouton équivalent de la barre d'outils Débogage

Le choix **Exécuter jusqu'au curseur** (Ctrl+F8) requiert de placer d'abord le curseur sur une instruction non exécutée ; l'exécution sera interrompue à la rencontre de cette instruction. Il est alors possible d'utiliser le mode "Pas à pas" pour la suite de l'exécution.

En combinant les quatre choix précédents, vous devriez rapidement venir à bout du bogue le plus coriace.

Le choix **Ajouter un espion** affiche la boîte de dialogue "Ajouter un espion" dans laquelle une *expression espionne* est saisie. Une expression espionne peut être une variable, une propriété, l'appel d'une fonction ou une expression VB valide. Un espion est défini dans le but de surveiller étroitement un comportement ou une valeur dans la fenêtre Espions.

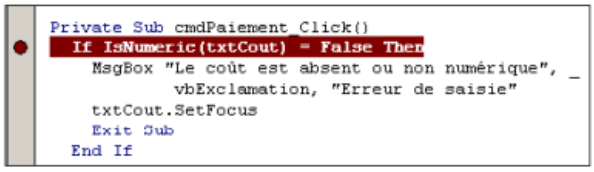


Le choix **Modifier un espion** affiche la boîte de dialogue "Modifier un espion", qui permet de modifier ou de supprimer des expressions espionnes qui ont été définies.

Le choix **Espion express (**Maj+F9) ouvre la boîte de dialogue "Espion Express" qui contient la valeur en cours de l'expression sélectionnée dans le code. Utilisez cette option pour les expressions espionnes qui n'ont pas été ajoutées dans la boîte de dialogue "Ajouter un espion".

- Bouton équivalent de la barre d'outils Débogage

Le choix **Basculer le point d'arrêt** (F9) permet d'ajouter ou de supprimer un point d'arrêt sur la ligne où se trouve le curseur. Lors de l'exécution du projet, un point d'arrêt permet d'interrompre celle-ci, à la ligne où un tel point est rencontré. Ce choix au menu équivaut à cliquer dans la marge de gauche d'une instruction exécutable.



À la rencontre d'un point d'arrêt, utilisez le mode "Pas à pas" pour observer attentivement la suite de l'exécution. En déclarant plusieurs points d'arrêt, la touche F5 peut être utilisée pour continuer l'exécution jusqu'au prochain point d'arrêt.

- Bouton équivalent de la barre d'outils Débogage

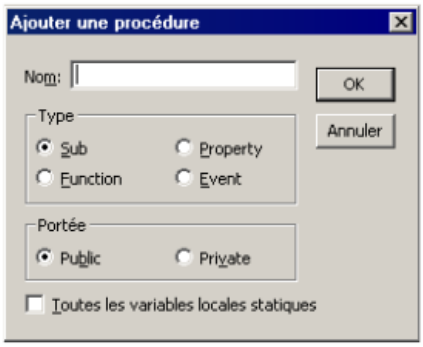
Le choix **Effacer tous les points d'arrêt** (Ctrl+Maj+F9) supprime tous les points d'arrêt qui ont été ajoutés.

Le choix **Définir l'instruction suivante** (Ctrl+F9) spécifie la prochaine ligne de code à être exécuter à l'intérieur d'une même procédure. Cette ligne est affichée en jaune et précédée d'une flèche dans la marge de gauche. Cette option permet d'ignorer des lignes de code ou d'exécuter plusieurs fois une même instruction.

Le choix **Afficher l'instruction suivante** permet de savoir quelle est la prochaine instruction qui devrait normalement être exécutée. Cette ligne est affichée en jaune et précédée d'une flèche dans la marge de gauche.

Le menu Outils

Le choix **Ajouter une procédure** ouvre la boîte de dialogue "Ajouter une procédure", qui permet de créer des *procédures personnalisées* et des *fonctions personnalisées*, ainsi que des propriétés et événements pour des contrôles utilisateur (contrôles ActiveX).



Déclaration et appel d'une procédure personnalisée *Sub*, page 235.

Le choix **Attributs de procédure** permet de modifier les caractéristiques (propriétés, méthodes, événements) des contrôles utilisateur.

Le choix **Créateur de menus** (Ctrl+E) permet d'ajouter, modifier ou supprimer un menu dans un projet. Ce choix est disponible lorsque la feuille de conception de feuille est activée.

- Bouton équivalent de la barre d'outils Standard

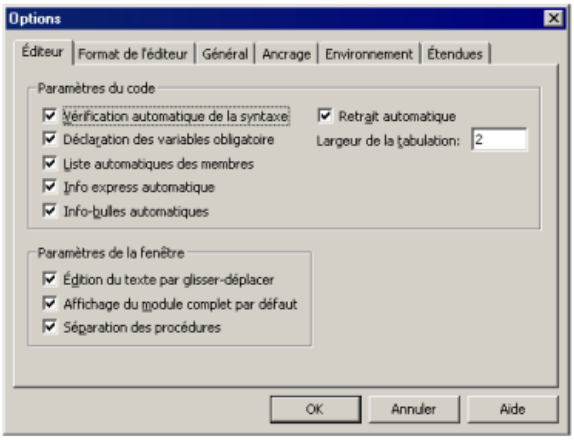
Consultez la section 4.7 Les menus.

Le choix **Options** donne accès à la boîte de dialogue Options qui permet de modifier les options par défaut de la configuration de l'environnement de développement.

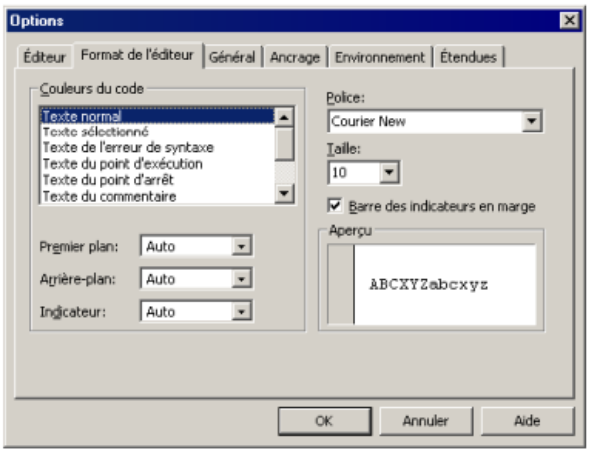
Quelques paramètres ont déjà été décrits dans le Chapitre 1 :

- Déclaration de variables obligatoire (onglet Éditeur) ;
- Largeur de la tabulation (onglet Éditeur) ;
- Invite d'enregistrement des modifications (onglet Environnement).

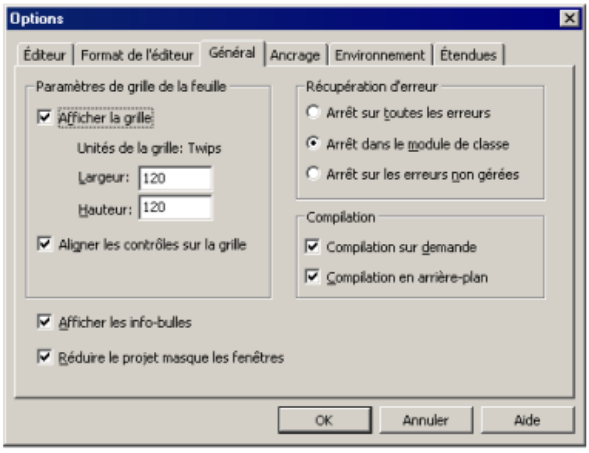
L'onglet **Éditeur** permet de modifier des paramètres liés à des fonctions automatiques de la fenêtre de code.



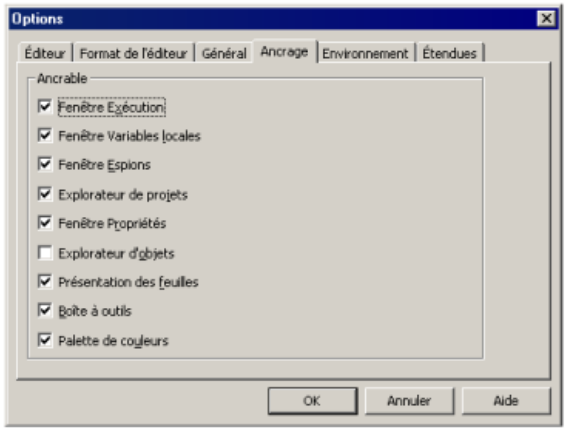
L'onglet **Format de l'éditeur** permet de changer l'apparence de la fenêtre de code (couleurs, police, taille des caractères).



L'onglet **Général** permet de modifier la grille de points de la fenêtre de conception de feuille ainsi que certains paramètres de récupération d'erreur, de compilation et d'affichage.

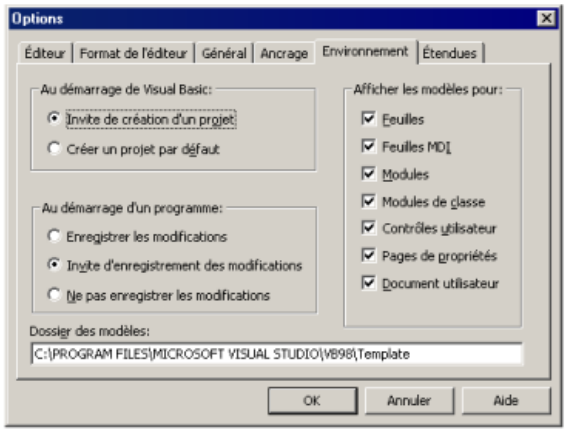


L'onglet **Ancrage** permet d'ancrer ou de désancrer les différentes fenêtres de l'environnement de développement.

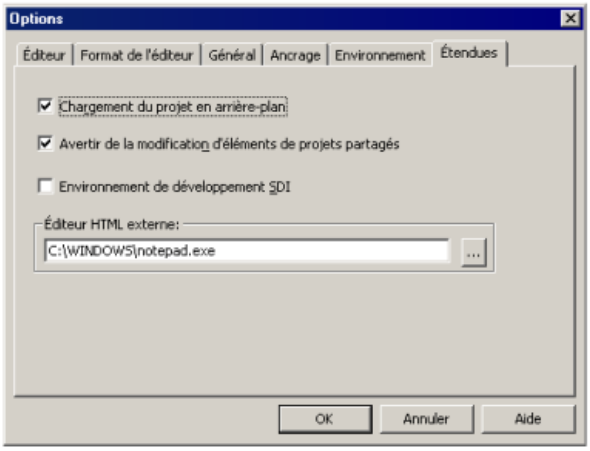


L'onglet **Environnement** spécifie des options au démarrage de Visual Basic et au démarrage de l'exécution des projets. Ces dernières options *sont très importantes pour l'enregistrement des modifications* apportées au projet.

Visual Basic disposent de plusieurs *modèles* qui sont affichés lorsque, par exemple, le choix suivant est sélectionné : Projet/Ajouter une feuille. À ce moment, un choix de modèles de feuilles est présenté. Décochez les cases de la partie de droite pour empêcher l'affichage de ces modèles.



L'onglet **Étendues** permet de sélectionner un environnement de type SDI plutôt que MDI et il est possible d'y choisir un éditeur HTML.

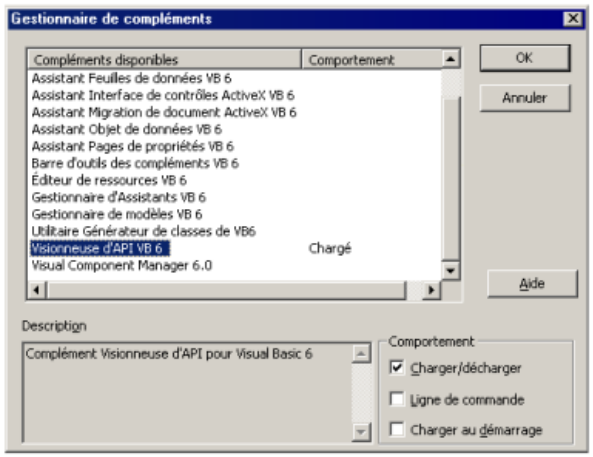


Le menu Compléments

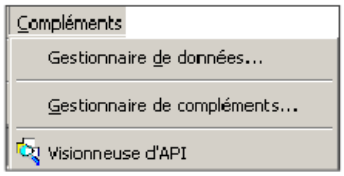
Par défaut, le menu Compléments compte deux choix : **Gestionnaire de données** et **Gestionnaire de compléments**. Ce menu est dynamique et au fur et à mesure que des compléments sont chargés à l'environnement, ceux-ci sont ajoutés au menu Compléments.

Le choix **Gestionnaire de données** ouvre une application de bases de données incluse dans Visual Basic appelée **VisData**. Dans les versions Professionnelle et Entreprise de Visual Basic 6.0, le *code source de cette application est fourni* dans les exemples. L'observation de ce code peut répondre à de nombreuses questions concernant la programmation des bases de données. Cette application a cependant été créée avec Visual Basic 5.0. Le Gestionnaire de données peut remplacer l'utilisation d'un logiciel comme Microsoft Access pour créer et gérer des bases de données avec une structure simple.

Le choix **Gestionnaire de compléments** affiche la boîte de dialogue "Gestionnaire de compléments".



Dans la fenêtre qui précède, le choix Visionneuse d'API VB 6 est sélectionné et la case Charger/Décharger est activée. La Visionneuse est donc chargée dans l'environnement de développement de Visual Basic et devient disponible dans le menu Compléments. Elle sera déchargée à la fermeture de Visual Basic.



Un complément est chargé jusqu'à la fermeture de Visual Basic (Charger/décharger), à moins de décocher la case pour le décharger immédiatement. L'option "Charger au démarrage" permet de le charger à chaque démarrage de Visual Basic.

Le menu Fenêtre

Comme dans la plupart des applications Windows, Visual Basic possède un menu **Fenêtre**. Celui-ci permet la réorganisation des fenêtres **Objet** (fenêtre de conception de feuille), **Code** et **Explorateur d'objets**, toutes situées dans le plan de travail. Les choix **Mosaïque horizontale**, **Mosaïque verticale** et **En cascade** sont très utiles lorsqu'un projet compte plusieurs feuilles.

Pour activer ou désactiver une fenêtre dans le plan de travail, cliquez sur celle-ci. Cependant, il arrive qu'une fenêtre ne soit pas visible. Dans ce cas, cochez la fenêtre à activer dans le menu Fenêtre où toutes les fenêtres ouvertes sont répertoriées.

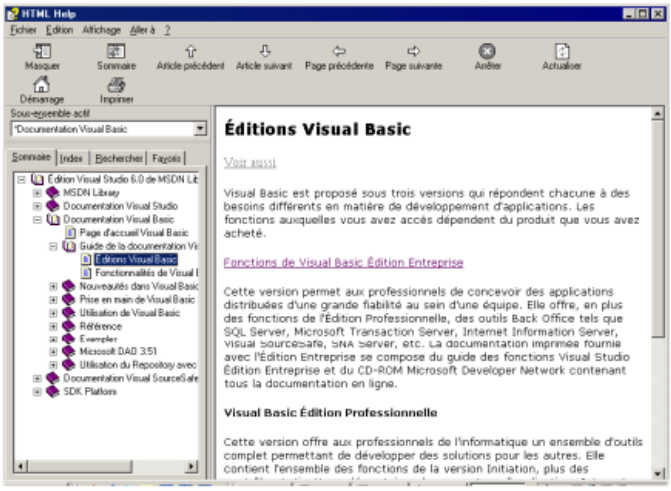
Le choix **Fractionner** permet de diviser la fenêtre de code en deux parties horizontales à partir de la ligne où est placé le curseur.

Le menu Aide (?)

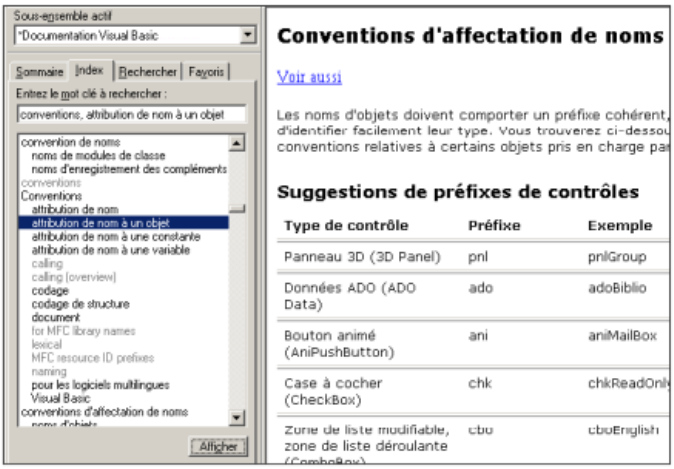
L'aide en ligne de Visual Basic fait partie de la bibliothèque MSDN Visual Studio (MSDN, Microsoft Developer Network) qui contient des informations sur l'ensemble des outils de développement de Microsoft. Pour *limiter les recherches*, sélectionnez "Documentation Visual Basic" dans la zone "Sous-ensemble actif". L'aide en ligne de Visual Basic utilise le format HTML, un format facilitant grandement les recherches. Vous trouverez dans l'aide une réponse à la plupart de vos questions et plusieurs exemples de programmes.

Les choix **Sommaire**, **Index de référence** et **Rechercher** du menu, démarrent l'aide et affichent l'onglet correspondant dans la fenêtre de gauche.

- L'onglet **Sommaire** présente une table des matières des manuels d'aide avec une structure arborescente. Pour accéder à des pages d'aide, double-cliquez d'abord sur un manuel, ensuite sur un chapitre et finalement cliquez sur une page d'information pour l'afficher dans la fenêtre de droite.



- L'onglet **Index** recherche un terme que l'on doit entrer, dans un index alphabétique. Pour consulter une rubrique, cliquez sur le bouton Afficher. Notez qu'une rubrique en gris n'est pas accessible.



- L'onglet **Rechercher** recherche un terme que l'on doit entrer, dans une base de données qui est générée lors de la première utilisation de l'aide. Cette recherche est beaucoup plus élaborée que la recherche dans un index. Elle affiche dans une liste des éléments trouvés, avec l'emplacement de chacun. Sélectionnez un élément trouvé et cliquez sur le bouton Afficher pour consulter l'information.
- L'onglet **Favoris** permet de conserver des noms de rubriques consultées, facilitant ainsi des recherches futures. Utilisez le bouton Ajouter pour conserver un signet.
- Il est possible d'imprimer une rubrique d'aide en sélectionnant au menu, Fichier /Imprimer.

Le choix **Support technique** fournit de l'information sur la façon d'obtenir un support technique de Microsoft.

Le choix **Microsoft sur le Web** fournit des liens Internet sur des pages Web de Microsoft.

Le choix **À propos de Microsoft Visual Basic** fournit des renseignements sur la licence d'utilisation de votre logiciel et un bouton "Infos système" donne des informations sur votre ordinateur (matériel et logiciel).

L'**aide contextuelle**, accessible par la **touche F1**, présente à partir de MSDN, l'information relative à un *élément pointé ou sélectionné* dans la fenêtre de code. Il peut s'agir d'un *mot clé* du langage Visual Basic, d'une *propriété*, d'un *message d'erreur* ou tout élément de l'environnement de Visual Basic.

Exemples

Le curseur est placé sur le mot clé Private. En pressant la touche F1, une page d'information concernant ce mot est affichée dans une fenêtre MSDN.

En cours d'exécution, le message d'erreur 5 apparaît. En pressant la touche F1, le résultat est le même qu'en sélectionnant le bouton Aide.

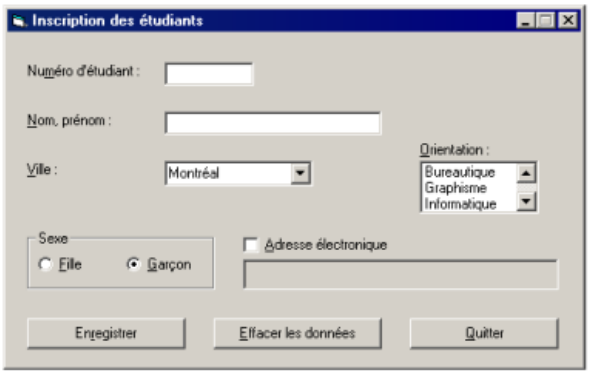
3 Découvrir et maîtriser Visual Basic par la pratique

3.1 EXERCICE 3 CRÉER LE PROJET INSCRIPTION

Étape 1 Préparation et enregistrement

1. Dans Visual Basic, ouvrez un nouveau projet de type Exe Standard.
2. Nommez le projet **Inscription** et la forme **frmInscription**. Enregistrez la forme dans le fichier **Inscription.frm** d'un nouveau dossier .../Visual Basic /Exercice 3 Inscription. Enregistrez le projet dans le même dossier sous le nom **Inscription.vbp**.

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de la feuille frmInscription :
   - La barre de titre contient le titre "Inscription des étudiants" ;
   - Le bouton Agrandir n'est pas accessible (**MaxButton = False**) ;
   - La feuille doit être affichée au centre de l'écran.

2. Ajoutez les étiquettes pour les titres Numéro d'étudiant, Nom, prénom, Ville et Orientation en spécifiant les *raccourcis clavier*.

3. Ajoutez les trois boutons de commande cmdEnregistrer, cmdEffacer et cmdQuitter. Le bouton Enregistrer est le *bouton par défaut* et le bouton Quitter est le *bouton d'annulation*.

4. Utilisez trois **zones de texte** (contrôles **TextBox**) pour la saisie du numéro, du nom et de l'adresse électronique.

Consultez la section Le contrôle TextBox, page 168.

- Nommez les trois zones de texte : txtNumero, txtNom et txtCourriel.
- Effacez le contenu de la propriété **Text** des trois zones de texte afin qu'aucun texte ne soit affiché au démarrage.
- Modifiez la couleur d'arrière-plan de la zone txtCourriel en sélectionnant Onglet système /Barres de défilement.
- Fixez la propriété **Enabled = False** de la zone txtCourriel afin que cette zone ne soit pas accessible au démarrage.

> La propriété **Enabled** permet de rendre accessible ou non un contrôle à l'utilisateur. La valeur True indique que l'utilisateur a accès à la zone de texte et la valeur False lui en interdit l'accès.

5. Ajoutez un contrôle **ComboBox** pour la saisie de la ville.

Consultez la section Les contrôles ListBox et ComboBox, page 172.

- Nommez le contrôle cboVille.
- Fixez la propriété **Text = Montréal**.
- Sélectionnez la propriété **List** et tapez **Montréal**. Pressez les touches **Ctrl+Enter** pour *changer de ligne*. Entrez les autres villes (une par ligne) : Québec, Sherbrooke, Chicoutimi, Gaspé, Granby et Trois-Rivières.

Le contrôle **ComboBox** ou **Zone de liste modifiable** permet d'afficher une liste de valeurs fixées par la propriété **List**. La propriété **Text** permet de fixer une valeur par défaut, ainsi Montréal sera la ville sélectionnée lors du démarrage.

L'utilisateur a la possibilité de choisir une ville dans la liste ou il *peut taper un autre nom de ville*.

- Exécutez le projet et déroulez la liste des villes. Celles-ci apparaissent dans la même séquence qu'elles ont été entrées. Fixez la propriété **Sorted = True** et exécutez de nouveau. Les villes devraient maintenant apparaître en *ordre alphabétique*.

6. Ajoutez un contrôle **ListBox** pour la saisie de l'orientation.

Consultez la section Les contrôles ListBox et ComboBox, page 172.

- Nommez le contrôle lstOrientation.
- Sélectionnez la propriété **List** et entrez les orientations suivantes (une par ligne) : Informatique, Réseautique, Multimédia, Bureautique et Graphisme.

Le contrôle **ListBox** ou **Zone de liste** permet d'afficher une liste de valeurs à l'utilisateur. Celui-ci ne peut pas modifier les valeurs. En tapant la première lettre d'une valeur, il peut accéder rapidement à un élément précis d'une longue liste. Généralement, les valeurs doivent être triées par la propriété Sorted.

- Affichez les orientations en ordre alphabétique par la propriété Sorted.
- Exécutez le projet, cliquez dans la zone de liste des orientations et pressez la lettre **r** pour accéder directement au choix Réseautique.

7. Tracez un **cadre** (contrôle **Frame**) pour la saisie du sexe de l'étudiant. Fixez la propriété **Caption = Sexe**.

Consultez la section Le contrôle Frame, page 176.

8. Utilisez deux **boutons d'option** (contrôles **OptionButton**) et nommez-les optFille et optGarcon. Fixez la propriété Caption de chacun des boutons : Fille et Garçon ainsi que les raccourcis clavier.

Consultez la section Le contrôle OptionButton, page 170.

9. Modifiez la propriété **Value = True** du bouton d'option optGarcon.

> La propriété **Value = True** permet de fixer une option par défaut. Un seul bouton peut être sélectionné dans un conteneur qui regroupe plusieurs boutons.

10. Utilisez une **case à cocher** (contrôle **CheckBox**) pour une saisie possible d'une adresse électronique. Nommez le contrôle chkCourriel, fixez la propriété Caption : "Adresse électronique" et le raccourci clavier.

Consultez la section Le contrôle CheckBox, page 171.

Étape 3 Ajuster l'ordre des tabulations par la propriété TabIndex

1. Exécutez le projet. Pressez la touche **Tab** plusieurs fois et observez le déplacement du pointeur. Celui-ci se déplace dans l'ordre de création des contrôles. L'ordre désiré serait plutôt : numéro, nom, ville, orientation, sexe, case à cocher, zone de texte pour la saisie de l'adresse électronique, bouton Enregistrer, bouton Effacer et bouton Quitter.

> La propriété **TabIndex** d'un contrôle détermine **l'ordre de tabulation**. C'est l'ordre dans lequel les déplacements sont effectués en pressant la touche Tab. Le premier contrôle qui reçoit le focus porte la valeur TabIndex = 0, le deuxième porte la valeur 1, le troisième la valeur 2, etc.
> Une étiquette qui identifie une zone de texte ou une zone de liste doit précéder celle-ci et porter la valeur : (**TabIndex de la zone – 1**) afin que le raccourci clavier soit fonctionnel.

Consultez la section Les propriétés TabIndex et TabStop, page 195.

2. Fixez les valeurs suivantes à la propriété TabIndex des contrôles afin d'obtenir l'ordre de tabulation souhaité.

titre Numéro	0	txtNumero	1
titre Nom	2	txtNom	3
titre Ville	4	txtVille	5
titre Orientation	6	txtOrientation	7
cadre Sexe	8	optFille	9
optGarcon	10	chkCourriel	11
txtCourriel	12	cmdEnregistrer	13
cmdEffacer	14	cmdQuitter	15

3. Exécutez le projet et testez *l'ordre de tabulation* avec la touche **Tab** ainsi que tous *les raccourcis clavier* avec la touche **Alt**.

Étape 4 Coder les procédures

1. Codez et testez les procédures en traduisant le pseudo-code en Visual Basic. Lisez attentivement les notions théoriques qui suivent avant de commencer.

> Pour *effacer une zone de texte*, affectez-lui une chaîne vide (2 guillemets consécutifs). ***Exemple :*** txtZoneTexte.Text = ""
>
> Pour donner le *focus* à un contrôle, utilisez la méthode **SetFocus**.
>
> ***Exemple :*** txtZoneTexte.SetFocus.
>
> Pour vérifier si une case est cochée ou non, comparez sa propriété *Value* avec les constantes **VbChecked** (cochée) ou **VbUnchecked** (non cochée).

Procédure Click du bouton cmdQuitter

```
Décharger la feuille de la mémoire
```

Procédure Click du bouton cmdEffacer

```
Effacer le contenu des trois zones de texte
Fixer la valeur par défaut "Montréal" pour la ville
Fixer la valeur par défaut "Informatique" pour l'orientation
(propriété Text)
Activer le bouton d'option Garçon
Désactiver la case à cocher du courriel
Rendre inaccessible la zone de texte du courriel
Donner le focus à la zone de texte du numéro d'étudiant
```

Procédure Click du bouton cmdEnregistrer

```
Si le numéro ou le nom est absent alors,
   afficher le message qui suit et donner le focus au numéro
   d'étudiant
```

```
Sinon
afficher le message qui suit et faire appel à la procédure
cmdEffacer_Click pour effacer les données après l'enregistrement
(instruction Call)
```

Consultez la section L'instruction If, les opérateurs logiques (page 262) et la section La fonction MsgBox, page 256.

Si un titre n'est pas spécifié pour la barre de titre, la fonction MsgBox utilisera le nom du projet Inscription.

Le numéro de l'étudiant et le nom sont des données obligatoires pour procéder à l'enregistrement.

Les données saisies ne sont pas vraiment enregistrées. Ce projet sera complété dans l'exercice 23.

Procédure Click de la case à cocher chkCourriel

```
Si la case est cochée, alors
   rendre la zone de texte du courriel accessible,
   afficher  l'arrière-plan en blanc (constante vbWhite),
   donner le focus à la zone de texte du courriel
Sinon
rendre inaccessible  la zone de texte du courriel,
donner une apparence d'inaccessibilité à la zone de
texte du courriel en utilisant la constante couleur
système vbScrollBars (couleur barre de défilement)
Fin Si
```

Les **constantes couleurs** sont répertoriées dans la section A.3 de l'Annexe A.

Les **constantes couleurs système** sont répertoriées dans la section A.4 de l'Annexe A.

2. Testez à fond le projet. Entrez des données et testez toutes les conditions.

Exemple Cliquez sur le bouton Enregistrer sans avoir tapé de numéro et de nom.

Activez et désactivez la case à cocher.

Effacez les données après un enregistrement.

Etc.

Étape 5 Découvrir les événements Form_Load et Form_Unload

L'événement **Form_Load** se produit au moment où la feuille est chargée en mémoire avec tous les objets qu'elle contient, avant que celle-ci soit affichée à l'écran.

L'événement **Form_Unload** se produit au moment où la feuille est déchargée de la mémoire, à la rencontre de l'instruction Unload Me ou lorsque l'utilisateur presse le bouton Fermer de la barre de titre.

1. Codez et testez la procédure suivante :

```
Private Sub Form_Load()
  lstOrientation.Text = "Informatique"
End Sub
```

Lors du démarrage du projet, aucune orientation n'est spécifiée par défaut dans la zone de liste, comme dans le cas des villes (Montréal par défaut). L'événement Form_Load est très utile pour coder des instructions d'initialisation.

Rappel :

Utilisez les zones de listes de la fenêtre de code pour ajouter une nouvelle procédure :

2. Lors de l'événement Form_Unload, affichez le message suivant :

3. Exécutez le projet et pressez le bouton Quitter. Exécutez de nouveau et pressez le bouton Fermer de la barre de titre. Le message devrait s'afficher dans les deux cas.

3.2 EXERCICE 4 CRÉER LE PROJET CADRAN

Étape 1 Préparation, observation et enregistrement

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre3 /Exercice4 /Debut dans un nouveau dossier .../Visual Basic /Exercice 4 Cadran.
2. Exécutez le fichier .../Visual Basic /Exercice 4 Cadran /**Cadran.exe**.

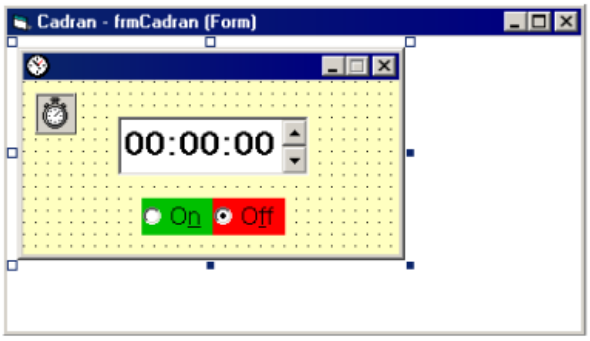
3. Placez le pointeur de la souris sur les chiffres du cadran. Une info-bulle informe l'utilisateur sur la façon de fixer l'heure de sonnerie. Fixez une heure qui déclenchera la sonnerie dans environ une minute (heure actuelle + 1 minute) et sélectionnez le bouton d'option ON.

4. Attendez la sonnerie qui sera confirmée par un message. En réponse au message, pressez OK et observez le cadran, celui-ci est maintenant en mode OFF.

5. Fixez de nouveau une heure de sonnerie et pressez le bouton ON. Fermez le cadran avant qu'il sonne. Observez le message affiché dans ce cas.

6. Dans Visual Basic, ouvrez un nouveau projet. Nommez le projet **Cadran** et la forme **frmCadran**. Enregistrez la forme dans le fichier **Cadran.frm** du dossier .../Visual Basic /Exercice 4 Cadran et le projet dans le fichier **Cadran.vbp** du même dossier.

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de **frmCadran** :
   - La barre de titre ne contient pas de titre ;
   - Le bouton Agrandir n'est pas disponible ;
   - La fenêtre doit être affichée dans le centre de l'écran ;
   - Fixez une couleur d'arrière-plan pour le cadran ;
   - L'icône de la barre de titre est le fichier Clock03.ico du dossier Program Files \Microsoft Visual Studio \Common \Graphics \Icons \Misc.

Visual Basic est fourni avec un ensemble de fichiers d'icônes. Ces fichiers sont généralement placés dans le dossier **Program Files\Microsoft Visual Studio\Common\Graphics\Icons**. Au cours des prochains exercices, nous accéderons souvent à ce dossier, c'est donc *un chemin d'accès à retenir.*

2. Ajoutez les contrôles et fixez les propriétés :

- 2 boutons d'option, **optOn** (raccourci clavier, arrière-plan vert) et **optOff** (raccourci clavier, arrière-plan rouge) ; activez le bouton Off par défaut ;
- 1 contrôle **Timer** portant le nom **tmrCadran**, propriété **Interval = 1000** ;

Le contrôle **Timer** permet de déclencher l'événement ***Timer*** à un intervalle de temps fixé par la propriété **Interval** exprimée en millième de secondes. Dans le cas présent, la valeur 1000 déclenchera l'événement Timer à chaque seconde. Le contrôle Timer n'est pas visible à l'exécution.

Consultez la section Le contrôle Timer, page 177.

- 1 contrôle **DateTimePicker** qui doit être ajouté dans la boîte d'outils comme suit :

- Affichez le menu contextuel de la boîte à outils avec le bouton droit de la souris et sélectionnez Composants ;
- Dans l'onglet Contrôles, cochez le choix "Microsoft Windows Common Controls-2 6.0" ;
- Ajoutez le contrôle **DTPicker** sur la feuille ;
- (Name) = dtpCadran, Format = 2 – dtpTime, taille 18, style gras ;
- Spécifiez le texte de *l'info-bulle* par la propriété **ToolTipText** = "Cliquez sur les heures, les minutes ou les secondes et par la suite utilisez les pointes de flèches pour fixer l'heure de sonnerie".

Le contrôle **DateTimePicker** permet à l'utilisateur de sélectionner des heures et des dates. Les dates peuvent être choisies dans un calendrier déroulant. Pour plus de détails sur ce contrôle, consultez l'Aide MSDN.

Étape 3 Coder les procédures événementielles

1. Entrez les lignes de code qui suivent :

```
Dim mblnOn As Boolean     'cadran en mode ON ou non
Dim mdtmSonnerie As Date  'heure de sonnerie
```

Pour les besoins de notre application, 2 variables de portée *module* doivent être déclarées par le mot clé **Dim** afin de conserver des valeurs qui seront vérifiées par plusieurs procédures.

La variable mblnOn de type *Boolean* permettra de déterminer si le cadran est en mode ON ou non.

La variable mdtmSonnerie de type *Date* sera utilisée pour conserver l'heure de sonnerie.

Consultez la section 5.7 Les variables et les types de données.

2. Ajoutez la procédure suivante :

```
Private Sub Form_Load()
  dtpCadran.Value = Time
End Sub
```

Lors du chargement de la feuille, l'heure actuelle (fonction **Time**) est affectée à la propriété Value du contrôle DateTimePicker.

Consultez la section Les fonctions de date et heure, page 292.

3. Ajoutez la procédure suivante :

```
Private Sub optOn_Click()
  mblnOn = True
  mdtmSonnerie = FormatDateTime(dtpCadran.Value, vbLongTime)
  'heure de sonnerie dans la barre de titre
  Me.Caption = mdtmSonnerie
End Sub
```

La valeur True est affectée à la variable mblnOn puisque l'utilisateur a cliqué sur le bouton ON.

La valeur de dtpCadran est exprimée dans un format long (date et heure). La fonction **FormatDateTime** permet d'extraire *seulement l'heure* et de la placer dans la variable mdtmSonnerie.

L'heure de sonnerie est placée dans la barre de titre de la feuille.

Consultez la section Les fonctions de date et heure, page 292.

4. Lorsque l'utilisateur clique sur le bouton OFF, affectez la valeur False à la variable mblnOn. Ajoutez le code de programmation.

5. Ajoutez la procédure suivante :

```
'À chaque seconde, vérification de l'heure de sonnerie
'avec celle de l'ordinateur et message d'avertissement
Private Sub tmrCadran_Timer()
  If mblnOn = True And Time = mdtmSonnerie Then
    Beep
    MsgBox "Il est : " & Time, vbInformation, "Allez Hop!!!"
    mblnOn = False
    Unload Me
  End If
End Sub
```

Cette procédure est exécutée à chaque seconde (intervalle = 1000). Si le bouton ON est enclenché et que l'heure de sonnerie est égale à l'heure actuelle, la sonnerie est activée (instruction **Beep**) et un message est affiché avec l'icône d'information. La feuille est ensuite déchargée.

La valeur de la variable mblnOn est fixée à False et son contenu sera vérifié dans l'événement **QueryUnload** qui sera codé au numéro 7.

Consultez la section 5.16 L'instruction Beep.

6. Exécutez le projet et assurez-vous de son bon fonctionnement avant de passer à l'étape suivante.

7. Ajoutez la procédure suivante : (Rappel : *Ne pas taper la première ligne. Utilisez les zones de listes de la fenêtre de code !*)

```
Private Sub Form_QueryUnload(Cancel As Integer, _
                        UnloadMode As Integer)
  Dim intRep As Integer
  Dim strMessage As String

  If mblnOn = True Then
    strMessage = "Le cadran n'a pas encore sonné, " & _
                 "désirez-vous réellement quitter?"
    intRep = MsgBox(strMessage, vbYesNo + vbQuestion, "Attention!")
    If intRep = vbYes Then
      End             'Fin de l'exécution
    Else
      Cancel = True   'Annulation du déchargement de la feuille
    End If
```

```
  Else
    End
  End If
End Sub
```

L'événement **QueryUnload** est déclenché juste avant que la feuille se décharge de la mémoire. Dans le cas présent, l'événement se produit lorsque l'utilisateur presse le bouton Fermer ou qu'il utilise les touches Alt + F4. Le mot **Query** signifie *interrogation* et **Unload** signifie *déchargement* de la mémoire. L'événement QueryUnload nous permet d'interroger l'utilisateur avant le déchargement de la feuille.

Consultez la section Les événements d'une feuille, page 160.

Deux variables de portée *locale* sont déclarées, celles-ci étant utilisées uniquement dans cette procédure. La variable intRep de type *Integer* est utilisée pour la saisie de la réponse de l'utilisateur. La variable strMessage de type *String* contient une chaîne de caractères qui permet de réduire la longueur du code de la fonction MsgBox. La chaîne de caractères est formée de deux chaînes jointes par le **caractère de concaténation &**.

Consultez la section 5.11 La concaténation des chaînes de caractères.

Une *structure imbriquée* de l'instruction If a été utilisée. Celle-ci est formée de deux instructions If/Else/End If.

Consultez la section L'instruction If, Structures imbriquées, page 264.

Dans la fonction MsgBox, la constante **vbYesNo** provoque l'affichage d'un bouton Oui et d'un bouton Non. Le caractère **+** permet de citer plusieurs constantes.

Consultez la section La fonction MsgBox, page 256.

Dans le cas où l'utilisateur répond Oui ou que le cadran est en mode OFF, l'exécution est interrompue par l'instruction **End**.

Consultez la section 5.5 Les instructions d'arrêt (End, Unload et Stop).

La procédure Form_QueryUnload offre la possibilité d'interrompre le déchargement de la feuille à l'aide de l'*argument* **Cancel**. En lui affectant la valeur True, la feuille n'est pas déchargée et l'exécution du projet n'est pas interrompue. Le concept *d'arguments* est traité de façon plus détaillée dans l'exercice 14.

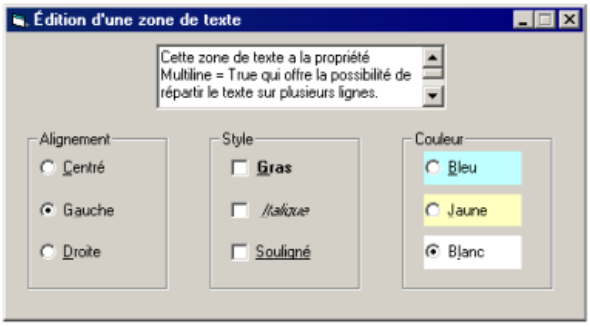
8. Exécutez le projet et testez les 2 cas suivants :
   - L'utilisateur quitte par le bouton Fermer et le cadran est en mode OFF.
   - L'utilisateur utilise les touches Alt+F4 pour fermer et le cadran est en mode ON. Testez le bouton Oui et le bouton Non du message.
9. Créez le fichier exécutable du projet (Cadran.exe). Exécutez ce fichier à partir de l'Explorateur de Windows afin de vous assurer de son fonctionnement.

3.3 EXERCICE 5 CRÉER LE PROJET ÉDITION DE TEXTE

Étape 1 Préparation, observation et enregistrement

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre3 /Exercice5 /Debut dans un nouveau dossier .../Visual Basic /Exercice 5 Edition.
2. Exécutez le fichier .../Visual Basic /Exercice 5 Edition /**Edition.exe**. Sélectionnez les différents choix et observez le résultat.
3. Placez le pointeur de la souris sur chacun des trois cadres et observez l'affichage des info-bulles.
4. Déplacez le pointeur de souris dans la zone de texte et cliquez sur le bouton droit de la souris. Un menu contextuel est affiché. Aucune programmation n'est nécessaire pour ce menu. Il fait partie des propriétés d'une zone de texte.
5. Dans Visual Basic, ouvrez un nouveau projet. Nommez le projet **Edition** et la forme **frmEdition**. Enregistrez la forme dans le fichier **Edition.frm** du dossier .../Visual Basic /Exercice 5 Edition et le projet dans le fichier **Edition.vbp** du même dossier.

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de frmEdition :
   - Entrez le titre de la barre de titre ;
   - Le bouton Agrandir de la barre de titre n'est pas accessible ;
   - L'utilisateur ne peut modifier la taille de la feuille.
2. Ajoutez les contrôles et fixez leurs propriétés :
   - Ajoutez un contrôle TextBox nommé **txtZoneTexte** :
     - Fixez la propriété **Multiline = True** pour permettre l'entrée sur plusieurs lignes.
     - Tapez le texte de l'encadré dans la propriété Text. Utilisez les touches Ctrl+Enter pour forcer un changement de ligne.

     > La propriété **Multiline = True** offre la possibilité de répartir le texte sur plusieurs lignes.
     >
     > La propriété **ScrollBars** permet l'ajout d'une barre de défilement verticale.

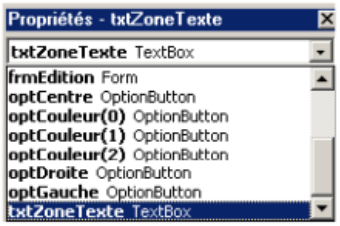
     - Fixez la propriété **ScrollBars = 2-Vertical** pour ajouter une barre de défilement verticale.
   - Ajoutez un contrôle Frame pour *l'alignement* et trois boutons d'option :
     - Une info-bulle affiche "Choisir l'alignement" ;
     - Noms des boutons d'option : optCentre, optGauche et optDroite ;
     - Le bouton optGauche est activé au démarrage.
   - Ajoutez un contrôle Frame pour *le style* et trois cases à cocher :
     - Une info-bulle affiche "Choisir un style" ;
     - Noms des cases à cocher : chkGras, chkItalique et chkSouligne.
   - Ajoutez un contrôle Frame pour *la couleur* et trois boutons d'option qui doivent être insérés de façon à créer un **groupe de contrôles** :
     - Une info-bulle affiche "Choisir une couleur d'arrière-plan" ;
     - Création du **groupe de contrôles** :
       1. Dessinez un premier bouton d'option dans le cadre des couleurs.
       2. Nommez le bouton optCouleur.

3. Copiez le bouton sélectionné par Ctrl+C.
4. Sélectionnez le cadre des couleurs.
5. Pressez Ctrl+V pour Coller.
6. Répondez OUI à la question :

   "Il y a déjà un contrôle nommé 'OptCouleur'. Voulez-vous créer un groupe de contrôles?"

7. Créez un troisième bouton par Ctrl+C et Ctrl+V.

- Observez le contenu de la **liste de sélection d'un objet** de la fenêtre des propriétés :



On devrait y retrouver les objets optCouleur(0), optCouleur(1) et Couleur(2). Si ce n'est pas le cas, recréez le groupe de contrôles avant de continuer.

- Observez la propriété **Index** de chacun des boutons d'option. Ces valeurs ont été automatiquement fixées à 0, 1 et 2.
- Fixez la couleur d'arrière-plan des boutons d'option :

Bleu	Onglet Palette, première ligne, troisième case à partir de la droite. Observez le code de couleur hexadécimal qui est affiché, soit &H00FFFFC0&. Ce code sera utilisé dans le programme pour appliquer cette couleur à la zone de texte.
Jaune	Onglet Palette, première ligne, quatrième case à partir de la gauche. Le code de couleur est &H00C0FFFF&.
Blanc	Onglet Palette, première case dans le coin supérieur gauche. La constante vbWhite est l'équivalent du code de cette couleur.

- Le bouton optCouleur(2) qui représente le blanc, doit être le bouton activé au démarrage.

Étape 3 Coder les procédures événementielles

1. Codez les événements Click des boutons d'option optCentre, optGauche et optDroite. Affectez les valeurs vbCenter, vbRightJustify et vbLeftJustify à la propriété **Alignment** de la zone de texte selon le cas. Testez le projet.

2. Codez les événements Click des trois cases à cocher chkGras, chkItalique et chkSouligne. Attribuez les valeurs True ou False aux propriétés **FontBold**, **FontItalic** et **FontUnderline** du contrôle txtZoneTexte selon le cas. Utilisez une instruction If dans chacune des procédures. Testez le projet.

3. Codez une procédure **commune** aux 3 boutons d'options qui forment un groupe de contrôles : (Rappel : *Ne pas taper la première ligne. Utilisez les deux zones de liste de la fenêtre de code.*)

```
Private Sub optCouleur_Click(Index As Integer)
  If Index = 0 Then
    txtZoneTexte.BackColor = &H00FFFFC0
  ElseIf Index = 1 Then
    txtZoneTexte.BackColor = &H00C0FFFF
  Else
    txtZoneTexte.BackColor = vbWhite
  End If
End Sub
```

> L'argument **Index** a été ajouté automatiquement entre parenthèses au début de la procédure. Lorsqu'un bouton d'option est activé, la variable Index est mise à jour par Visual Basic. Elle prend alors la valeur de la propriété Index du bouton d'option activé. La variable Index ne doit pas être déclarée.
>
> Les **groupes de contrôles** sont un élément très utile en Visual Basic. Le fait de regrouper des contrôles comme des zones de texte, des boutons de commande ou encore des étiquettes permet de réduire de façon appréciable le code de programmation et par le fait même, d'améliorer la logique d'un programme.
>
> Lors de l'entrée d'un code de couleur dans l'éditeur de Visual Basic, celui-ci est automatiquement réduit lors d'un changement de ligne.

4. Testez le projet et modifiez la couleur de l'arrière-plan. Observez les info-bulles des cadres ainsi que le menu contextuel de la zone de texte avec le bouton droit de la souris.

Étape 4 Affectation d'une valeur contraire avec le mot clé Not (Bascule)

1. Modifiez la procédure chkSouligne_Click :

```
Private Sub chkSouligne_Click()
  txtZoneTexte.FontUnderline = Not txtZoneTexte.FontUnderline
End Sub
```

En affectant la valeur inverse (Not), si la case est cochée, elle sera décochée et si elle n'est pas cochée, elle sera cochée. Cette technique de programmation est appelée **programmation d'une bascule**. Cette technique est souvent utilisée lors de la programmation de cases à cocher.

2. Utilisez la *technique de programmation d'une bascule* pour les cases à cocher chkGras et chkItalique. Testez les changements.

3.4 EXERCICE 6 MODIFIER LE PROJET GROUPE DE CONTRÔLES

Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre3 /Exercice6 /Debut dans un nouveau dossier .../Visual Basic /Exercice 6 GroupeControle.

2. Dans Visual Basic, ouvrez le projet .../Visual Basic /Exercice 6 /Debut /GroupeControle.vbp. Exécutez le projet et observez son fonctionnement.

3. Observez le nouveau contrôle **Shape** de la boîte à outils :

Le contrôle **Shape** permet de tracer des formes géométriques comme un rectangle, un carré, une ellipse ou un cercle. Sa propriété **Shape** permet de sélectionner le style de la forme. Dans le cas présent, le style 4-Rounded Rectangle est activé.

Consultez la section Le contrôle *Line* et le contrôle *Shape*, page 180.

4. Attribuez différentes valeurs à la propriété *Shape* du contrôle shpForme afin d'observer les différentes formes disponibles.

5. Dans la fenêtre code, observez attentivement le code de programmation. Une procédure Click est codée pour chacune des images. Le but de cet exercice est de former un **groupe de contrôles** avec les quatre images afin de remplacer les quatre procédures par une *seule* procédure commune.

La propriété **Left** d'un contrôle spécifie la distance entre la position du contrôle sur la feuille et l'extrémité gauche de la feuille. Si la distance du contrôle Shape et d'un contrôle Image est identique, les deux contrôles seront nécessairement *superposés*. Comme la propriété Top, la modification de la propriété Left provoque un déplacement de l'objet.

Étape 2 Créer un groupe de contrôles avec des contrôles existants

1. Effectuez les opérations suivantes afin de **mettre en commentaire** un bloc d'instructions :
   - Affichez la barre d'outils **Édition** en cliquant avec le bouton droit de la souris à un endroit non utilisé de la barre d'outils **Standard** ;
   - Sélectionnez le texte des 4 procédures Click des images ;
   - Cliquez sur le bouton **Commenter le bloc** de la barre Édition.

La méthode qui précède est très utile pour commenter rapidement des blocs de code ainsi que le bouton **Ne pas commenter le bloc** .

Les boutons **Mettre en retrait** et **Retrait négatif** de la barre d'outils Édition sont aussi utiles pour la mise en retrait d'un bloc d'instructions sélectionnées. Notez que les touches Tab et Maj+Tab sont équivalentes.

2. À partir de la fenêtre des propriétés, modifiez la propriété (*Name*) des 4 images, dans l'ordre suivant : trèfle, carreau, cœur et pique, pour un seul nom **imgCouleur**. Répondez **Oui** à la question concernant un *groupe de contrôles*. Observez la valeur de la propriété *Index* de chacune des images (0, 1, 2, 3).

3. Codez la procédure commune pour les 4 images :

```
Private Sub imgCouleur_Click(Index As Integer)
  shpForme.Left = imgCouleur(Index).Left

  Codez une instruction If/ElseIf/EndIf pour vérifier la valeur de la
  variable Index afin de mettre à jour l'étiquette lblInfo.
End Sub
```

4. Testez le projet et assurez-vous de son fonctionnement.

***Étape 3 La structure de décision* Select Case**

1. Remplacez l'instruction If/ElseIf/EndIf de la procédure imgCouleur_Click par la structure de décision **Select Case**. Placez l'instruction If en commentaire.

```
Select Case Index
  Case 0
    lblInfo.Caption = "Couleur sélectionnée : Trèfle"
  Case 1
    lblInfo.Caption = "Couleur sélectionnée : Carreau"
  Case 2
    lblInfo.Caption = "Couleur sélectionnée : Coeur"
  Case 3
    lblInfo.Caption = "Couleur sélectionnée : Pique"
End Select
```

Le contenu de la variable **Index** est vérifié. Dans le *cas* où la variable prend l'une des valeurs citées (0 à 3), l'instruction citée à la suite de la valeur est exécutée.

Consultez la section L'instruction *Select Case*, page 265, pour observer d'autres structures de l'instruction Select Case.

Dans cet exercice, les instructions If et Select Case sont *répétitives*. Dans la partie 2 de l'exercice 10, une nouvelle technique de programmation nous permettra de réduire ces instructions à *une seule ligne de code*, en utilisant un *tableau en mémoire*.

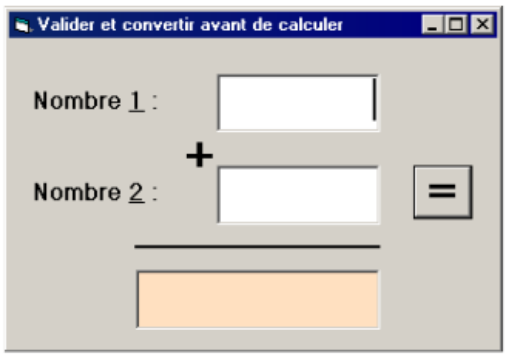
2. Testez le changement et assurez-vous du bon fonctionnement.
3. Enlevez le code de programmation inutile dans votre programme.

3.5 EXERCICE 7 VALIDER ET CONVERTIR AVANT DE CALCULER

Étape 1 Préparation et enregistrement

1. Dans Visual Basic, ouvrez un nouveau projet de type Exe Standard.
2. Nommez le projet **ValiderConvertir** et la feuille **frmCalcul**. Enregistrez la feuille dans le fichier **Calcul.frm** d'un nouveau dossier .../Visual Basic /Exercice 7 ValiderConvertir. Enregistrez le projet dans le même dossier sous le nom **ValiderConvertir.vbp**.

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de la feuille frmCalcul :
   - Le titre de la barre de titre ;
   - La feuille doit être affichée dans le centre de l'écran.
2. Ajoutez les contrôles et fixez les propriétés :
   - Utilisez deux étiquettes pour les titres Nombre 1 et Nombre 2. Spécifiez les raccourcis clavier, la taille 12 et le style gras.
   - Ajoutez une troisième étiquette pour le signe + (taille 24, style gras).
   - Ajoutez deux zones de texte pour la saisie des nombres, nommez-les txtNb1 et txtNb2 et fixez la taille des caractères à 14. Les caractères saisis doivent être *alignés à droite*.
   - Tracez une ligne à l'aide du contrôle **Line** de la boîte d'outils. Modifiez la propriété : BorderWidth = 2.

Le contrôle **Line** permet de tracer une ligne horizontale, verticale ou en diagonale. Sa propriété **BorderWidth** définit la largeur de la ligne.

Consultez la section Le contrôle *Line* et le contrôle *Shape*, page 180.

   - Ajoutez un bouton de commande pour le signe =. Nommez-le cmdCalculer, fixez la taille à 24 et le style gras.

- Utilisez une étiquette pour le résultat du calcul et nommez-la lblTotal. Donnez-lui l'apparence d'une zone de texte et sélectionnez une couleur pour l'arrière-plan. Le résultat du calcul doit être *aligné à droite*.

3. Exécutez le projet et vérifiez l'ordre de tabulation qui devrait être : Nombre 1, Nombre 2 et bouton =. Vérifiez aussi les raccourcis clavier Alt+1 et Alt+2. Apportez des changements si nécessaire.

Étape 3 Coder les procédures

1. Codez la procédure suivante :

```
Private Sub cmdCalculer_Click()
  lblTotal = txtNb1 + txtNb2
  txtNb1.SetFocus
End Sub
```

> L'instruction de calcul aurait pu prendre la forme suivante :
>
> ```
> lblTotal.Caption = txtNb1.Text + txtNb2.Text
> ```
>
> Les propriétés **Caption** et **Text** peuvent être omises, puisque celles-ci sont respectivement les **propriétés par défaut** des contrôles **Label** et **TextBox**. Une *seule* propriété d'un objet peut être définie comme *propriété par défaut* lors de la conception de l'objet.

2. Exécutez et entrez les nombres 5 et 7. Observez attentivement le résultat.

> Une zone de texte contient des données de type *chaîne de caractères*. Dans cette instruction, le signe + a le même effet que le caractère de *concaténation* &. Afin que Visual Basic considère le contenu des zones de texte comme une zone de type numérique, celles-ci doivent être converties au format numérique, à l'aide des **fonctions de conversion de types de données**.
>
>
> Consultez la section Les fonctions de conversion de types de données, page 286.

3. Modifiez l'instruction d'addition comme suit :

```
lblTotal = Val(txtNb1) + CInt(txtNb2)
```

> La fonction **Val** convertit en *valeur numérique* et la fonction **Cint** convertit au type de données *Integer* (entier).

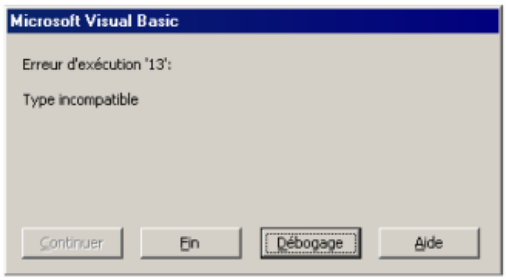
4. Exécutez et effectuez les tests suivants :

- Entrez les nombres 5 et 7 et calculez.

- Effacez la zone de texte du premier nombre et calculez.
- Entrez des lettres dans la zone du premier nombre et calculez.

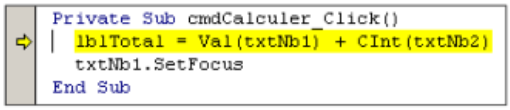
> La fonction **Val** affecte la *valeur numérique* 0 en présence d'une chaîne de caractères *vide* ("") ou d'une chaîne de caractères *non numérique*.

5. Exécutez et effectuez les tests suivants :
   - Entrez le nombre 7 dans la zone du premier nombre et aucune donnée dans la zone du deuxième nombre. Calculez et observez le résultat. Le message d'erreur qui suit devrait s'afficher :



> L'erreur 13 est une erreur d'exécution très courante de Visual Basic. Elle spécifie que des données sont de **type incompatible**. Dans le cas présent, une chaîne de caractères *vide* ne peut être convertie au type de donnée Integer par la fonction **Cint**.

   - Cliquez sur le bouton Débogage. Visual Basic vous indique alors **l'instruction en erreur** :



```
Private Sub cmdCalculer_Click()
    lblTotal = Val(txtNb1) + CInt(txtNb2)
    txtNb1.SetFocus
End Sub
```

   - Arrêtez l'exécution en cliquant sur le bouton **Fin** de la barre d'outils Standard.

6. Utilisez la fonction Val au lieu de la fonction CInt :

```
lblTotal = Val(txtNb1) + Val(txtNb2)
```

7. Exécutez de nouveau et effectuez les tests suivants :
   - Entrez les nombres 5,5 et 4,7 ; calculez et observez le résultat. (virgule)
   - Entrez les nombres 5.5 et 4.7 ; calculez et observez le résultat. (point décimal)

   En présence d'une virgule, la fonction **Val** supprime les valeurs décimales. Cette fonction *ne reconnaît que le point comme symbole décimal*. Elle n'est donc pas valide dans le cas où *la virgule est le symbole décimal* fixé par les paramètres régionaux de Windows.

   Utilisez la fonction Val pour le traitement des nombres entiers et la fonction **CSng** pour les nombres avec décimales. La fonction CSng permet de convertir des nombres au type de donnée **Single**.

   Consultez la section Tableau des types de données, page 243.

8. Utilisez le fonction Csng :

```
lblTotal = CSng(txtNb1) + CSng(txtNb2)
```

9. Exécutez de nouveau et effectuez les tests suivants :
   - Entrez les nombres 5,5 et 4,7 ; calculez et observez le résultat. (OK)
   - Entrez les nombres 5.5 et 4.7 ; calculez et observez le résultat. (Erreur 13 : type incompatible)

   Lorsque la virgule est le symbole décimal fixé par les paramètres régionaux de Windows, le point est considéré comme un caractère *non numérique*, donc incompatible dans une conversion numérique.

10. Ajoutez les instructions en gras :

```
Private Sub cmdCalculer_Click()
  If IsNumeric(txtNb1) = False Or Not IsNumeric(txtNb2) Then
    MsgBox "L'un des nombres n'est pas numérique ou absent."
  Else
    lblTotal = CSng(txtNb1) + CSng(txtNb2)
  End If
  txtNb1.SetFocus
End Sub
```

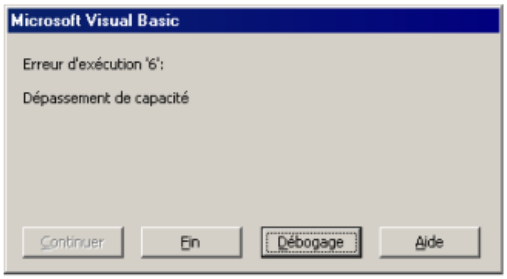
La fonction **IsNumeric** vérifie si les caractères de la zone sont numériques.

À titre d'exemple, dans le code qui précède, la fonction est comparée avec la valeur **False** et utilisée avec le mot **Not**. Les deux syntaxes sont équivalentes.

11. Exécutez et testez avec des nombres absents, des points décimaux, les signes + et –, des espaces à l'intérieur d'un nombre. Exemple : 5␣45 + 4␣4 (le caractère ␣ représente un blanc ou une espace). Observez le résultat ; les espaces sont acceptées.

12. Testez avec les nombres 5555555555 + 6666666666. Le résultat devrait être affiché selon la **notation scientifique** (lettre E dans le nombre), parce que l'étiquette n'est pas assez grande pour afficher le résultat.

13. Effectuez un test avec des nombres qui dépassent la plage de valeurs du type de données Single, par exemple, un nombre supérieur à 3 E38. Le message d'erreur suivant devrait apparaître :



- Sélectionnez le bouton Débogage afin de repérer la ligne de code qui a provoqué l'erreur. Par la suite, cliquez sur le bouton Fin de la barre d'outils Standard pour interrompre l'exécution.

L'erreur d'exécution 6, **Dépassement de capacité** est générée lorsque le contenu d'une variable ou une valeur à convertir dépasse la plage de valeurs prévue pour un type de données.

Les plages de valeurs de chacun des types de données sont spécifiées dans la section Tableau des types de données, page 243.

Dans le cas d'une zone de texte, la propriété **MaxLength** permet de limiter le nombre de caractères saisis.

14. Fixez la valeur 9 à la propriété MaxLength des deux zones de texte. Testez le changement avec des nombres de plus de 9 caractères.

15. Ajoutez la ligne en gras et testez le projet en utilisant des nombres avec plusieurs décimales. Observez l'affichage du résultat.

```
Private Sub cmdCalculer_Click()
  If IsNumeric(txtNb1) = False Or Not IsNumeric(txtNb2) Then
    MsgBox "L'un des nombres n'est pas numérique ou absent."
  Else
    lblTotal = CSng(txtNb1) + CSng(txtNb2)
    lblTotal = FormatNumber(lblTotal, 1)
  End If
  txtNb1.SetFocus
End Sub
```

> La fonction **FormatNumber** permet le formatage des nombres pour une meilleure présentation des résultats à l'utilisateur. Dans le cas présent, le résultat est affiché avec *une seule décimale*.

> Consultez la section Les fonctions de formatage, page 295, pour des détails sur les fonctions : FormatCurrency, FormatPercent, FormatNumber, FormatDateTime et Format.

16. Utilisez la fonction FormatCurrency au lieu de la fonction FormatNumber :

```
lblTotal = FormatCurrency(lblTotal)
```

> La fonction **FormatCurrency** formate les nombres selon le *format monétaire* défini dans les paramètres régionaux de Windows.

17. Testez avec plusieurs nombres. Observez le résultat.

***Étape 4 Coder un gestionnaire d'erreurs avec l'instruction* On Error**

1. Entrez les lignes de code illustrées en gras et mettez *les lignes grisées en commentaire* :

```
Private Sub cmdCalculer_Click()
  Dim strMess As String

  On Error GoTo GestionErreur
  If IsNumeric(txtNb1) = False Or Not IsNumeric(txtNb2) Then
    MsgBox "L'un des nombres n'est pas numérique ou absent."
  Else
    lblTotal = CSng(txtNb1) + CSng(txtNb2)
    lblTotal = FormatCurrency(lblTotal)
  End If
```

```
  txtNb1.SetFocus
  Exit Sub
GestionErreur:
  Select Case Err.Number
    Case 13
      strMess = "Donnée non numérique ou absente." & _
                vbNewLine & "Seulement les caractères " & _
                "0 à 9 , + et - sont permis."
    Case 6
      strMess = "L'un des nombres est trop élevé."
    Case Else
      strMess = "La cause de l'erreur est inconnue." & _
                vbNewLine & "Veuillez entrer d'autres nombres."
  End Select
  MsgBox "Impossible d'effectuer le calcul." & vbNewLine _
              & strMess, vbExclamation, "Erreur!"
  txtNb1.SetFocus
End Sub
```

L'instruction **On Error** permet de définir un **gestionnaire d'erreurs** qui permet d'intercepter toute erreur d'exécution. Lorsqu'une erreur est rencontrée, Visual Basic effectue un branchement à *l'étiquette* citée à la suite du mot **GoTo**. Les instructions qui suivent l'étiquette (GestionErreur) sont alors exécutées.

Consultez la section Activer un gestionnaire d'erreurs avec l'instruction *On Error*, page 274.

La propriété **Number** de l'objet **Err** retourne le numéro de l'erreur d'exécution. L'instruction Select Case permet d'affecter un *message d'erreur significatif* de l'erreur rencontrée, à la variable strMess. Le contenu de cette variable est affiché sur la *deuxième ligne* du message de la fonction MsgBox.

Consultez la section L'objet *Err* et les principaux codes d'erreur, page 277.

La constante **vbNewLine** provoque un changement de ligne. Elle permet d'afficher un message sur plusieurs lignes.

L'instruction **Exit Sub** provoque une *sortie de la procédure*, donc un branchement inconditionnel à la ligne End Sub. Dans le cas où les nombres

sont valides, l'absence de cette instruction provoque l'affichage d'un *faux message d'erreur*.

2. Exécutez et effectuez les tests suivants :
   - Calculez sans aucun nombre. Observez attentivement le message d'erreur. Celui-ci est réparti sur plusieurs lignes.
   - Entrez des nombres invalides et calculez.
   - Affectez la valeur 0 à la propriété MaxLength des deux zones de texte. Testez avec un nombre qui dépasse la plage de valeurs. Observez le message d'erreur.

Étape 5 Calculer avec des variables numériques

1. Mettez en commentaire toutes les lignes de code de la procédure cmdCalculer_Click.
2. Créez une nouvelle procédure cmdCalculer_Click à l'aide des fonctions d'édition Copier/Coller. Les lignes en gras sont de nouvelles lignes et les autres peuvent être copiées/collées.

```
Private Sub cmdCalculer_Click()
  Dim sngNb1 As Single, sngNb2 As Single, sngTotal As Single
  Dim strMess As String
  On Error GoTo GestionErreur
  'Calculer avec des variables numériques pour plus d'efficacité
  sngNb1 = CSng(txtNb1)
  sngNb2 = CSng(txtNb2)
  sngTotal = sngNb1 + sngNb2
  lblTotal = FormatNumber(sngTotal) 'formatage avec 2 décimales
  Exit Sub
GestionErreur:
  .
  . copiez/collez les lignes de la procédure en commentaire
  .
End Sub
```

Dans le code qui précède, les calculs sont effectués avec des **variables numériques** au lieu d'utiliser le contenu des zones de texte. L'utilisation de variables *réduit de façon appréciable le temps d'exécution* des calculs.

L'utilisation de variables numériques pour des calculs est une *excellente méthode de programmation*.

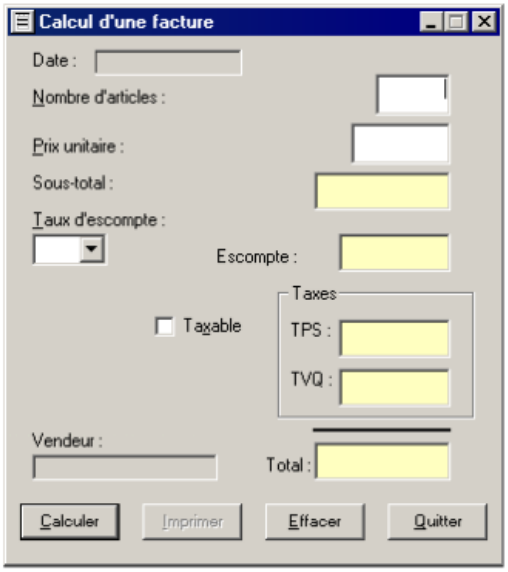
3. Exécutez le projet et assurez-vous de son bon fonctionnement. Testez avec plusieurs nombres valides et invalides.

3.6 EXERCICE 8 CRÉER LE PROJET CALCUL DE FACTURES

Étape 1 Préparation et enregistrement

1. Dans Visual Basic, ouvrez un nouveau projet de type Exe Standard.
2. Nommez le projet **Facture** et la feuille **frmFacture**. Enregistrez la feuille dans le fichier **Facture.frm** d'un nouveau dossier .../Visual Basic /Exercice 8 Facture. Enregistrez le projet dans le même dossier sous le nom **Facture.vbp**.

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de la feuille frmFacture :
   - Le titre de la barre de titre ;
   - L'icône de la barre de titre (Icons \Writing \NOTE01.ICO) ;
   - La taille n'est pas modifiable ;
   - Le bouton Agrandir n'est pas accessible ;
   - La feuille doit être affichée dans le centre de l'écran.

Dossier habituel des **icônes** fournies avec Visual Basic : Program Files \Microsoft Visual Studio \Common \Graphics \Icons.

2. Ajoutez les contrôles et fixez les propriétés :
   - Utilisez des étiquettes pour la date, le sous-total, l'escompte, la TPS, la TVQ, le vendeur et le total (lblDate, lblSousTotal, lblEscompte, lblTps, lblTvq, lblVendeur et lblTotal.
   - Utilisez des zones de texte pour le nombre d'articles (txtNombre) et le prix unitaire (txtPrix).
   - Ajoutez une zone de liste modifiable (cboTauxEsc), une case à cocher (chkTaxable) et un contrôle Line.
   - Ajoutez quatre boutons de commande : cmdCalculer, cmdImprimer, cmdEffacer et cmdQuitter. Le bouton cmdAnnuler est le *bouton par défaut* et le bouton cmdQuitter est le *bouton d'annulation*.
   - Le contenu des deux zones de texte doit être *aligné à droite* ainsi que le contenu des étiquettes lblSousTotal, lblEscompte, lblTps, lblTvq et lblTotal.
   - Le bouton de commande cmdImprimer *n'est pas accessible* au démarrage.
   - La zone txtNombre ne doit pas accepter plus de quatre caractères et la zone txtPrix ne doit pas accepter plus de sept caractères.

Étape 3 Coder les procédures

1. Codez le bouton cmdQuitter qui procède au déchargement de la feuille.
2. Codez la procédure suivante et testez le projet :

```
Private Sub Form_Load()
  lblDate = FormatDateTime(Date, vbLongDate)
  lblVendeur = InputBox("Entrez le nom du vendeur :", _
                        "Saisie du nom du vendeur")
End Sub
```

La fonction **FormatDateTime** formate la *date du jour* (fonction **Date**) avec le format complet (JJ mois AAAA).

Consultez la section Les fonctions de date et heure, page 292.

La fonction **InputBox** permet la saisie d'une chaîne de caractères dans une boîte de dialogue prédéfinie.

Consultez la section La fonction *InputBox*, page 259.

3. Ajoutez les lignes de code à la procédure Form_Load :

```
With cboTauxEsc
  'effacer le contenu de la liste modifiable
  .Clear
  'ajouter des valeurs à la liste modifiable
  .AddItem 0
  .AddItem 5
  .AddItem 10
  .AddItem 15
  .AddItem 20
  .AddItem 25
  .AddItem 30
  .AddItem 40
  .AddItem 50
  .AddItem 60
  .AddItem 70
  'affecter la troisième valeur de la liste comme
  'valeur par défaut (10%)
  .Text = .List(2)
End With
```

La méthode **Clear** d'un contrôle ComboBox permet d'effacer le contenu de celui-ci.

La méthode **AddItem** permet d'ajouter un élément dans la liste des valeurs d'un ComboBox. La première valeur ajoutée à la liste aura une valeur d'index égale à 0, la deuxième aura la valeur 1 et ainsi de suite. La propriété **Text** permet de fixer une *valeur par défaut* ; **List(2)** représente la troisième valeur de la liste (10 %).

Consultez la section Les contrôles *ListBox* et *ComboBox*, page 172.

La structure **With** permet de réduire le code de programmation lorsqu'un objet est cité sur plusieurs lignes de code consécutives. L'objet cboTauxEsc est identifié une seule fois à la suite du mot With au lieu d'être cité sur *toutes* les lignes.

Consultez la section 5.15 La structure *With*.

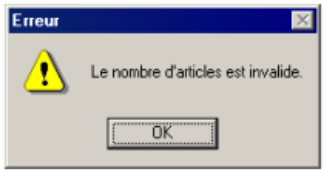
4. Testez le projet et observez le contenu de la liste modifiable.

Étant donné que cette liste est modifiable, l'utilisateur peut entrer un autre taux qui peut être invalide. La propriété **Style** offre un choix de liste *non modifiable* : 2 – Dropdown List.

5. Dans la fenêtre des propriétés, modifiez la propriété Style du contrôle cboTauxEsc afin que la liste des taux ne soit pas modifiable. Testez et observez le changement.

6. Codez la procédure cmdCalculer_Click :

   - Déclarez une variable locale sngSousTotal ;
   - Validez le nombre d'articles avec la fonction IsNumeric. S'il n'est pas valide, affichez le message qui suit et donnez le focus à la zone txtNombre ;



   - Validez le prix unitaire avec la fonction IsNumeric. S'il n'est pas valide, affichez le message qui suit et donnez le focus à la zone txtPrix ;

   - Testez les validations avant de poursuivre ;
   - Calculez le sous-total et affichez le résultat dans lblSousTotal avec le *format monétaire*. Testez avant de continuer ;
   - Déclarez une variable locale sngEscompte, calculez l'escompte (propriété Text de cboTauxEsc) et affichez le résultat avec le format monétaire. Testez avant de poursuivre ;
   - Réduisez le sous-total du montant d'escompte puisque les taxes sont calculées sur le sous-total *moins l'escompte*.
   - Calculez les taxes :

     - Déclarez 2 constantes symboliques dans la section **Declarations** du programme :

```
Option Explicit
Const csngTauxTps As Single = 0.07
Const csngTauxTvq As Single = 0.075
```

La section **Declarations** est accessible en sélectionnant les choix suivants dans le haut de la fenêtre de code :

Consultez la section Les constantes symboliques ou définies par le programmeur, page 250.

- Si la case à cocher Taxable est activée, calculez la TPS et la TVQ. Dans le cas contraire, la TPS et la TVQ prendront la valeur 0. La TPS est calculée sur le sous-total de la facture. La TVQ est calculée sur le sous-total de la facture plus la TPS.

Exemple

Sous-total = 600

TPS = 600 * 0.07

TPS = 42

TVQ = (600 + 42) * 0.075

TVQ = 48,15

- Affichez les montants de taxes avec 2 décimales (sans $).

- Calculez le total de la facture et affichez le résultat avec le format monétaire.

7. Codez la procédure du bouton Effacer. Effacez le contenu des zones de texte et des étiquettes qui affichent les résultats. Décochez la case Taxable et affectez la troisième valeur du contrôle ComboBox, comme *valeur par défaut*.

8. Complétez la procédure du bouton Imprimer :

```
Private Sub cmdImprimer_Click()
  - la feuille doit être imprimée avec un arrière-plan blanc
  - les 4 boutons de commande doivent être cachés avant d'imprimer

  Me.PrintForm    'impression de la feuille

  - les 4 boutons doivent réapparaître après l'impression
  - le bouton Imprimer ne doit plus être accessible
  - la couleur de l'arrière-plan de la feuille doit être modifiée
  - le bouton Effacer doit avoir le focus
End Sub
```

La méthode **PrintForm** de l'objet Form permet d'imprimer une feuille et les objets visibles qu'elle contient.

9. Après avoir calculé le total de la facture, rendez accessible le bouton Imprimer. Testez le bouton Imprimer.

10. Ajoutez la procédure suivante :

```
Private Sub txtPrix_KeyPress(KeyAscii As Integer)
  If KeyAscii = Asc(".") Then
    KeyAscii = Asc(",")
  End If
End Sub
```

L'événement **KeyPress** est déclenché à chaque fois que l'utilisateur presse une touche dans la zone de texte txtPrix. Si l'utilisateur tape un *point*, il est aussitôt converti en *virgule*. Le point n'étant pas numérique, cette petite astuce nous permet d'accepter le point décimal et la virgule.

Consultez la section Les événements clavier, page 206.

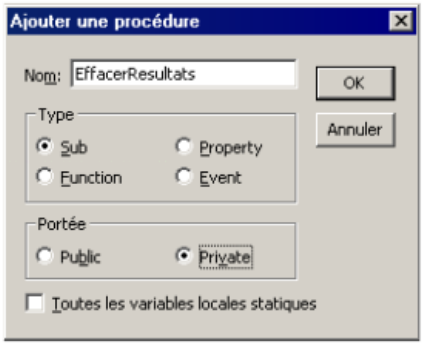
L'argument **KeyAscii** retourne un code ANSI. Pour comparer un code ANSI avec un caractère, ce dernier doit être converti avec la fonction **Asc**.

Consultez l'Annexe B Tableau des caractères ANSI.

11. Testez la saisie d'un point décimal dans la zone txtPrix.

12. Créez une **procédure personnalisée** :

- Sélectionnez Outils au menu et "Ajouter une procédure".



- Entrez le *nom* de la procédure EffacerResultats et sélectionnez une portée *Private*.

> **Outils/Ajouter une procédure** permet de créer une **procédure personnalisée**. L'instruction **Call** permet d'en faire l'appel.
>
>
> Consultez la section Déclaration et appel d'une procédure personnalisée *Sub*, page 235.

- Codez les instructions de la nouvelle procédure. Celle-ci doit effacer les 5 étiquettes qui affichent un résultat et rendre le bouton Imprimer inaccessible.

> Afin de *ne pas afficher de résultats faux* à l'utilisateur, aussitôt que celui-ci *change* une donnée, les résultats devraient être effacés.
>
> L'événement **Change** est déclenché aussitôt qu'un changement a été apporté dans un contrôle. Dans l'exercice présent, cet événement est utilisé pour effacer les résultats lorsqu'une information est modifiée.

13. Ajoutez les procédures suivantes :

```
Private Sub txtNombre_Change()
  Call EffacerResultats
End Sub

Private Sub txtPrix_Change()
  Call EffacerResultats
End Sub
```

14. Pour tester les 2 nouvelles procédures, calculez d'abord un total et par la suite, modifiez le nombre d'articles ou le prix unitaire. Tous les résultats devraient s'effacer.

15. Ajoutez les procédures suivantes :

```
Private Sub cboTauxEsc_Click()
  Call EffacerResultats
End Sub

Private Sub chkTaxable_Click()
  Call EffacerResultats
End Sub
```

> Lorsque l'utilisateur clique sur un taux d'escompte ou dans la case à cocher Taxable, les résultats doivent aussi être effacés puisque les données ont changées.

16. Testez les 2 nouvelles procédures.

17. Afin d'éviter une répétition de code, faites appel à la procédure EffacerResultats dans la procédure cmdEffacer_Click. Enlevez les lignes de code inutiles.

> Lorsqu'un projet se termine, prenez la bonne habitude de replacer les procédures de vos modules dans un ordre *logique* plutôt qu'*alphabétique*. Laissez une ligne vide entre chacune de vos procédures et enlevez les lignes vides en trop.

18. Voici une suggestion pour replacer vos procédures dans un ordre logique :

    Form_Load, cmdCalculer_Click, cmdImprimer_Click, cmd_Effacer_Click, txtNombre_Change, txtPrix_Change, txtPrix_KeyPress, cboTauxEsc_Click, chkTaxable_Click, EffacerResultats et cmdQuitter_Click.

3.7 EXERCICE 9 CRÉER LE PROJET FINANCEMENT AUTOMOBILE

Étape 1 Préparation, observation et enregistrement

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre3 /Exercice9 /Debut dans un nouveau dossier .../Visual Basic /Exercice 9 Auto.

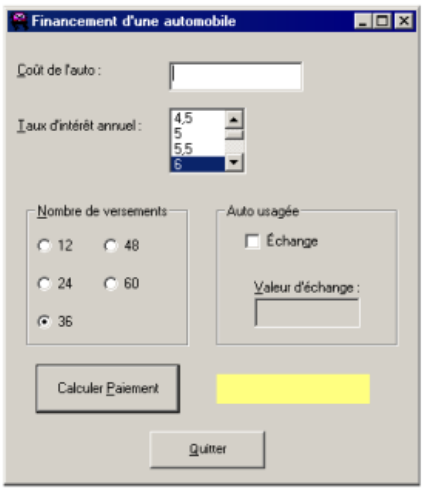
2. Exécutez le fichier .../Visual Basic /Exercice 9 Auto /**Auto.exe**.

> Cette application permet de calculer le montant du **paiement mensuel** à rembourser pour l'achat d'une voiture. Les données saisies sont le coût de l'auto, le taux d'intérêt annuel et le nombre de versements. Si le client possède une voiture usagée, la case Échange doit être cochée et l'utilisateur fournit la valeur de l'échange.

3. À partir de Visual Basic, créez un nouveau projet. Nommez le projet **Auto** et la feuille **frmAuto**.

4. Enregistrez la feuille et le projet dans le dossier .../Visual Basic /Exercice 9 Auto / (**Auto.frm** et **Auto.vbp**).

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de la feuille frmAuto :
   - Le titre de la barre de titre ;
   - L'icône de la barre de titre (Icons \Traffic \trffc16.ico).
2. Ajoutez les contrôles et fixez les propriétés :
   - Attribuez des noms personnels aux contrôles, tout en respectant les normes de programmation ;

Consultez la section 1.8 Convention d'affectation des noms aux objets.

   - Le coût et la valeur d'échange sont saisis dans un contrôle TextBox.
   - Utilisez une Zone de liste (contrôle ListBox) pour les taux d'intérêt annuels. Utilisez la propriété **List**, pour charger les différents taux dans la Fenêtre des propriétés. Entrez les 13 taux suivants :

4	4,5	5	5,5	6	6,5	7	7,5	8	8,5	9	9,5	10

Utilisez la combinaison de touches Ctrl+Enter pour changer de ligne. Entrez une virgule comme séparateur décimal (pas de point).

- Créez un groupe de contrôles pour les 5 boutons d'option. Ceux-ci porteront le même nom suivi des valeurs d'index 0 à 4. Le bouton "36 versements" doit être activé lors du démarrage.

> Un groupe de contrôles se crée en dessinant d'abord le premier bouton et en lui attribuant un nom (exemple : optVers). Les autres boutons sont créés en *copiant* le premier bouton.

- Fixez un arrière-plan *gris* (même ton que la feuille) à la zone "Valeur d'échange" et rendez-la *inaccessible* au démarrage.
- La taille de la police de *l'étiquette* qui affiche le paiement est 18 et l'arrière-plan est en couleur.
- Assurez-vous que le déplacement de la touche Tab est logique et que les raccourcis clavier sont opérationnels.

Étape 3 Coder les procédures

1. Programmez le bouton Quitter. Celui-ci procède au déchargement de la feuille.
2. Lors du chargement de la feuille, spécifiez le *cinquième* taux de la liste comme *valeur par défaut*.
3. Lorsque l'utilisateur coche la case Échange, la zone de texte "Valeur d'échange" doit être *accessible* et son arrière-plan doit être *blanc*. Elle doit aussi être *effacée* et recevoir le *focus*. Par contre, si la case Échange n'est pas cochée, il faut s'assurer que la zone de texte "Valeur d'échange" est *inaccessible*, que son *contenu est effacé* et que son arrière-plan est *gris*.
4. Calculez le paiement :
   - Déclarez une variable entière de portée module qui servira à conserver la valeur de l'index du bouton d'option actif. Nommez-la **mintNbVers**.
   - Codez la mise à jour de la variable mintNbVers lors d'un clic sur un bouton d'option :

```
Private Sub optVers_Click(Index As Integer)
  mintNbVers = Index
End Sub
```

Lorsque l'utilisateur sélectionne un nombre de versements, l'argument **Index** est automatiquement mis à jour et contient une valeur de 0 à 4. Pour effectuer le paiement, dans une autre procédure que optVers_Click, cette valeur doit être conservée dans une variable, dont la portée est pour l'ensemble du module.

- Adaptez la procédure suivante à vos noms de contrôles personnels :

```
Private Sub cmdPaiement_Click()

  lblPaiement.Caption = -Pmt(lstTaux.Text / 12 / 100, _
                    optVers(mintNbVers).Caption, _
                    CCur(txtCout.Text) - CCur(txtEchange.Text))
End Sub
```

La fonction financière **Pmt** permet de calculer un paiement mensuel, dont la syntaxe est la suivante :

Paiement = Pmt(Taux annuel/12, Nombre de versements, Montant financé)

Étant donné qu'un versement mensuel est une dette, le résultat de la fonction Pmt est *négatif*. Donc, pour obtenir un résultat positif, un signe – (moins) est placé en avant de Pmt.

Le *taux annuel* doit être divisé par 12 pour obtenir un *taux mensuel*.

La propriété **Text** du contrôle ListBox retourne le *taux d'intérêt* sélectionné. Celui-ci doit être divisé par 100 pour obtenir un pourcentage.

Exemple : 6 doit être converti à ,06 (6 %).

La propriété **Caption** des boutons d'option renvoie le nombre 12, 24, 36, 48 ou 60 qui représente *le nombre de versements* selon la valeur de mintNbVers (0 à 4).

Le *montant financé* est le résultat du coût de l'auto moins la valeur d'échange.

La fonction de conversion **CCur** convertit les zones de texte au format monétaire.

Consultez la section Les fonctions financières page 290 et la section Les fonctions de conversion de types de données, page 286.

En utilisant la *propriété par défaut* des contrôles TextBox, Label et ListBox, le code de l'instruction qui précède peut ainsi être réduit à la forme suivante :

```
lblPaiement = -Pmt(lstTaux / 12 / 100, _
              optVers(mintNbVers).Caption, _
              CCur(txtCout) - Ccur(txtEchange)
```

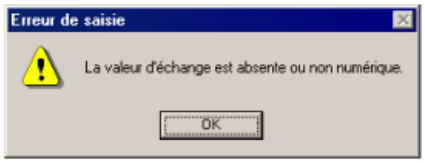
- Réduisez le code de la fonction Pmt en omettant les propriétés par défaut des contrôles TextBox, Label et ListBox.
- Exécutez le projet, entrez un coût de 1000, 12 versements, un taux de 8 % et une valeur d'échange de 500. Assurez-vous de l'exactitude du résultat qui devrait être 43,49421454.

> Dans les prochains tests, entrez *obligatoirement* un coût et une valeur d'échange *numériques* afin de ne pas provoquer l'erreur d'exécution "Type incompatible". Cette erreur sera corrigée lors de la validation des données.

- Attribuez le format monétaire au paiement afin d'afficher 43,49 $. Testez le changement avec les *mêmes données*.

5. Validez les données :
   - Avant de calculer, vérifiez si le coût est numérique. S'il n'est pas numérique, affichez un message d'erreur et redonnez le focus à la zone de saisie du coût. Testez la validation.

   - Avant de calculer, vérifiez si la case Échange est cochée. Dans ce cas, affectez la valeur numérique 0 à la zone de texte "Valeur d'échange". Testez le cas où il n'y a pas d'échange.
   - Avant de calculer, vérifiez si la valeur d'échange est numérique, dans le cas où la case Échange est cochée. Si elle n'est pas numérique, affichez un message d'erreur et redonnez le focus à la zone de saisie de l'échange. Testez la validation.



6. Afin de ne pas afficher de résultats faux, le paiement doit être effacé automatiquement dans les cas suivants :
   - Lorsque l'utilisateur choisit un autre taux d'intérêt ;
   - Lorsque l'utilisateur choisit un autre nombre de versements ;
   - Lorsque l'utilisateur tape un autre coût ;
   - Lorsque l'utilisateur tape une autre valeur d'échange.

L'opération d'effacement automatique du paiement ne requiert pas une procédure personnalisée, parce qu'une seule instruction est nécessaire. Dans le cas présent, il vaut mieux répéter l'instruction.

7. Avant de continuer, testez minutieusement le projet avec des données valides et invalides.

Étape 4 Ajouter une feuille au projet

1. Sélectionnez au menu Projet /Ajouter une feuille /Onglet Nouveau /Form /Ouvrir et nommez la nouvelle feuille **frmBienvenue**. Enregistrez-la sous le nom **Bienvenue.frm**.

2. Créez l'interface graphique qui suit en tenant compte des consignes à la page suivante :

Quelques consignes :

- Pour afficher l'ordinateur, sélectionnez la propriété **Picture** de la feuille et le fichier Computer.wmf stocké dans le dossier Program Files /Microsoft Visual Studio /Common /Graphics /Metafile /Business.

La propriété **Picture** d'une feuille permet de fixer une image en arrière-plan.

En plus des icônes, Visual Basic fournit des images de type Metafile stockées dans le dossier Program Files /Microsoft Visual Studio /Common /Graphics /Metafile.

- Pour supprimer la barre de titre de la feuille, affectez la valeur 0–None à la propriété BorderStyle.
- Utilisez une étiquette pour afficher le mot Bienvenue ! dans l'écran de l'image. Modifiez la taille des caractères et la couleur de ceux-ci.
- Affichez la feuille au centre de l'écran lors du démarrage.

3. Modifiez l'objet de démarrage :
   - Sélectionnez au menu Projet /Propriétés de Auto /Objet de démarrage /frmBienvenue /OK.

L'objet de démarrage est généralement une feuille qui est affichée à l'écran, au démarrage d'un projet.

Consultez la section 5.3 L'objet de démarrage.

   - Exécutez le projet, observez la nouvelle feuille et cliquez sur le bouton Fin de la barre d'outils pour arrêter l'exécution.
4. Changez l'apparence du pointeur de la souris :
   - Ajoutez un contrôle Timer à la feuille, nommez-le tmrTemps et fixez sa propriété Interval à 5000 afin d'afficher la feuille de bienvenue pendant 5 secondes.
   - Sélectionnez le choix 13–Arrow and Hourglass pour la propriété **MousePointer** de la feuille.

La propriété **MousePointer** d'un objet permet de changer l'apparence du pointeur de la souris, lorsque celui-ci est pointé. Dans le cas présent, un **sablier** est utilisé afin que l'utilisateur comprenne qu'il n'a pas d'autre choix que d'attendre...

Consultez la section Les propriétés MousePointer et MouseIcon, page 194.

- Ajoutez le code de la feuille frmBienvenue :

```
Private Sub tmrTemps_Timer()
  frmAuto.Show
  Unload Me
End Sub
```

Lorsque 5 secondes sont écoulées, la feuille frmAuto est affichée par la méthode **Show** et la feuille frmBienvenue est déchargée de la mémoire par l'instruction Unload.

Consultez la section Les principales méthodes d'une feuille page 163 pour plus de détails sur la méthode Show.

- Exécutez le projet et pointez l'ordinateur avec la souris. Observez le sablier.

Étape 5 Découvrir et corriger un bogue

1. Exécutez le projet et effectuez le test suivant :

   Coût = 1000
   Taux = 7 %
   Versements = 36
   Le résultat devrait être 86,53 $.

   Est-ce possible ? Rembourser 86,53 $ par mois pendant 3 ans, fait un total de 3115,08 $. C'est beaucoup trop pour rembourser 1 000 $. C'est évident qu'il y a un *bogue*...

2. Testez de nouveau avec un coût de 1000, un taux de 7 % et un nombre de versements de 12. Observez bien le résultat. Ceci devrait vous donner une piste pour trouver et corriger l'erreur de programmation.

Vous trouverez la solution du bogue dans le code de programmation du projet Auto.vbp stocké dans le dossier Chapitre3 /Exercice9 /Fin.

3. Lorsque le bogue est résolu, replacez vos procédures dans un ordre *logique*. Laissez une ligne vide entre chacune de vos procédures et enlevez les lignes vides en trop.

3.8 EXERCICE 10

Partie 1 Modifier le projet *Calcul de factures*

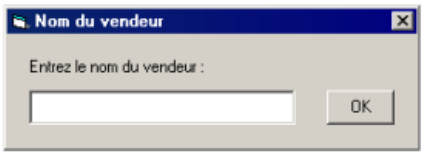
Étape 1 Préparation

1. À partir de l'Explorateur de Windows, copiez votre dossier .../Visual Basic /Exercice 8 Facture dans un nouveau dossier .../Visual Basic /Exercice 10 Facture.

2. Ouvrez le projet .../Visual Basic /Exercice 8 Facture /Facture.vbp.

Étape 2 Ajouter une feuille et un module standard au projet

1. Mettez en commentaire la fonction InputBox de la procédure Form_Load. Ajoutez la feuille illustrée ci-après au projet, nommez-la **frmNomVendeur** et enregistrez-la sous le nom **NomVendeur.frm**.



   - Utilisez une zone de texte et un bouton de commande. Nommez-les selon les normes de programmation. La zone de texte doit recevoir le focus au démarrage.
   - La feuille doit être centrée dans l'écran et sa taille n'est pas modifiable.
   - Le bouton OK doit être le *bouton par défaut*. Ansi, si l'utilisateur tape la touche Enter, le code du bouton OK sera automatiquement exécuté.

2. Définissez la feuille frmNomVendeur comme *objet de démarrage* du projet.

3. Lorsque le bouton OK est sélectionné, déchargez la feuille frmSaisieNom de la mémoire. Affichez la forme frmFacture dans un événement Form_Unload.

Il est possible que l'utilisateur ferme la feuille par le bouton Fermer de la barre de titre. Dans ce cas, frmFacture sera aussi affichée.

4. Ajoutez un **module standard** au projet afin d'y déclarer une variable de *portée globale* pour le nom du vendeur. Sélectionnez au menu Projet /Ajouter un module /Onglet Nouveau /Module / Ouvrir.

Consultez la section 5.1 Les modules pour plus de détails sur les modules standards.

5. Donnez le nom ModuleStandard (propriété Name) au nouveau module et enregistrez-le sous le nom ModuleStandard.bas.

6. Ajoutez la déclaration suivante dans la partie Déclarations du module standard :

```
Public gstrNomVendeur As String
```

La variable gstrNomVendeur est utilisée par 2 feuilles différentes. Elle doit donc être obligatoirement de **portée globale** ou **publique**. Les variables publiques doivent être déclarées dans un module standard.

Consultez la section La portée des variables, page 245.

7. Lorsque l'utilisateur entre le nom du vendeur, celui-ci doit apparaître sur la facture dans l'étiquette lblVendeur. Codez les instructions et testez.

***Étape 3 Intercepter des caractères lors de l'événement* KeyPress**

1. Ajoutez la procédure suivante dans le module de code de la forme frmNomVendeur :

```
Private Sub txtNomVendeur_KeyPress(KeyAscii As Integer)
  Select Case KeyAscii
    Case vbKeyA To vbKeyZ       'majuscules
    Case Asc("a") To Asc("z")   'minuscules
    Case vbKeySpace, vbKeyReturn, Asc("-")
    Case Else
      KeyAscii = 0
  End Select
End Sub
```

Seulement ces touches sont acceptées dans le nom du vendeur : les *lettres majuscules* et *minuscules*, le *caractère d'espacement* (vbKeySpace), la touche *Enter* (vbKeyReturn) et le *tiret*. Toutes les autres touches sont annulées par l'instruction `KeyAscii = 0`.

Plusieurs constantes représentent le code de différentes touches. Si un caractère ne correspond à aucune constante, la fonction Asc est utilisée pour convertir le caractère en valeur AscII.

Consultez la section Les événements clavier page 206 et la section A.1 Les constantes des codes de touches.

2. Exécutez le projet et entrez des caractères valides et invalides dans le nom du vendeur. Pressez la touche Retour Arrière (BackSpace) ; celle-ci ne devrait pas être acceptée.

3. Acceptez la touche BackSpace dans le nom du vendeur. Utilisez la constante **vbKeyBack**. Testez le changement.

4. Modifiez la procédure txtPrix_KeyPress de la forme frmFacture :

```
Private Sub txtPrix_KeyPress(KeyAscii As Integer)
  Select Case KeyAscii
    Case vbKeyBack
    Case vbKey0 To vbKey9
    Case Asc(","), Asc(".")
      If KeyAscii = Asc(".") Then
        KeyAscii = Asc(",")
      End If
      If InStr(txtPrix, ",") > 0 Then
        KeyAscii = 0
      End If
    Case Else
      KeyAscii = 0
  End Select
End Sub
```

Seulement ces touches sont acceptées dans le prix unitaire : *BackSpace*, les *chiffres 0 à 9*, la *virgule* et le *point*. Les points sont convertis en virgule et une *seule* virgule peut être acceptée. La fonction **InStr** recherche la présence d'une virgule dans la zone de texte txtPrix. Si c'est le cas, toute autre virgule est rejetée.

Consultez la section Les fonctions de chaînes de caractères, page 287.

5. Dans la zone de texte txtNombre, acceptez seulement les chiffres 0 à 9 et la touche BackSpace. Toutes les autres touches doivent être refusées. Testez la saisie du nombre d'articles.

6. Enregistrez les composantes du projet.

Partie 2 Introduction aux tableaux en mémoire

Étape 1 Ajouter un tableau en mémoire au projet Groupe de contrôles

1. À partir de l'Explorateur de Windows, copiez votre dossier .../Visual Basic /Exercice 6 GroupeControle dans un nouveau dossier .../Visual Basic /Exercice 10 Cartes.
2. Ouvrez le projet .../Visual Basic /Exercice 10 Cartes /GroupeControle.vbp.
3. Modifiez le nom de la feuille pour **frmCartes** et le nom du projet pour **Cartes**. Enregistrez la feuille sous le nom **Cartes.frm** et le projet sous le nom **Cartes.vbp** dans le dossier .../Visual Basic /Exercice 10 Cartes.
4. Exécutez le projet et observez le contenu de l'étiquette lors d'un clic sur une image.
5. Déclarez un tableau en mémoire dans la section Déclarations du module de code :

```
Dim mstrCouleur(3) As String
```

L'instruction **Dim** déclare un tableau de 4 chaînes de caractères représentées par un **indice** variant de 0 à 3.

Représentation graphique du tableau en mémoire :

Trèfle	Carreau	Coeur	Pique
mstrCouleur(0)	mstrCouleur(1)	mstrCouleur(2)	mstrCouleur(3)

Consultez la section 5.18 Les tableaux en mémoire.

6. Affectez les valeurs au tableau :

```
Private Sub Form_Load()
  mstrCouleur(0) = "Trèfle"
  mstrCouleur(1) = "Carreau"
  mstrCouleur(2) = "Coeur"
  mstrCouleur(3) = "Pique"
End Sub
```

7. Mettez en commentaire toutes les lignes de l'instruction Select Case de la procédure imgCouleur_Click et la remplacez par l'instruction suivante :

```
lblInfo = "Couleur sélectionnée : " & mstrCouleur(Index)
```

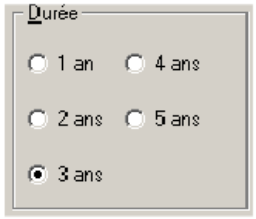
8. Testez le projet et observez.

Les **tableaux en mémoire** sont souvent une solution pour éviter des *répétions illogiques* dans le code de programmation.

9. Enlevez toutes les lignes de code inutiles et enregistrez les composantes du projet.

***Étape 2 Ajouter un tableau en mémoire au projet* Financement automobile**

1. À partir de l'Explorateur de Windows, copiez votre dossier .../Visual Basic /Exercice 9 Auto dans un nouveau dossier .../Visual Basic /Exercice 10 Auto.
2. Ouvrez le projet .../Visual Basic /Exercice 10 Auto /Auto.vbp.
3. Modifiez le cadre "Nombre de versements" comme suit :



> Notez que le titre du cadre et le titre des boutons d'option ont été modifiés.

4. Exécutez le projet et calculez un paiement ; le programme devrait *boguer*...
5. Trouvez la cause de l'erreur et déclarez un *tableau en mémoire* de type *Integer*, dans la section Déclarations du module de code. Chargez les valeurs dans le tableau.

   Représentation graphique du tableau :

12	24	36	48	60

6. Modifiez le calcul du paiement afin d'utiliser les *valeurs du tableau en mémoire* au lieu de la propriété Caption des boutons d'option. Testez minutieusement la modification.
7. Enregistrez les composantes du projet.

3.9 *EXERCICE 11 CRÉER* LE PROJET EMPLOIS EN INFORMATIQUE

Étape 1 Préparation, observation et enregistrement

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre3 /Exercice11 /Debut dans un nouveau dossier .../Visual Basic /Exercice 11 Emplois.
2. Exécutez le fichier .../Visual Basic /Exercice 11 Emplois /**Emplois.exe**. Sélectionnez différents choix, calculez les salaires et observez les résultats *fictifs*.

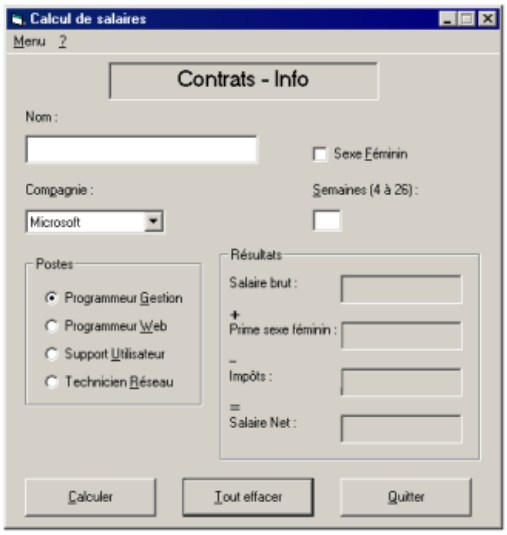
Ce projet calcule le salaire qu'une personne œuvrant en informatique peut gagner, en signant un contrat avec une multinationale aux États-Unis. Les femmes bénéficient d'une prime afin de les encourager à choisir une carrière en informatique.

3. Observez les choix du *menu*. L'aide au menu (?) affiche un message *humoristique*. Observez attentivement le contenu du message qui apparaît à la fin de l'exécution.

4. À partir de Visual Basic, créez un nouveau projet. Nommez le projet **Emplois** et la feuille **frmEmplois**.

5. Enregistrez la feuille et le projet dans le dossier .../Visual Basic /Exercice 11 Emplois / (**Emplois.frm** et **Emplois.vbp**).

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de la feuille frmEmplois et ajoutez les contrôles en tenant compte des consignes suivantes :

   - Utilisez les noms suivants :

     txtNom, chkFeminin, cboCie, txtSemaine, optPoste(0) à optPoste(3), lblSalBrut, lblPrime, lblImpot, lblSalNet, cmdCalculer, cmdToutEffacer, cmdQuitter

- La liste modifiable affichant les compagnies, *ne doit pas être modifiable* par l'utilisateur. Entrez ces 3 noms de compagnie en mode création :

  IBM
  Apple
  Microsoft

- Microsoft doit être la compagnie sélectionnée au démarrage.
- "Programmeur gestion" est le poste sélectionné au démarrage.
- Les signes +, - et = dans les étiquettes ont une taille de 12 et un style gras.
- La feuille doit s'afficher dans le centre de l'écran et le titre général doit posséder une taille de 14.
- Tous les résultats des calculs sont affichés dans des étiquettes dont la taille des caractères est 12 et le style gras. Alignez les résultats à *droite*.

2. Créez le menu :

- Sur la barre d'outils Standard, cliquez sur l'icône **Créateur de menus** ;
- Caption : &Menu, Name : mnuMenu ; cliquez sur le bouton

;
- Cliquez sur la flèche *droite* ;
- Caption : &Calculer, Name : mnuCalculer, ShortCut : choisissez F2 dans la liste et cliquez sur le bouton Suivant ;
- Caption : &Effacer, Name : mnuEffacer ; cliquez sur le bouton Suivant et sur la flèche *droite ;*
- Caption : Effacez les &données, Name : mnuEffacerDon, ShortCut : Ctrl + D ; cliquez sur le bouton Suivant ;
- Caption : Effacer les &résultats, Name : mnuEffacerRes, ShortCut : Ctrl + R ; cliquez sur le bouton Suivant ;
- Caption : &Tout effacer, Name : mnuToutEffacer, ShortCut : Ctrl + T ; cliquez sur le bouton Suivant et sur la flèche *gauche* ;
- Caption : &Quitter, Name : mnuQuitter, ShortCut : F7 ; cliquez sur le bouton Suivant et sur la flèche *gauche* ;

- Caption : &?, Name : mnuAide ;
- Pressez le bouton OK.

3. Observez le menu construit en cliquant sur les différents choix.

Consultez la section Le Créateur de menus, page 215.

4. Sous le menu, tracez une ligne blanche afin de donner une apparence 3D.
5. **Verrouillez tous les contrôles** afin de ne pas les déplacer par inadvertance. Sélectionnez au menu Format / Verrouiller les contrôles.

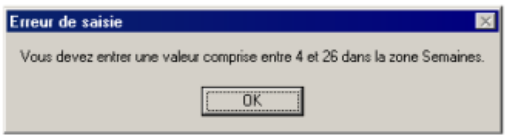
En verrouillant les contrôles, il n'est plus possible de les déplacer. Pour les déplacer à nouveau, sélectionnez le choix Verrouiller les contrôles par le menu Format ou le menu contextuel.

Consultez la section Le menu Format, page 56.

6. Exécutez le projet et assurez-vous que les déplacements avec la touche Tab sont logiques et que les raccourcis clavier sont fonctionnels.

Étape 3 Coder les procédures

1. Codez le bouton cmdCalculer :
   - Le nombre de semaines doit être validé avant d'effectuer les calculs. S'il n'est pas valide, affichez le message qui suit, effacez le contenu de txtSemaine et donnez le focus à la zone de texte txtSemaine.



   - Pour plus de rapidité, dans l'exécution des calculs, utilisez les variables locales suivantes :

```
Dim curSalBrut As Currency
Dim curPrime As Currency
Dim curImpot As Currency
Dim curSalNet As Currency
```

   - Les salaires sont calculés pour l'ensemble du contrat c'est-à-dire pour **toutes les semaines**.

- Une semaine de travail compte 35 heures et le taux horaire pour chaque poste est :

Programmeur Gestion	50,95
Programmeur Web	60,45
Support à l'usager	39,95
Technicien Réseau	54,90

- Utilisez un tableau en mémoire de portée module et de type Single pour stocker les taux horaire.

50,95	60,45	39,95	54,90
msngTaux(0)	msngTaux(1)	msngTaux(2)	msngTaux(3)

Pour affecter une valeur décimale, un *point* doit être utilisé au lieu d'une *virgule*, cette dernière étant un séparateur d'arguments en Visual Basic.

- Affichez tous les résultats avec le format *monétaire*.
- Les personnes de sexe féminin bénéficient d'une prime de 5 % du salaire brut. Pour une personne de sexe masculin, affichez 0,00 $ (format monétaire).
- L'impôt est de 30 % du salaire brut. La prime pour les femmes n'est pas imposable.
- Réduisez l'impôt de 500 $ si la compagnie est IBM.

2. Testez les calculs avec les 2 exemples suivants :

Femme	Salaire brut	17 832,50 $
IBM	Prime	891,63 $
10 semaines	Impôt	4 849,75 $
Programmeur gestion	Salaire net	13 874,38$
Homme	Salaire brut	23 058,00 $
Microsoft	Prime	0,00 $
12 semaines	Impôt	6 917,40 $
Technicien réseau	Salaire net	16 140,60 $

3. Testez le projet avec des données invalides dans la zone Semaines (donnée absente et non numérique).

4. Dans la zone de saisie du **nom** :
   - N'acceptez que les lettres minuscules et majuscules, le caractère d'espacement, le tiret, la touche BackSpace et le touche Escape. Utilisez la constante **vbKeyEscape** ;
   - Le *tiret* - doit être converti en espace (Marie-France devient Marie France) ;
   - Si l'utilisateur presse la touche *Esc*, effacez la zone de texte ;
   - Testez la saisie du nom en utilisant des caractères non permis, le tiret, des lettres majuscules/minuscules, la touche Esc, l'espace et la touche BackSpace. Assurez-vous que la touche Esc efface tous les caractères entrés et que le tiret est converti en espace.
5. Codez le choix **Calculer** au menu et testez le projet :

```
Private Sub mnuCalculer_Click()
  Call cmdCalculer_Click
End Sub
```

> Étant donné que le *bouton Calculer* et le *choix Calculer au menu* effectuent les *mêmes tâches*, il est logique de faire appel à la même procédure.

6. Codez le bouton cmdQuitter, qui procède au déchargement de la feuille, et le *choix correspondant au menu*. Testez le bouton et le choix au menu.
7. Codez l'effacement des données :

> Il y a 3 choix pour l'effacement des données :
>
> | | |
> |---|---|
> | **Effacer les données :** | nom, sexe féminin et nombre de semaines (menu) |
> | **Effacer les résultats :** | salaire brut, prime, impôt et salaire net (menu) |
> | **Tout effacer :** | les données et les résultats (bouton et menu) |

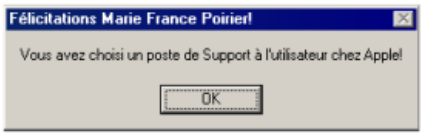
   - Codez 2 procédures personnalisées : **EffacerDonnees** et **EffacerResultats**. La première efface les données (nom, sexe féminin, nombre de semaines) et la seconde efface les résultats (salaire brut, prime, impôt, salaire net).
   - Codez le bouton "Tout effacer" et les 3 sous-choix du menu Effacer. Faites appel aux procédures personnalisées EffacerDonnees et EffacerResultats. Testez le bouton "Tout effacer" et tous les choix du menu Effacer.
8. Codez le choix **Aide (?)** au menu :
   - Ajoutez au projet, la feuille Aide.frm qui est stockée dans le dossier .../Visual Basic /Exercice 11 Emplois.

- Codez le choix mnuAide qui affiche la forme frmAide.
- Dans la forme frmAide, codez le bouton cmdOk qui procède au déchargement de la forme frmAide.
- Testez le choix Aide et cliquez d'abord sur le bouton OK afin d'accéder à la forme frmEmplois.
- Testez de nouveau le choix Aide mais cette fois, cliquez dans la forme frmEmplois lorsque la forme frmAide est affichée... La forme Aide se retrouve en arrière de la forme frmEmplois.

> La méthode Show affiche une feuille de façon **non modale**. Pour obliger l'utilisateur à presser le bouton OK avant d'activer la forme frmEmplois, il faut afficher la forme Aide de façon **modale**, comme suit :
>
> ```
> frmAide.Show vbModal
> ```
>
> Consultez la section Les principales méthodes d'une feuille, page 163.

- Affichez la forme frmAide de façon modale et testez le changement.

9. Ajoutez un message de fin de programme :
   - Lors du déchargement de la feuille (bouton Quitter ou bouton Fermer), affichez un message à l'utilisateur. Observez l'exemple qui suit :



   Ce message contient 3 éléments : nom de l'employé, la description du poste et la compagnie.

   Utilisez un tableau en mémoire pour conserver une description de chacun des postes :

Programmeur de gestion
Programmeur de pages Web
Support à l'utilisateur
Technicien de réseau

10. Testez la fin du programme (bouton Quitter et bouton Fermer) et vérifiez le texte du message.

3.10 EXERCICE 12 MODIFIER LE PROJET ÉDITION DE TEXTE

Étape 1 Préparation

1. À partir de l'Explorateur de Windows, copiez votre dossier .../Visual Basic /Exercice 5 Edition dans un nouveau dossier .../Visual Basic /Exercice 12 Edition.
2. Ouvrez le projet .../Visual Basic /Exercice 12 Edition /Edition.vbp.
3. Réduisez le code de la procédure optCouleur_Click à **une seule instruction** en utilisant la propriété BackColor des boutons d'option. Testez la modification.
4. Créez le menu suivant :

&Édition	mnuEdition	
---&Couper	mnuEditionCouper	Ctrl+X
---Co&pier	mnuEditionCopier	Ctrl+C
---Co&ller	mnuEditionColler	Ctrl+V
---&Supprimer tout	mnuEditionSupprimer	Ctrl+S
---Sélectionner &tout	mnuEditionSelectionner	Ctrl+A

5. Codez les procédures suivantes :

```
Private Sub mnuEditionCopier_Click()
' Place le texte sélectionné dans le Presse-papiers
  Clipboard.SetText txtZoneTexte.SelText
End Sub

Private Sub mnuEditionCouper_Click()
' Place le texte sélectionné dans le Presse-papiers
  Clipboard.SetText txtZoneTexte.SelText
' Supprime le texte sélectionné de la zone de texte
  txtZoneTexte.SelText = ""
End Sub

Private Sub mnuEditionColler_Click()
' Place le texte du Presse-papiers dans la zone de texte
  txtZoneTexte.SelText = Clipboard.GetText
End Sub
```

La propriété **SelText** d'une zone de texte, renvoie la chaîne contenant un texte sélectionné. Si aucun texte n'est sélectionné, la propriété SelText renvoie une chaîne vide ("").

La méthode **SetText** de l'objet **Clipboard** (Presse-papiers de Windows), place une chaîne de caractères dans l'objet Clipboard.

La méthode **GetText** de l'objet **Clipboard** renvoie une chaîne de caractères contenue dans le Presse-papiers de Windows.

Consultez la section Le contrôle *TextBox* page 168 et la section 4.11 L'objet *ClipBoard* (le Presse-Papiers).

6. Testez les choix Couper/Copier/Coller. Sélectionnez un texte avant de couper ou copier.

7. Codez la procédure suivante :

```
Private Sub mnuEditionSelectionner_Click()
' Sélection de la zone de texte, de la première position
' (0) à la dernière (longueur du texte)
  txtZoneTexte.SelStart = 0
  txtZoneTexte.SelLength = Len(txtZoneTexte.Text)
End Sub
```

Dans cet exemple, la propriété **SelStart** d'une zone de texte définit la première position d'un texte à sélectionner et la propriété **SelLength** définit le nombre de caractères qui doivent être sélectionnés.

La fonction **Len** retourne la longueur de la zone de texte.

Consultez la section Le contrôle *TextBox* page 168 et la section Les fonctions de chaînes de caractères (Len), page 287.

8. Testez le choix "Sélectionner tout" et ensuite le choix Copier. Sélectionnez de nouveau le choix "Sélectionner tout" et tapez un caractère dans la zone de texte ; le texte devrait disparaître.

Cette technique de sélection peut être utile lors d'une validation de certaines données. Lorsqu'une donnée est erronée, on affiche un message d'erreur, on donne le focus à la zone en erreur et on *sélectionne le texte erroné*. L'utilisateur peut alors constater son erreur et retaper une autre donnée.

9. Codez la procédure mnuEditionSupprimer_Click() qui doit effacer le contenu de la zone de texte. Testez le choix "Supprimer tout".

10. Codez l'accessibilité des choix du menu, en modifiant leur propriété Enabled :

```
Private Sub mnuEdition_Click()
   Rendre accessible les choix Couper et Copier seulement si un texte est
   sélectionné dans la zone de texte.

   Rendre accessible le choix Coller seulement si le presse-papiers
   contient du texte.

   Rendre accessible les choix Supprimer tout et Sélectionner tout
   seulement si la zone de texte contient du texte.
End Sub
```

11. Ajoutez la procédure suivante :

```
Private Sub Form_Load()
  Clipboard.Clear
End Sub
```

> La méthode **Clear** de l'objet Clipboard vide le contenu du Presse-papiers de Windows. Au démarrage du projet, il est possible que le presse-papiers contienne du texte qui a été coupé ou copié par d'autres applications Windows. En effaçant le contenu du presse-papiers lors de l'événement Form_Load, le choix Coller ne sera pas accessible au démarrage, puisque le presse-papiers est vide.

12. Avant de terminer, replacez les procédures dans un ordre logique et enlevez les lignes vides inutiles.

    Séquence suggérée :

    Form_Load, optCentre, optGauche, optDroite, chkGras, chkItalique, chkSouligne, optCouleur, mnuEdition, mnuEditionCouper, mnuEditionCopier, mnuEditionColler, mnuEditionSupprimer, mnuEditionSelectionner

3.11 EXERCICE 13 MODIFIER LE PROJET CALCUL DE MOYENNES

Étape 1 Préparation, observation et enregistrement

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre3 /Exercice13 /Debut dans un nouveau dossier .../Visual Basic /Exercice 13 Moyennes.
2. Exécutez le fichier .../Visual Basic /Exercice 13 Moyennes /**Moyennes.exe**.
   - Entrez des notes entières et pressez la touche Enter (au lieu de Tab) pour changer de zone de texte. Calculez la moyenne et observez les résultats ;

- Entrez des notes supérieures à 100 et observez ;
- Entrez des notes non numériques et observez ;
- Entrez des notes avec des décimales et observez ;
- Tapez une note et utilisez la touche Escape pour l'effacer ;
- Calculez la moyenne sans aucune note ;
- Testez le bouton Effacer ;
- Fermez l'application et observez.

> Cette application a été conçue pour un utilisateur qui n'est pas à l'aise avec la touche Tab pour confirmer l'entrée d'une donnée. Il préfère utiliser la touche Enter, celle-ci étant plus facile d'accès aux droitiers et plus large que la touche Tab.
>
> En utilisant seulement la touche Enter, l'utilisateur peut entrer ses notes, calculer la moyenne et effacer toutes les informations rapidement.

3. À partir de Visual Basic, ouvrez le projet .../Visual Basic /Exercice 13 Moyennes /**Moyennes.vbp**.
4. Observez la forme frmNotes. Elle contient 2 *groupes de contrôles* lblEtiq(0) à lblEtiq(4) et txtNote(0) à txtNote(4).
5. Observez la forme frmMoyenne.

Étape 2 Coder les procédures de la forme frmNotes

1. Codez la procédure qui suit :

```
Private Sub cmdEffacer_Click()
  Dim I As Integer
  For I = 0 To 4 Step 1
    txtNote(I) = ""
  Next I
End Sub
```

> Cette procédure efface les 5 notes par une boucle **For/Next**. La boucle **For/Next** permet d'exécuter un certain nombre de fois un groupe d'instructions comprises entre le mot **For** (**début** de la boucle) et le mot **Next** (**fin** de la boucle).
>
> Une variable *numérique entière* (exemple : I) sert de **compteur** pour calculer le nombre de répétitions. Selon les normes de programmation, ce type de variable qui représente un indice ou un index, peut être déclaré *sans abréviation*.

Dans l'expression : `For I = 0 To 4 Step 1`

`I = 0` :	spécifie la valeur de I au **début** de l'exécution de la boucle ;
`To 4` :	spécifie la valeur de I à la **fin** de l'exécution de la boucle ;
`Step 1` :	représente l'**incrément** (facteur d'augmentation de la variable après chaque exécution). Si l'augmentation est de 1, cette partie peut être omise (exemple : `For I = 0 To 4`) ;
`Next I` :	délimite la fin du groupe d'instructions à répéter (la variable `I` est optionnelle).

Consultez la section La boucle *For/Next*, page 268.

2. Exécutez le projet, entrez 5 notes et pressez le bouton Effacer. Observez le résultat.
3. Simplifiez la syntaxe de la boucle For/Next en supprimant les éléments optionnels :

```
For I = 0 To 4
  txtNote(I) = ""
Next
```

4. Pour visualiser l'exécution d'une boucle, exécutez le projet en mode *Pas à pas*. Pressez la touche F8 pour débuter l'exécution, entrez des notes et sélectionnez le bouton Effacer. Pressez F8, pointez la **variable I** et observez sa valeur affichée dans une info-bulle. Pressez plusieurs fois F8 et observez la progression du compteur I.
5. Utiliser un **accumulateur** :

```
Private Sub cmdCalculer_Click()
  Dim intTotal As Integer
  Dim I As Integer
  For I = 0 To 4
    intTotal = intTotal + Val(txtNote(I).Text)
  Next
End Sub
```

Cette boucle permet d'accumuler chaque note dans la variable intTotal de type **accumulateur**. Un accumulateur est une variable numérique destinée à totaliser des valeurs.

La fonction **Val** affecte la valeur 0 aux notes absentes et non numériques.

6. Ajoutez les instructions suivantes à la *fin* de la procédure cmdCalculer_Click et déclarez la variable sngMoy :

```
sngMoy = Round(intTotal / 5, 1)
lblMoy.Caption = FormatNumber(sngMoy, 1)
```

> La fonction **Round** arrondit le résultat à 1 décimale. La fonction FormatNumber affiche le résultat avec une décimale.
>
>
> Consultez la section Les fonctions mathématiques, page 289.

7. Testez le projet et calculez une moyenne avec des notes absentes, non numériques et numériques.

8. Après avoir calculé la moyenne, donnez le focus au bouton Effacer et affichez la cote selon la moyenne dans l'étiquette lblCote :

Cote	Valeur de sngMoy
A	90 et plus
B	80 à 89.9
C	70 à 79.9
D	60 à 69.9
E	> 0 et moins de 60
""	= 0

9. Ajoutez une procédure personnalisée pour effacer les résultats. Celle-ci doit effacer la moyenne et la cote.

10. Complétez la procédure cmdEffacer() et testez les changements :

    - Après avoir effacé les 5 notes, effacez les résultats (procédure personnalisée) ;
    - Donnez le focus à la zone de saisie de la *première* note.

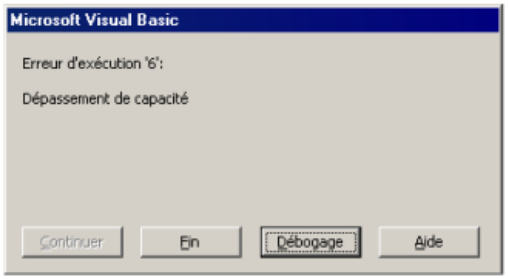
11. Lorsque l'utilisateur *change* une note, effacez les résultats, afin de ne pas afficher de résultats faux. Testez le changement.

12. Modifiez le calcul de la moyenne :

    - Actuellement, la moyenne est calculée sur 5 notes. Apportez les changements pour ne pas inclure dans le calcul, les notes absentes. Par exemple, si l'utilisateur entre 3 notes, il ne faut pas diviser par 5, mais plutôt par 3. Utilisez une variable **compteur** intNombre pour compter le nombre de notes, et ainsi diviser par ce nombre, au lieu de 5.

- Testez ce changement.

13. Effectuez un test en calculant une moyenne sans entrer *aucune note*. Le message suivant devrait apparaître :



- Cliquez sur le bouton Débogage pour localiser l'erreur dans le code de programmation.

> La cause de cette erreur est une **division par zéro** puisque la variable intNombre vaut 0. Notez que ce message de Visual Basic est *peu significatif*...

14. Ajoutez une condition supplémentaire pour corriger le bogue. N'affichez pas de message d'erreur, affichez 0,0 dans l'étiquette lblMoy. Testez le changement.

15. Validation et saisie des notes :

- N'acceptez que les touches 0 à 9, la touche BackSpace, la touche Escape et la touche Enter.
- Lorsque l'utilisateur presse la touche Escape, effacez la zone de texte de la note.
- Rendez la touche **Enter** accessible pour compléter l'entrée d'une note, en plus de la touche Tab. Ajoutez la condition suivante :

```
Case vbKeyReturn
  If Index = 4 Then
    cmdCalculer.SetFocus
  Else
    txtNote(Index + 1).SetFocus
  End If
```

> La zone de texte suivante reçoit le focus (Index +1), sauf s'il s'agit de la dernière note (Index = 4). Le bouton Calculer reçoit alors le focus.

- Exécutez le projet et utilisez la touche Enter pour complétez l'entrée des notes. Remarquez avec quelle rapidité l'utilisateur peut ainsi entrer les données et effectuer les calculs.
- Refusez toute note supérieure à 100. Codez cette validation dans une procédure événementielle **LostFocus**.

```
Private Sub txtNote_LostFocus(Index As Integer)
  Si la note est supérieure à 100, afficher le message qui
  suit et donner le focus à la note invalide.
  (Assurez-vous d'afficher le bon numéro de note ! Exemple : note 3.)
End Sub
```

L'événement **LostFocus** est déclenché lorsqu'un objet perd le focus. Dans le cas présent, l'événement est déclenché lorsque l'utilisateur presse la touche Enter, la touche Tab ou qu'il utilise la souris pour pointer un autre objet.

Consultez la section Les événements *GotFocus* et *LostFocus*, page 202.

Dans l'exemple présent, l'événement **Validate** aurait pu être utilisé à la place de LostFocus, le résultat aurait été le même. L'événement Validate survient juste avant l'événement LostFocus. Il empêche un contrôle de perdre le focus selon certains critères. Dans le cas présent, le critère aurait été tant que la note est > 100.

Consultez la section L'événement *Validate*, page 212.

- Testez la validation des notes supérieures à 100.

16. Lorsque l'utilisateur ferme l'application, affichez la forme frmMoyenne.

Étape 3 Coder les procédures de la forme frmMoyenne

1. Dans la procédure cmdOk_Click, déchargez de la mémoire la feuille frmMoyenne.
2. Codez la procédure Form_Load. Elle doit effectuer les tâches suivantes :

   - Afficher le *nombre de calculs effectués* dans lblNombre ;

   - Calculer la moyenne générale ;

- Afficher la moyenne générale avec une *seule décimale*.

- Pour afficher le *nombre de calculs effectués*, il faut d'abord calculer ce résultat dans la forme frmNotes. Codez ce calcul en tenant compte des éléments suivants :

  - Le calcul requiert une variable entière *globale*, puisque le calcul est effectué dans une feuille et que le résultat est affiché dans une autre feuille.

  - Une variable *globale* ou *publique* doit être déclarée dans un *module standard*. Utilisez le nom ModuleStandard pour le nouveau module et le nom gintNombreCalculs pour la variable.

  - Lors du calcul de gintNombreCalculs, les calculs effectués *sans aucune note* ne doivent pas être accumulés.

  - Pour calculer la *moyenne générale*, il faut accumuler toutes les moyennes calculées dans une variable *globale* (gsngTotMoy). Pour ce calcul, les moyennes calculées *sans aucune note* ne doivent pas être accumulées.

  - Affichez la moyenne générale avec une seule décimale.

- Codez les instructions nécessaires pour effectuer les calculs et les afficher. Modifiez les modules frmNotes, frmMoyenne et ModuleStandard. Testez les calculs et l'affichage.

17. Effectuez ce dernier test :

    - Calculez une moyenne sans avoir entrer de notes ;

    - Fermez l'application ;

    - Si une erreur est générée, corrigez le bogue afin d'afficher une moyenne générale = 0.

3.12 EXERCICE 14

Partie 1 Modifier le projet *Cartes*

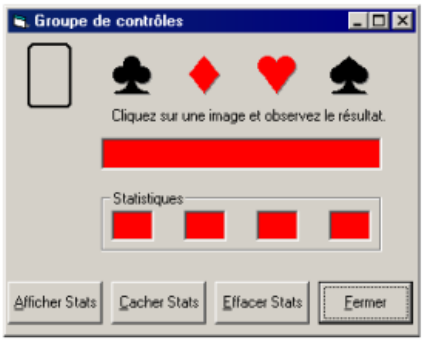
Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez votre dossier …/Visual Basic /Exercice 10 Cartes dans un nouveau dossier …/Visual Basic /Exercice 14 Cartes.

2. Copiez le fichier du volume Chapitre3 /Exercice14 /Debut dans votre dossier …/Visual Basic /Exercice 14 Cartes.

3. Exécutez le fichier .../Visual Basic /Exercice 14 Cartes /**Cartes.exe**.
4. Observez les détails suivants :
   - Observez le bouton "Cacher Stats" qui n'est pas accessible au démarrage.
   - Cliquez plusieurs fois sur les différentes couleurs de cartes et pressez le bouton "Afficher Stats". Les statistiques indiquent le nombre de fois qu'une couleur a été sélectionnée. Notez que le bouton "Cacher Stats" est maintenant accessible.
   - Pressez le bouton "Cacher Stats" ; le bouton "Afficher Stats" est redevenu accessible.
   - Cliquez plusieurs fois sur les couleurs et pressez le bouton "Afficher Stats".
   - Pressez le bouton "Effacer Stats" ; les statistiques sont mises à zéro et réaffichées.
5. À partir de Visual Basic, ouvrez le projet .../Visual Basic /Exercice 14 Cartes /Cartes.vbp.

Étape 2 Modifier l'interface graphique



1. Déplacez la forme rectangulaire, les 4 couleurs de cartes et les 2 étiquettes vers le haut de la feuille. Déplacez le bouton Fermer vers la droite.
2. Ajoutez 3 nouveaux boutons de commande : cmdAfficher, cmdCacher et cmdEffacer. Le bouton cmdCacher ne doit pas être accessible au démarrage. Le bouton cmdAfficher est le *bouton par défaut* et il doit recevoir le focus au démarrage.
3. Dessinez un cadre pour les statistiques et nommez-le **fraStats**. Celui-ci ne doit pas être visible au démarrage.

4. À l'intérieur du cadre, créez un groupe de contrôles lblFois(0) à lblFois(3). Les 4 étiquettes ont l'apparence d'une zone de texte, une taille de 10, le style gras, le contenu centré, et l'arrière-plan rouge.

5. Testez le déplacement de la touche Tab dans l'interface graphique.

Étape 3 Programmer les statistiques

1. Déclarez un tableau en mémoire de niveau module qui servira à compiler les statistiques :

```
Dim mintFois(3) As Integer
```

> Un tableau en mémoire sert à stocker des données. Comme une variable simple, il peut aussi être utilisé comme *compteur* ou *accumulateur*.

2. Ajoutez au programme l'instruction qui accumule les statistiques. À vous de dénicher le bon endroit...

```
mintFois(Index) = mintFois(Index) + 1
```

3. Codez et testez le bouton cmdCacher :

```
Private Sub cmdCacher_Click()
  Cacher le cadre des statistiques ;
  Rendre accessible le bouton Afficher ;
  Rendre inaccessible le bouton Cacher.
End Sub
```

4. Codez et testez le bouton cmdAfficher :

```
Private Sub cmdAfficher_Click()
  Le cadre des statistiques doit être visible ;
  Les 4 résultats doivent être affichés dans les étiquettes
  avec une boucle For/Next ;
  Le bouton Cacher doit être accessible ;
  Le bouton Afficher ne doit plus être accessible.
End Sub
```

5. Codez et testez le bouton cmdEffacer :

```
Private Sub cmdEffacer_Click()
  Les statistiques doivent être mises à zéro ;
  Faire appel à la procédure cmdAfficher_Click pour réafficher.
End Sub
```

6. Lorsque l'utilisateur clique sur une couleur de carte, cachez les statistiques afin de ne pas afficher des résultats qui ne sont pas à jour. Testez le changement et assurez-vous que l'accessibilité des boutons Afficher et Cacher est logique.

7. Enregistrez les composantes du projet.

Partie 2 Modifier le projet *Financement automobile*

Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez votre dossier .../Visual Basic /Exercice 9 Auto dans un nouveau dossier .../Visual Basic /Exercice 14 Auto.
2. À partir de Visual Basic, ouvrez le projet .../Visual Basic /Exercice 14 Auto /**Auto.vbp**.
3. Observez le code du projet et exécutez-le afin de vous remémorer ses divers éléments.
4. Afin de réduire le temps des tests, modifiez l'objet de démarrage pour frmAuto. Testez le changement.

Étape 2 Ajouter une procédure personnalisée avec des arguments

Un **argument** est une variable dont la valeur est transmise entre 2 procédures : celle qui fait appel et celle qui est appelée. Les arguments sont généralement cités entre parenthèses. Dans d'autres langages de programmation comme le langage C, un argument est appelé *paramètres*. On parle alors de *passage de paramètres* au lieu de **transmission des arguments**.

Consultez la section Transmission des arguments, page 239.

1. Ajoutez la procédure personnalisée suivante :

```
Private Sub Validation(CodeTouche As Integer, ZoneTexte As TextBox)
  Select Case CodeTouche
    Case vbKey0 To vbKey9
    Case vbKeyBack
    Case vbKeyEscape
      ZoneTexte = ""
    Case Else
      CodeTouche = 0
  End Select
End Sub
```

La procédure Validation sera utilisée pour valider 2 zones de texte, lors d'un événement KeyPress : txtCout et txtEchange.

Deux arguments sont transmis : la variable CodeTouche de type *Integer* et la variable ZoneTexte de type *Objet*.

2. Faites appel à la procédure Validation lors d'un événement KeyPress :

```
Private Sub txtCout_KeyPress(Keyascii As Integer)
  Call Validation(Keyascii, txtCout)
End Sub

Private Sub txtEchange_KeyPress(Keyascii As Integer)
  Call Validation(Keyascii, txtEchange)
End Sub
```

> La procédure Validation doit connaître quelle touche a été pressée (Keyascii) et quelle est la zone de texte à valider (txtCout ou txtEchange).
>
> La procédure Validation associe le premier argument de l'instruction Call (Keyascii) avec son premier argument (CodeTouche) et le deuxième argument de l'instruction Call (txtCout ou txtEchange) avec son deuxième argument (ZoneTexte).
>
> L'ordre de transmission des arguments doit être respecté et les arguments doivent être séparés par des virgules.

3. Testez les éléments suivants lors de la saisie du **coût** et de la **valeur d'échange** :
   - Entrez des lettres ;
   - Utilisez la touche Escape pour effacer la zone de saisie ;
   - Vérifiez si la touche BackSpace est fonctionnelle ;
   - Tapez une virgule comme symbole décimal (zut !).
4. Modifiez votre programme afin d'accepter la *virgule* et le *point décimal*. Ce dernier doit cependant être *converti en virgule*. De plus, n'acceptez qu'une *seule virgule* (réf. Exercice 10, Partie 1, Étape 3). Testez la modification dans les 2 zones de texte.
5. Déclarez 3 variables locales dans la procédure cmdPaiement_Click :

```
Dim sngCout As Single, sngEchange As Single
Dim sngPaiement As Single
```

> Suite à l'ajout de la procédure Validation, les validations par la fonction IsNumeric deviennent redondantes. Les prochaines opérations visent à épurer notre code de programmation. L'emploi de variables pour les calculs est une amélioration sensible pour la rapidité d'exécution.

6. Enlevez la fonction IsNumeric qui valide le coût. Vérifiez seulement si celui-ci est présent, et modifiez le message d'erreur : "Le coût est absent".
7. Affectez le contenu de la zone de texte à une variable :

```
sngCout = CSng(txtCout)
```

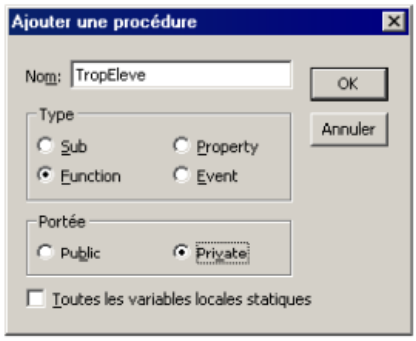
8. Répétez les 2 dernières opérations pour la *valeur d'échange*.
9. Dans le calcul du paiement, utilisez les variables sngCout, sngEchange et placez le résultat dans la variable sngPaiement.
10. Testez les derniers changements.

Étape 3 Ajouter une fonction personnalisée avec des arguments

En Visual Basic, la différence entre une procédure personnalisée et une **fonction personnalisée**, est que cette dernière *renvoie un résultat*.

Consultez la section Déclaration et appel d'une procédure *Function*, page 237.

1. Ajouter une fonction :
   - Sélectionnez Outils au menu et "Ajouter une procédure". Entrez le nom de la fonction, sélectionnez les choix *Function* et *Private*. Confirmez par OK.



   - Ajoutez *l'argument* Montant et les mots "As Boolean" dans la déclaration de la fonction. Insérez l'instruction If dans la fonction.

```
Private Function TropEleve(Montant As Single) As Boolean
  If Montant > 120000 Then
    TropEleve = True
  Else
    TropEleve = False
  End If
End Function
```

Cette fonction détermine si un montant dépasse la valeur 120,000 $. Le montant peut être le coût de l'auto ou la valeur d'échange. Si le montant excède 120,000 $, la fonction TropEleve *retournera la valeur True* et dans le cas contraire, *sa valeur sera False*.

2. Faire appel à une fonction :

   - Ajoutez les lignes en gras dans la procédure cmdPaiement_Click :

```
Private Sub cmdPaiement_Click()
  Dim sngCout As Single, sngEchange As Single
  Dim sngPaiement As Single
  Dim blnRetour As Boolean
  .
  .
  blnRetour = TropEleve(sngEchange)
  If TropEleve(sngCout) = True Or blnRetour Then
     MsgBox "L'une des valeurs dépasse 120,000$", , "Erreur"
     txtCout.SetFocus
     Exit Sub
  End If
  sngPaiement = -Pmt(lstTaux / 1200, _                    (Modifiez /12/100)
                 optNbVersement(mintNbVers).Caption, _
                 sngCout - sngEchange)

  lblPaiement = FormatCurrency(sngPaiement)
End Sub
```

La variable blnRetour sert à conserver la valeur de retour de la fonction TropEleve. Elle n'est pas obligatoire mais permet d'illustrer *2 façons de faire appel à une fonction* :

Première façon : `TropEleve(sngCout) = True`

Deuxième façon :
```
blnRetour = TropEleve(sngEchange)
If blnRetour
```

Dans le calcul du paiement, la division a été optimisée (/1200 au lieu de /12/100).

3. Afin de limiter les entrées de l'utilisateur, affectez la valeur 6 à la propriété MaxLength des zones de texte txtCout et txtEchange.

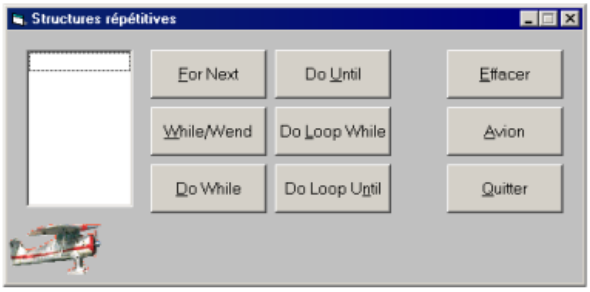
4. Testez la nouvelle fonction personnalisée TropEleve.

3.13 EXERCICE 15 LES DIFFÉRENTES STRUCTURES RÉPÉTITIVES

Étape 1 Préparation et observation

1. Copiez le dossier du volume Chapitre3/Exercice15/Boucles dans un nouveau dossier .../Visual Basic/Chapitre3/Boucles.

2. Observez l'interface graphique. En cliquant sur l'un des boutons représentant chacun une structure répétitive (boucle), les nombres Un à Dix seront affichés dans le contrôle *ListBox* situé à gauche de l'écran. Le bouton Effacer effacera le contenu de la liste et le bouton Avion déplacera l'avion dans la fenêtre.



Étape 2 Créer un tableau avec la fonction Array

1. Ajoutez le code de programmation suivant :

```
Dim mvntNombre As Variant

Private Sub Form_Load()
  mvntNombre = Array("Un", "Deux", "Trois", "Quatre", "Cinq", _
                     "Six", "Sept", "Huit", "Neuf", "Dix")
End Sub
```

La fonction ***Array*** renvoie un tableau dans la variable mvntNombre. Le tableau compte 10 éléments et les indices varient de 0 à 9.

Consultez la section Les fonctions *Array*, *LBound* et *UBound*, page 283.

2. Codez le bouton Quitter.

Étape 3 Coder les différentes boucles

Consultez la section 5.14 Les structures répétitives.

1. Codez le bouton cmdForNext. En utilisant une boucle **For/Next**, affichez dans le contrôle lstListe chacun des éléments du tableau mvntNombre (Un à Dix). Utilisez une variable locale entière I ainsi que la méthode AddItem. Testez le bouton.
2. Codez le bouton cmdEffacer qui doit effacer tous les nombres de la liste. Utilisez la méthode *Clear* de l'objet *ListBox*. Testez le bouton.
3. Le bouton cmdWhileWend doit aussi afficher les nombres Un à Dix. Entrez et complétez le code de programmation qui suit :

```
Private Sub cmdWhileWend_Click()
  Dim I As Integer

  'TANT QUE la condition est vraie, les instructions incluses
  'dans la boucle sont exécutées. La condition est vérifiée AVANT
  'l'exécution des instructions.

  While I < 10
    ajouter mvntNombre(I) dans la liste
        augmenter I de 1
  Wend
End Sub
```

La boucle **While/Wend** répète un ensemble d'instructions (comprises entre les mots *While* et *End*), TANT QUE la condition est vraie. Donc dans l'exemple qui précède, les instructions sont répétées tant que I < 10. Aussitôt que I = 10, le traitement est terminé.

La condition est vérifiée AVANT l'exécution des instructions.

Si on compare cette boucle à la boucle For/Next, dans une boucle While/Wend, il est nécessaire d'incrémenter la variable I.

4. Testez le bouton cmdWhileWend qui devrait produire le même résultat que le bouton cmdForNext.

Pour bien comprendre le fonctionnement d'une boucle, il serait judicieux d'exécuter les instructions en mode *Pas à pas* (F8).

5. Codez et testez le bouton cmdDoWhile. Entrez et complétez le code de programmation qui suit :

```
Private Sub cmdDoWhile_Click()
  Dim I As Integer

  'TANT QUE la condition est vraie, les instructions incluses
  'dans la boucle sont exécutées. La condition est vérifiée AVANT
  'l'exécution des instructions.

  Do While I < 10
    .
    .
  Loop
End Sub
```

> Quelle est la différence entre **While/Wend** et **Do While** ?
>
> Dans les 2 exemples qui précèdent, aucune. Par contre, toute boucle commençant par le mot **Do** offre l'option **Exit Do** qui permet une sortie éventuelle de la boucle.

6. Codez une sortie de boucle. Lorsque le mot Cinq a été ajouté dans la liste, effectuez une sortie de boucle :

```
If mvntNombre(I) = "Cinq" Then
  Exit Do
End If
```

7. Testez la sortie de boucle et par la suite, mettez l'instruction If en commentaire.

8. Codez et testez le bouton cmdDoUntil. Entrez et complétez le code de programmation qui suit :

```
Private Sub cmdDoUntil_Click()
  Dim I As Integer

  'les instructions incluses dans la boucle sont exécutées
  'JUSQU'À ce que la condition soit vraie. La condition
  'est vérifiée AVANT la première exécution.

  Do Until I = 10
    .
    .
  Loop
End Sub
```

> Quelle est la différence entre **Do While** et **Do Until** ?
>
> Une boucle **Do While** exécute les instructions TANT QUE la condition est vraie tandis que la boucle **Do Until** exécute les instructions JUSQU'À ce que la condition soit vraie. Il faut alors vérifier la condition différemment (exemple : I <10 avec Do While et I = 10 avec Do Until).

9. Codez et testez le bouton cmdDoLoopWhile. Entrez et complétez le code de programmation qui suit :

```
Private Sub cmdDoLoopWhile_Click()
  Dim I As Integer

  'TANT QUE la condition est vraie, les instructions incluses
  'dans la boucle sont exécutées. La condition est vérifiée APRÈS
  'une première exécution.

  Do
    .
    .
  Loop While I < 10
End Sub
```

> Une boucle **Do Loop** (condition spécifiée à la ***fin*** de la boucle) vérifie la condition APRÈS une première exécution du bloc d'instructions. On utilise ce genre de boucles lorsqu'il est nécessaire d'avoir effectuer au moins un traitement avant de vérifier la condition.

10. Codez et testez le bouton cmdDoLoopUntil. Entrez et complétez le code de programmation qui suit :

```
Private Sub cmdDoLoopUntil_Click()
  Dim I As Integer

  'Exécute les instructions incluses dans la boucle
  'JUSQU'À ce que la condition soit vraie.  La condition
  'est vérifiée APRÈS la première exécution.

  Do
    .
    .
  Loop Until I = 10
End Sub
```

11. Déplacez l'avion vers la droite avec une boucle **For/Next** et testez le bouton cmdAvion :

```
Private Sub cmdAvion_Click()
  Dim I As Integer
  For I = 0 To 6000 Step 2
    imgAvion.Left = I
  Next I
End Sub
```

> Pour déplacer un objet vers la droite, il suffit d'augmenter graduellement sa propriété Left. Ainsi, pour déplacer un objet vers la gauche, la valeur de sa propriété Left devra être réduite.

Étape 4 Jongler avec les boucles

1. Modifiez le bouton **cmdForNext** afin qu'il affiche les nombres en *ordre décroissant* (Dix à Un) dans la liste. Testez le changement.

> La valeur d'incrémentation dans une boucle For/Next peut être positive ou négative.

2. Modifiez le bouton cmdWhileWend afin qu'il affiche seulement les nombres *impairs*. Testez le changement.
3. Modifiez le bouton cmdDoWhile afin qu'il affiche seulement les nombres *pairs*. Testez le changement.
4. Modifiez le bouton cmdDoUntil afin qu'il affiche seulement les nombres *impairs en ordre décroissant*. Testez le changement.
5. Modifiez le bouton cmdDoLoopWhile afin qu'il affiche seulement les nombres *pairs en ordre décroissant*. Testez le changement.
6. Modifiez le bouton cmdDoLoopUntil afin qu'il affiche seulement les nombres Trois, Six et Neuf. Testez le changement.
7. Déplacez l'avion à *reculons*. Avant d'exécuter, déplacez l'avion à l'extrémité droite de l'interface graphique. Testez le changement.
8. Dessinez un nouveau bouton cmd3Fois à un endroit quelconque de la feuille. Il doit contenir le titre "Afficher 3 fois". Ce bouton doit afficher *trois fois, chacun des nombres* du tableau. Utilisez une boucle **For Next** imbriquée. Testez le bouton.

Exemple Un, Un, Un, Deux, Deux, Deux, Trois, Trois, Trois, ..., Dix, Dix, Dix.

9. Dessinez un autre bouton cmdCasseTete à un endroit quelconque de la feuille. Il doit contenir le titre Casse-tête. Ce bouton doit afficher chacun des nombres du tableau, *un nombre de fois égal à sa valeur*.

Exemple Un, Deux, Deux, Trois, Trois, Trois, Quatre, Quatre, Quatre, Quatre, etc.

3.14 EXERCICE 16

Partie 1 Compléter le projet *Course automobile*

Étape 1 Préparation et observation

1. Copiez le dossier du volume Chapitre3 /Exercice16 /Debut Course dans un nouveau dossier .../Visual Basic/Exercice 16 Course.
2. Exécutez le fichier .../Visual Basic /Exercice 16 Course /**Course.exe**. Pressez le bouton GO à plusieurs reprises et observez le résultat. Notez la forme du pointeur de la souris.
3. Ouvrez le projet .../Visual Basic /Exercice 16 Course /**Course.vbp**. Observez l'interface graphique. Vous y trouverez les contrôles imgAuto1, imgAuto2, cmdGo, lblResultat et tmrTemps (contrôle Timer).
4. Observez la valeur de la propriété Top des 2 autos.

> Lorsqu'une auto est placée à la position de départ, sa propriété Top est fixée à 6 600. Pour déplacer une auto vers le haut de la feuille, il faut *réduire graduellement la valeur de la propriété Top*, jusqu'à ce qu'elle atteigne 0 (haut de la feuille).

Étape 2 Personnaliser le pointeur de la souris

1. Affectez la valeur *99 – Custom* à la propriété MousePointer de la feuille.
2. Sélectionnez la propriété MouseIcon et retrouvez le fichier Icons \Traffic \trffc16.ico.

> La valeur **99 – Custom** de la propriété **MousePointer** permet de personnaliser le pointeur de souris lorsqu'il est placé sur un objet. La propriété **MouseIcon** spécifie l'image utilisée.
>
> Consultez la section Les propriétés *MousePointer* et *MouseIcon*, page 194.

Étape 3 Générer aléatoirement des nombres

1. Codez la procédure du bouton GO. Entrez les lignes de code ou convertissez le pseudo-code en Visual Basic.

```
Private Sub cmdGo_Click()
  Dim intNb1 As Integer, intNb2 As Integer
  Dim I As Integer
  Randomize
```

L'instruction **Randomize** sert à initialiser le générateur de nombres aléatoires à partir de l'heure système. Celui-ci est utilisé pour générer automatiquement des nombres dont la valeur sera utilisée pour réduire aléatoirement la propriété Top des autos. La fonction **Rnd** (Random) permet de générer un nombre aléatoire.

Consultez la section Autres Fonctions, *Rnd* (*Randomize*), page 304.

- `Cacher l'étiquette lblResultat.`
- `Si l'une des autos n'est pas en position de départ, placer les 2 autos en position de départ et sortir de la procédure par l'instruction Exit Sub.`

Il faut prévoir le cas où une course a eu lieu et que l'utilisateur clique sur le bouton GO. Les autos sont alors replacées en position de départ et l'utilisateur doit cliquer de nouveau sur le bouton GO pour démarrer une autre course.

```
  Do Until imgAuto1.Top <= 0 Or imgAuto2.Top <= 0
    intNb1 = Int(11 * Rnd)
    intNb2 = Int(11 * Rnd)
    imgAuto1.Top = imgAuto1.Top - intNb1
    imgAuto2.Top = imgAuto2.Top - intNb2
  Loop
```

Deux nombres aléatoires **entiers** intNb1 et intNb2 compris **entre 0 et 10** sont générés jusqu'à ce qu'une auto atteigne le haut de la feuille (Top <= 0). Ces nombres réduisent la valeur de la propriété Top de chacune des autos.

Formule pour obtenir un nombre aléatoire dans une plage donnée :

Int((limite supérieure - limite inférieure + 1) * **Rnd** + limite inférieure)

Nombre aléatoire = Int((10 - 0 + 1) * Rnd + 0)

Nombre aléatoire = Int(11 * Rnd)

La fonction **Int** (Integer) ne renvoie que la partie entière du résultat.

```
    • Si les deux autos atteignent le haut de la feuille en même temps,
      afficher "Match nul!" dans l'étiquette lblResultat. Sinon, afficher
      "L'auto 1 a gagné!" ou "L'auto 2 a gagné!" selon le cas.
    • Afficher l'étiquette lblResultat.
End Sub
```

2. Codez les instructions pour le clignotement de l'étiquette lblResultat :

```
Private Sub tmrTemps_Timer()
  With lblResultat
    If .Visible = True Then
      If .BackColor = vbRed Then
        .BackColor = &H800080
        .ForeColor = vbRed
      Else
        .BackColor = vbRed
        .ForeColor = &H800080
      End If
    End If
  End With
End Sub
```

> La propriété Interval du contrôle Timer a été fixée à 500. Cette procédure est donc exécutée à chaque *demi-seconde*. Les couleurs de l'arrière-plan et du texte sont interchangées.

3. Exécutez le projet et assurez-vous de son bon fonctionnement.

***Étape 4 Générer aléatoirement des couleurs avec la fonction* RGB**

1. Mettez en commentaire toutes les lignes de code de la structure With dans la procédure tmrTemps_Timer.

2. Entrez les 2 instructions suivantes :

```
lblResultat.BackColor = RGB(Int(256 * Rnd), Int(256 * Rnd), _
                                    Int(256 * Rnd))
lblResultat.ForeColor = RGB(Int(256 * Rnd), Int(256 * Rnd), _
                                    Int(256 * Rnd))
```

> La fonction **RGB** (Red, Green, Bleue) produit une couleur à partir de 3 codes de couleur compris entre 0 et 255. La fonction Rnd génère le code de couleur (entier de 0 à 255). Le code 0 est le noir et le code 255 est le blanc.
>
>
> Consultez la section Autres fonctions, RGB, page 306.

3. Exécutez le projet et observez l'étiquette lblRésultat à la fin d'une course.

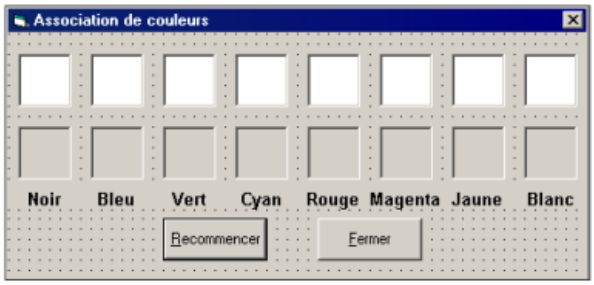
4. Enregistrez les composantes du projet.

Partie 2 Créer le projet *Association de couleurs*

Étape 1 Préparation, observation et enregistrement

1. Copiez le dossier du volume Chapitre3 /Exercice16 /Debut Couleurs dans un nouveau dossier .../Visual Basic /Exercice 16 Couleurs.
2. Exécutez le fichier .../Visual Basic /Exercice 16 Course /**Association.exe**. Associez les 8 couleurs. Testez une mauvaise association et le bouton Recommencer.
3. À partir de Visual Basic, créez un nouveau projet. Nommez le projet **Association** et la feuille **frmAssocie**.
4. Enregistrez la feuille et le projet dans le dossier .../Visual Basic /Exercice 9 Auto /(**Association.frm** et **Association.vbp**).

Étape 2 Conception de l'interface graphique du projet



1. Créez un premier groupe de contrôles txtCouleur pour les 8 couleurs. Placez les zones de texte de la gauche (Index = 0) vers la droite (Index = 7). Fixez la valeur 1 – Automatic à la propriété **DragMode** de chaque zone de texte.

La propriété **DragMode = 1 – Automatic** d'un objet permet le déplacement de celui-ci.
Consultez la section Les événements DragDrop et DragOver (propriété DragMode), page 204.

2. Créez un deuxième groupe de contrôles lblNom pour recevoir les couleurs déplacées. Fixez la propriété BorderStyle = 1 – Fixed Single.
3. Créez 8 étiquettes pour le nom des couleurs, dans *le même ordre* que celui illustré dans la figure de l'interface graphique.

Étape 3 Coder les procédures

1. Codez le bouton Fermer qui procède au déchargement de la feuille.
2. Déclarez 2 variables de portée module et codez l'événement Form_Load :

```
Dim I As Integer
Dim mintExact As Integer    'Nombre d'associations exactes

Private Sub Form_Load()
  Call ChargerCouleurs
End Sub
```

3. Ajoutez la procédure personnalisée ChargerCouleurs :

```
Private Sub ChargerCouleurs()
  txtCouleur(0).BackColor = QBColor(0)        'Noir
  For I = 9 To 15                             'couleurs 9 à 15
    txtCouleur(I - 8).BackColor = QBColor(I)
  Next I
End Sub
```

> La fonction **QBColor** retourne 16 couleurs (0 à 15). Le nombre 0 représente le noir et les nombres 9 à 15 correspondent aux autres couleurs (bleu, vert, cyan, rouge, magenta, jaune, blanc). Notez que les couleurs sont dans *le même ordre* que les étiquettes des noms de couleurs.
>
>
> Consultez la section Autres fonctions, QBColor, page 305.

4. Exécutez le projet et observez l'ordre des couleurs. Cet ordre sera modifié un peu plus loin dans l'exercice.
5. Ajoutez la procédure suivante :

```
Private Sub lblNom_DragDrop(Index As Integer, Source As Control, _
                                X As Single, Y As Single)
  If Index = Source.Index Then
    lblNom(Index).BackColor = Source.BackColor
    Source.Visible = False
    mintExact = mintExact + 1
  End If
  If mintExact = 8 Then
    MsgBox "Association terminée!", , ""
    Call cmdRecommencer_Click
    cmdFermer.SetFocus
  End If
End Sub
```

L'événement **DragDrop** se produit lorsqu'une opération Glisser/déplacer prend fin, au moment où l'utilisateur relâche le bouton de la souris. L'argument **Source** représente le contrôle en déplacement (txtCouleur).

Si les index des 2 contrôles sont égaux, alors l'association est exacte et la couleur sera affectée à l'étiquette lblNom correspondante. La zone de texte est cachée puisque la couleur a été affectée à l'étiquette.

Quand les 8 associations sont exactes, un message est affiché et la procédure du bouton Recommencer est exécutée.

6. Ajoutez la procédure suivante :

```
Private Sub cmdRecommencer_Click()
  mintExact = 0
  Call ChargerCouleurs
  For I = 0 To 7
    lblNom(I).BackColor = vbInactiveCaptionText
    txtCouleur(I).Visible = True
  Next I
End Sub
```

Le bouton Recommencer recharge les couleurs et l'arrière-plan des étiquettes redevient gris. Les zones de texte des 8 couleurs sont de nouveau visibles.

7. Testez le projet. Associez les 8 couleurs et observez le résultat. Par la suite, associez de mauvaises couleurs.
8. En mode création, déplacez les 8 zones de texte des couleurs pour qu'une couleur ne soit pas au-dessus de son nom. Testez ce changement.
9. Enregistrez les composantes du projet.

Partie 3 Modifier le projet *Exercices de calculs*

Étape 1 Préparation et observation

1. Copiez le dossier du volume Chapitre3 /Exercice16 /Debut Calculs dans un nouveau dossier .../Visual Basic /Exercice 16 Calculs.
2. Exécutez le fichier .../Visual Basic /Exercice 16 Calculs /**Calculs.exe**. Ce projet est destiné à des enfants pour les aider à maîtriser les mathématiques. Effectuez de bons et de mauvais calculs. Cliquez sur le bouton OK sans avoir donné une réponse. Testez toutes les options.

3. À partir de Visual Basic, ouvrez le projet .../Visual Basic /Exercice 16 Calculs /**Calculs.vbp**.

4. Observez ces éléments dans l'interface graphique :

   - La valeur de la propriété **Appearance** des boutons d'option et de la case à cocher a été fixée à **0-Flat**, ce qui modifie leur apparence ;
   - 3 groupes de contrôles ont été créés imgSigne, optNiveau et optOperation ;
   - 8 images ont été utilisées dont 4 pour les signes, 2 figures (souriante et triste), imgOperation et imgResultat dont aucune image n'a été fixée ; celles-ci seront affectées par programmation ;
   - Le cadre fraReponses n'est pas visible au démarrage.

Étape 2 Coder les procédures

1. Dans le module de code, fixez les options par défaut : Addition et Niveau 1.

2. Déclarez 2 variables de portée module pour les nombres des opérations :

```
Dim mintNb1 As Integer
Dim mintNb2 As Integer
```

3. Ajoutez une procédure personnalisée GenererNombres qui doit générer aléatoirement 2 nombres entiers qui seront ensuite affichés dans les étiquettes lblNombre1 et lblNombre2. Tenez compte des consignes suivantes :

   - Les nombres pour le *niveau 1* des additions, soustractions et multiplications doivent se situer dans la plage 1 à 10.
   - Les nombres pour le *niveau 2* des additions, soustractions et multiplications doivent se situer dans la plage 1 à 50.
   - Le *nombre 2* pour le *niveau 1* de la division doit être dans la plage 1 à 10 et le *nombre 2* pour le *niveau 2* de la division doit être dans la plage 10 à 20. Le *nombre 1* de la division est un *multiple de 1 à 10* du nombre 2, afin que les réponses soient toujours *entières*. Générez le nombre 2 *avant* le nombre 1.
   - Pour les soustractions, assurez-vous que le nombre 1 *est plus grand* que le nombre 2 avant de ne pas obtenir un résultat *négatif*. Dans ce cas, utilisez le truc suivant qui consiste à *permuter* le contenu de 2 variables à l'aide d'une troisième variable intTemp :

```
intTemp = mintNb1
mintNb1 = mintNb2
mintNb2 = intTemp
```

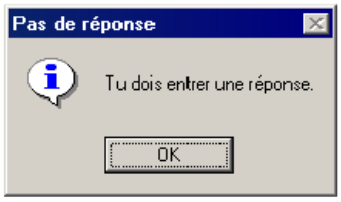
- Affichez les nombres dans les étiquettes et effacez la zone de la réponse.

Pour vous aider à coder cette procédure, consultez la section Autres Fonctions, *Rnd* (*Randomize*) page 304 et la Partie 1 de ce même exercice.

4. Codez le bouton "Au Revoir" qui procède au déchargement de la feuille.
5. Dans la zone txtReponse, n'acceptez que les chiffres 0 à 9 et la touche BackSpace.
6. Limitez le nombre de chiffres saisis à 4, afin d'éviter une erreur d'exécution de type Dépassement de capacité, puisque les variables utilisées sont de type Integer (valeur maximum : 32,767).

Modifiez la valeur de l'une des propriétés de txtReponse.

7. Lors d'un clic sur l'un des boutons d'option des opérations, générez de nouveaux nombres et affectez l'image du signe correspondant à la propriété Picture du contrôle imgOperation.
8. Lors d'un changement de niveau, générez de nouveaux nombres.
9. Codez la procédure cmdOk_Click en tenant compte de ces consignes :
   - Assurez-vous que l'enfant a bien tapé une réponse. En l'absence de réponse, affichez ce message et donnez le focus à la zone de la réponse :



   - Calculez la bonne réponse (intReponse) pour chacune des 4 opérations.
   - Si la réponse entrée est *exacte*, affichez l'image *souriante*, effacez txtReponse et générez de nouveaux nombres.
   - Si la réponse entrée est *inexacte*, affichez l'image *triste* et sélectionnez tout le texte de la zone txtReponse (réf. Exercice 12). L'enfant pourra ainsi visualiser son erreur et le contenu de la zone sera effacé automatiquement, aussitôt qu'il inscrira une nouvelle réponse.

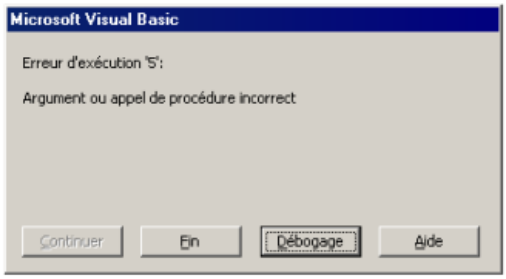
10. Effectuez plusieurs tests du projet : tous les choix d'opérations, tous les niveaux, de bonnes et de mauvaises réponses ainsi que des réponses absentes.
11. Lorsque l'enfant clique sur le bouton Passe, générez de nouveaux nombres.
12. Calculez le nombre de bonnes et de mauvaises réponses (lblBons et lblMauvais).
13. Lorsque la case à cocher Voir les résultats est cochée, affichez le cadre fraReponses. Cependant, si elle n'est pas cochée, le cadre ne doit pas être visible.

Une seule instruction est nécessaire.

14. Lorsque l'enfant clique sur le bouton "Effacer les résultats", la valeur 0 doit être affichée pour les 2 résultats.
15. Testez le bouton Passe, l'affichage et l'effacement des résultats.
16. Lorsque l'enfant effectue les choix suivants, donnez le focus à la zone de la réponse :
    - Changement d'opération ;
    - Changement de niveau ;
    - Voir les résultats ;
    - Effacer les résultats ;
    - Boutons OK et Passe.

Évitez la répétition d'instructions.

17. Testez le dernier changement. L'erreur suivante devrait se produire :



Cette erreur est causée par l'utilisation de la méthode SetFocus à un moment où elle n'est pas disponible, c'est-à-dire lors de l'événement Form_Load. À ce moment, les *contrôles ne sont pas encore dessinés* sur la feuille. Il est donc *impossible que l'un d'eux reçoive le focus*.

18. Pour bien comprendre l'erreur, exécutez le projet en mode "Pas à pas" avec la touche F8 et observez attentivement la séquence des instructions exécutées.

Lorsque la propriété Value d'un bouton d'option est modifiée, l'événement Click de celui-ci est alors déclenché, provoquant l'exécution automatique de la procédure correspondante. Plus précisément, le résultat est le même que si l'utilisateur avait cliqué sur le bouton d'option.

Pour corriger l'erreur, utilisez l'événement **Form_Activate** au lieu de Form_Load. L'événement **Activate** se produit *après* que les contrôles aient été dessinés. La méthode Setfocus est alors disponible.

Consultez la section Les événements d'une feuille, page 160.

19. Corrigez l'erreur et testez le changement. Testez tous les choix que l'enfant peut sélectionner et assurez-vous que la zone de la réponse reçoit le focus après chacun de ses choix.

Partie 4 Observer le projet *Exercices de calculs* avec sons et animations

1. Ouvrez le projet **Calculs.vbp** stocké dans le dossier du volume Chapitre3 /Exercice16 /Sons Animations. Ce dossier comporte deux sous-dossiers, l'un pour les images et l'autre pour les sons utilisés dans le projet.

Ce projet est une réalisation de Sébastien Saucier Fortier, un étudiant en informatique au Collège de Sherbrooke.

2. Exécutez le projet et observez les sons et les animations qu'on y retrouve.

3. En mode création, observez la présence d'u nouveau contrôle MMControl dans la boîte à outils. Ce contrôle a été utilisé sur les deux feuilles du projet.

Le contrôle **MCI (Media Control Interface)** multimédia permet de lire des fichiers son et vidéo. Il apparaît comme suit sur une interface graphique :

Dans le cas présent, il n'est pas visible lors de l'exécution.

4. Notez que le contrôle PictureBox a été utilisé pour afficher certaines images.

Le contrôle **PictureBox** permet d'afficher un fichier image chargé par la fonction **LoadPicture**.

Consultez la section Les contrôles *Image* et *PictureBox* page 178 et la section Autres fonctions, *LoadPicture*, page 303.

5. Observez attentivement le code de programmation afin de découvrir comment :
   - Faire jouer un son ;
   - Charger un fichier image par la fonction LoadPicture ;
   - Programmer une animation.

3.15 EXERCICE 17 COMPLÉTER LE PROJET REGROUPEMENT DE PROJETS

Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre3 /Exercice17 /Debut dans un nouveau dossier .../Visual Basic /Exercice 17 Regroupement.
2. Exécutez le fichier .../Visual Basic /Exercice 17 Regroupement /**Regroupement.exe**. Ce projet consiste à regrouper quelques projets en un seul.
3. Observez l'écran d'accueil et cliquez dans la fenêtre pour continuer. Testez les boutons de commande et les choix du menu général. Cliquez, *entre* les boutons de commande des choix de projets, avec le bouton *droit* de la souris. Testez les différents choix du menu contextuel affiché.
4. À partir de Visual Basic, ouvrez le projet .../Visual Basic /Exercice 17 Regroupement /**Regroupement.vbp**.

5. Observez ces éléments dans la feuille frmPrincipale :
   - Le groupe de contrôles des boutons de commande (cmdProjet) ;
   - Le groupe de contrôles des choix du menu Projets (mnuProjet) ;
   - Le bouton Quitter et le choix Quitter au menu Fichier.

Étape 2 Réaliser le projet

1. Créez un *menu contextuel* pour la feuille comprenant les 4 choix du menu Projets. Ajoutez la procédure qui suit. (Ne pas taper la première ligne !)

```
Private Sub Form_MouseUp(Button As Integer, Shift As Integer, _
                                    X As Single, Y As Single)
  If Button = vbRightButton Then
    PopupMenu mnuProjets, vbPopupMenuRightButton + _
                                    vbPopupMenuCenterAlign
  End If
End Sub
```

> L'événement **MouseUp** se produit lorsque l'utilisateur a cliqué sur l'objet Form, au moment où il relâche le bouton de la souris.
>
> L'argument **Button** permet de vérifier quel bouton de la souris a été enfoncé.
>
> La méthode **PopupMenu** de l'objet Form affiche un menu contextuel déjà construit par le Créateur de menus. Dans le cas présent, il s'agit du menu mnuProjets qui sera centré sur le pointeur de la souris.
>
>
> Consultez la section Les menus contextuels (méthode PopupMenu), page 219.

2. Exécutez le projet et cliquez, avec le bouton droit de la souris, sur un endroit en *bleu* de la feuille. Observez le menu contextuel affiché.

3. Insérez les lignes de code qui suivent dans la section Général/Déclarations du module de code.

```
Private Enum Projet
  QuestionPiege       'valeur = 0
  CourseAutomobile    'valeur = 1
  AssociationCouleurs 'valeur = 2
  ExercicesCalculs    'valeur = 3
End Enum
```

L'instruction **Enum** est utilisée pour déclarer une **énumération de constantes**. Ce procédé améliore la compréhension du code de programmation. Par exemple, le mot CourseAutomobile sera employé au lieu du nombre 1, qui est *peu significatif*.

Les valeurs affichées en commentaire sont les valeurs par défaut affectées aux constantes.

Consultez la section L'énumération de constantes (instruction *Enum*), page 250.

4. Déclarez une variable de portée module qui sera utilisée pour conserver l'index du projet choisi par l'utilisateur.

```
Dim mintProjet As Projet
```

Le mot Projet cité à la suite du mot Enum, n'est pas utilisable dans les instructions, d'où la nécessité de déclarer la variable mintProjet.

5. Ajoutez une procédure personnalisée AfficherProjet :

```
Private Sub AfficherProjet()
  Me.Hide
  Select Case mintProjet
    Case Questionpiege
      frmQuestion.Show vbModal
    Case CourseAutomobile
      frmCourse.Show vbModal
    Case AssociationCouleurs
      frmAssocie.Show vbModal
    Case ExercicesCalculs
      frmCalculs.Show vbModal
  End Select
  Me.Show
End Sub
```

La procédure AfficherProjet affiche la feuille correspondante au projet sélectionné. Lorsqu'une autre feuille est affichée, la feuille principale est cachée par la méthode **Hide**. Notez l'utilisation des constantes pour plus de clarté.

6. Lorsque l'utilisateur clique sur l'un des 4 projets, la variable mintProjet doit être mise à jour par l'instruction suivante :

```
mintProjet = Index
```

> Lors de la saisie de l'instruction précédente, au moment où le caractère = est tapé, l'éditeur de Visual Basic affiche la *liste des constantes énumérées*.

7. Après avoir mis à jour la variable mintProjet, faites appel à la procédure AfficherProjet.
8. Ajoutez 4 feuilles existantes au projet :

Fichier de la feuille	Dossier
Question.frm	Visual Basic/Exercice1 Question
Course.frm	Visual Basic/Exercice16 Course
Association.frm	Visual Basic/Exercice16 Couleurs
Calculs.frm	Visual Basic/Exercice16 Calculs

9. **Enregistrez chacune des feuilles** ajoutées dans le dossier du projet .../Visual Basic /Exercice 17 Regroupement. *Soyez vigilant* lors de l'enregistrement, car le dossier *assumé par défaut est celui d'où la feuille a été ajoutée.*
10. Exécutez le projet et testez chacun des choix de projets.
11. Codez et testez le bouton Quitter ainsi que le choix Quitter au menu.
12. Ajoutez la procédure suivante :

```
Private Sub mnuProjet_Click(Index As Integer)
  Call cmdProjet_Click(Index)
End Sub
```

> Les index des choix au menu mnuProjet correspondent aux index des boutons de commande cmdProjet. Il est ainsi possible par une *seule instruction* de synchroniser les boutons de commande, les choix du menu Projets et les choix du menu contextuel.

13. Exécutez le projet et testez les choix du menu Projets et les choix du menu contextuel.
14. Utilisez un *modèle de feuille* pour une fenêtre d'accueil :
    - Sélectionnez au menu Projet /Ajouter une feuille /onglet Nouveau / Écran d'accueil /Ouvrir.
    - Modifiez le nom de la feuille pour frmIntro et enregistrez-la sous le nom Intro.frm dans le dossier du projet.

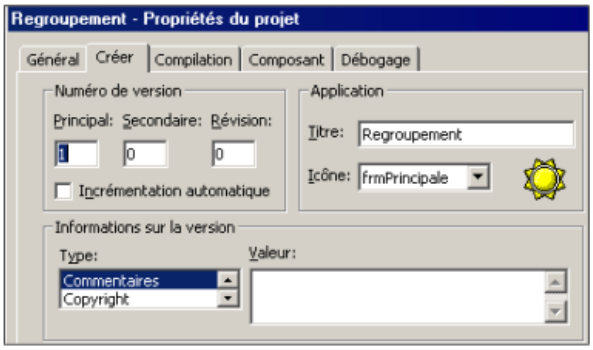
- Modifiez *l'objet de démarrage* pour frmIntro.
- Exécutez et observez le résultat. Le *titre* du projet Regroupement a été inscrit ainsi que la *version* du projet 1.0.0. Cliquez dans la fenêtre pour la fermer.
- Observez et modifiez le code de programmation de la feuille frmIntro :

  - Enlevez la procédure Form_KeyPress.

  - Dans la procédure Frame1_Click, affichez la feuille frmPrincipale par la méthode Show (avant Unload Me).

  - Observez la procédure Form_Load.

La procédure Form_Load utilise les propriétés du projet : App.Major, App.Minor, App.Revision et App.Title. L'objet **App** (Application) est un objet global qui représente le projet. Ses propriétés sont illustrées dans la boîte de dialogue "Propriétés du projet" accessible par Projet/Propriétés du projet.



Dans l'onglet Créer, on retrouve les informations qui ont été utilisées dans la procédure Form_Load.

Consultez la section 4.13 L'objet *App*.

- Dans la boîte de dialogue Propriétés du projet, modifiez le titre de l'application pour Mes projets préférés. Indiquez une version fictive 3.4.2.
- Modifiez l'interface graphique afin d'obtenir un résultat semblable à celui illustré à la page suivante.

- Exécutez le projet, observez la feuille d'accueil et cliquez dans la fenêtre pour continuer.

4 Les objets, leurs propriétés, méthodes et événements

4.1 L'OBJET FORM

Un objet **Form** est une instance de la classe d'objet Form. Il représente une feuille sur laquelle d'autres objets sont ajoutés (contrôles). L'ensemble des feuilles ou fenêtres d'un projet constitue *l'interface graphique utilisateur* de l'application. Chaque objet Form est un élément de la **collection Forms**.

Collection

Une **collection** est formée d'un *ensemble d'objets indexés*, l'index maximum étant le nombre d'éléments dans la collection – 1. (Exemple : Form(0) à Form(4), 5 feuilles dans la collection.)

L'ensemble des contrôles d'un objet Form forme la **collection Controls** dans laquelle il est possible de circuler pour effectuer certaines tâches globales.

Exemple 1 Masquer tous les contrôles de la feuille frmTest lors d'un clic sur la feuille.

```
Private Sub Form_click()
  Dim Element As Control
  For Each Element In frmTest.Controls
    Element.Visible = False
  Next
End Sub
```

Exemple 2 Attribuer la couleur rouge au texte de tous les contrôles Label de la feuille frmFeuille.

```
Dim Controle As Control
For Each Controle In frmFeuille.Controls
  If TypeOf Controle Is Label Then
    Controle.ForeColor = vbRed
  End If
Next Controle
```

La boucle For Each est décrite dans la section La boucle *For Each*, page 272.

Menu système

Une feuille contient une barre de titre dans laquelle on retrouve une icône, un titre et un **menu système**, formé des boutons **Réduire**, **Agrandir** et **Fermer**.

Comme tous les objets, l'objet Form possèdent des propriétés, méthodes et événements qui lui sont propres.

Les principales propriétés d'une feuille

(Name)	Le préfixe **frm** identifie un objet Form. Dans le code de programmation, le mot clé **Me** désigne la forme courante.
BorderStyle	Définit le style de la bordure de la feuille. 0-None : aucune bordure 1-Fixed Single : taille non modifiable 2-Sizable : taille modifiable (par défaut) 3-Fixed Dialog : taille non modifiable, sans les boutons Réduire et Agrandir (style boîte de dialogue)
Caption	Détermine la légende de la barre de titre.
ControlBox	Indique si oui ou non le menu système apparaît sur la barre de titre.
Controls	Permet d'accéder à l'ensemble des contrôles de la feuille (collection Controls).
Count	Retourne le nombre d'objets dans une collection. Exemple : `frmTest.Controls.Count`
Icon	Spécifie l'icône de la barre de titre. Celle-ci est affichée sur la barre des tâches lorsque la fenêtre est réduite. Chemin d'accès des icônes incluses dans Visual Basic : Program Files \Microsoft Visual Studio \Common \Graphics \Icons
KeyPreview	Détermine si les événements du clavier KeyDown, KeyUp et KeyPress de la feuille sont déclenchés *avant* ceux des contrôles. Section Les événements clavier, page 206.

<table>
<tr><td>Left</td><td>Désigne la distance entre l'extrémité gauche de l'écran et l'extrémité gauche de la feuille, exprimée en twips.</td></tr>
<tr><td>MaxButton</td><td>Spécifie si le bouton Agrandir est affiché ou non dans le menu système.</td></tr>
<tr><td>MinButton</td><td>Spécifie si le bouton Réduire est affiché ou non dans le menu système.</td></tr>
<tr><td>MDIChild</td><td>Spécifie qu'une feuille est de type MDI fille et qu'elle doit être affichée à l'intérieur de la feuille mère. Par défaut, une feuille n'est pas de type MDI.</td></tr>
<tr><td>Moveable</td><td>Définit si l'utilisateur peut déplacer la feuille ou non.</td></tr>
<tr><td>Picture</td><td>Permet d'ajouter une image à une feuille.
 Exercice 9, frmBienvenue.</td></tr>
<tr><td>ScaleMode</td><td>Permet de définir une échelle à partir d'une unité de mesure standard, par exemple des twips, des points, des pixels, des caractères, des pouces, des millimètres ou des centimètres.
Les propriétés suivantes tiennent compte de l'unité de mesure fixée par ScaleMode : ScaleHeight, ScaleLeft, ScaleTop, ScaleWidth.</td></tr>
<tr><td>ShowInTaskBar</td><td>Spécifie si la feuille est affichée ou non sur la barre des tâches. Par défaut, l'icône et le titre de la barre de titre apparaissent sur la barre des tâches.</td></tr>
<tr><td>StartUpPosition</td><td>Détermine la position qu'une feuille occupe au démarrage de l'exécution.
<table>
<tr><td>0-Manual</td><td>Les propriétés Top et Left spécifient la position. (par défaut)</td></tr>
<tr><td>1-Center owner</td><td>La feuille est centrée dans une autre feuille.</td></tr>
<tr><td>2-Center screen</td><td>La feuille est centrée dans l'écran.</td></tr>
<tr><td>3-Windows default</td><td>La feuille est placée dans l'angle supérieur gauche de l'écran.</td></tr>
</table></td></tr>
<tr><td>Top</td><td>Désigne la distance entre le haut de l'écran et le haut de la feuille, exprimée en twips.</td></tr>
<tr><td>WindowState</td><td>Détermine l'état de l'affichage d'une feuille au démarrage.
<table>
<tr><td>0-Normal</td><td>taille normale</td><td>vbNormal</td></tr>
<tr><td>1-Minimized</td><td>réduite sur la barre des tâches</td><td>vbMinimized</td></tr>
<tr><td>2-Maximized</td><td>agrandie sur toute la surface de l'écran</td><td>vbMaximixed</td></tr>
</table></td></tr>
<tr><td colspan="2"> Les propriétés suivantes sont décrites dans la section 4.4 Les propriétés communes des objets : Appearance, BackColor, Enabled, Font, ForeColor, Height, MousePointer, MouseIcon, Tag, Visible et Width.</td></tr>
</table>

Les événements d'une feuille

Les événements de feuille sont déclenchés de façon souvent automatique, sans que l'utilisateur ait à les déclencher. Au démarrage d'un projet, la feuille étant identifiée comme objet de démarrage provoque une série d'événements qui se produisent selon une séquence prédéterminée. Une feuille peut aussi être créée dynamiquement. Dans ce cas, la même série d'événements sera déclenchée.

Exemple d'une feuille créée dynamiquement

```
Private Sub cmdNouvelle_Click()
  Dim frmNouvelle As New frmExemple
  frmNouvelle.Caption = "Nouvelle forme"
  frmNouvelle.Show
End Sub
```

Une nouvelle forme `frmNouvelle` est créée à partir de la forme existante `frmExemple` lors d'un clic sur un bouton de commande. Après avoir été créée, son titre est modifié et la nouvelle feuille est affichée.

Séquence des événements d'une feuille

1° Form_Initialize	Premier événement à survenir lors de la création d'une instance de l'objet Form. Il sera suivi de l'événement Load. On y code généralement des instructions d'initialisation.
	Exemple : `Me.Caption = "Exemple"`
	`mintVariable = 100`
	`cmdBouton.Enabled = False`

2° Form_Load	Événement qui se produit lorsqu'une feuille est chargée en mémoire, au démarrage d'un projet ou à la suite d'une instruction Load (réf. Les principales méthodes d'une feuille). On y code aussi des instructions d'initialisation.
	Exemple : `cmdBouton.Enabled = False`
	`lstListe.Clear`
	`cboListe.Additem "Item 1"`
	`optOption.Value = True`
	Lors des deux premiers événements, les contrôles ne sont pas encore dessinés sur la feuille. L'utilisation de la méthode SetFocus, pour donner le focus à un contrôle, provoquera l'erreur d'exécution 5 : Argument ou appel de procédure incorrect. Il faudrait plutôt invoquer la méthode SetFocus dans un événement subséquent à Form_Load.
	 Exercice 3

3°Form_Resize	Événement qui se produit lorsque la feuille est affichée pour la première fois ou lorsque sa taille est modifiée (agrandie, réduite, restaurée, redimensionnée).
4°Form_Paint	Événement qui se produit lorsque la feuille est affichée pour la première fois ou après que la feuille ait été agrandie ou déplacée.
5°Form_Activate	L'événement Activate se produit lorsque la feuille est affichée pour la première fois, au moment où elle devient active. Une feuille est activée lorsque l'utilisateur clique sur celle-ci ou lorsque les méthodes Show ou SetFocus sont invoquées (réf. Les principales méthodes d'une feuille). On reconnaît une feuille active ou inactive par la couleur de sa barre de titre. L'événement Activate se produit *avant* l'événement GotFocus. Exercice 16, Exercices de calculs
6°Form_Deactivate	L'événement optionnel Deactivate se produit lorsque la feuille n'est plus active, une autre feuille ayant été activée. Cet événement n'est pas déclenché au déchargement de la feuille. L'événement LostFocus se produit *avant* l'événement Deactivate.

Séquence des événements de fermeture d'une feuille

Lors de la fermeture d'une feuille, trois événements de feuille peuvent être déclenchés, dans la séquence suivante : QueryUnload, Unload et Terminate.

Une feuille peut être fermée de plusieurs façons :

- Par le bouton **Fermer** du menu système ou par les touches **Alt+F4** ; dans ce cas, les 3 événements de fermeture sont déclenchés.
- Par l'instruction **Unload** : dans ce cas, les 3 événements de fermeture sont déclenchés.
- Par l'instruction **End** : dans ce cas, les 3 événements de fermeture *ne sont pas déclenchés*.
- Par les instructions Unload et End : dans ce cas l'événement Terminate n'est pas déclenché.

Les instructions Unload et End sont détaillées dans la section 5.5 Les instructions d'arrêt.

1° Form_QueryUnload	Événement qui se produit *avant* la fermeture de la feuille. C'est un moment idéal pour interroger l'utilisateur (*query* signifie question). Désire-t-il vraiment fermer la feuille et ainsi quitter s'il s'agit de la dernière feuille ? Y a-t-il des données non enregistrées ? Dans le cas où l'utilisateur change d'idée, affectez la valeur **True** à l'argument **Cancel** pour annuler la fermeture. Exercice 4
2° Form_Unload	Événement déclenché à la suite de l'événement QueryUnLoad, lorsque la feuille est sur le point d'être supprimée à l'écran. On peut y coder le même genre d'instructions que dans l'événement QueryUnload ou encore afficher un message de fin d'application. L'argument **Cancel** est aussi disponible dans cet événement, permettant ainsi l'annulation du déchargement de la feuille. Exercice 3
3° Form_Terminate	Événement qui se produit *après* l'événement Unload. Il n'est pas déclenché si l'instruction End est utilisée. Quand l'instruction End est invoquée (avec ou sans Unload), elle est exécutée avant que le déchargement de la feuille se complète normalement. Dans certains cas, il est possible que l'instruction End soit nécessaire pour arrêter l'exécution de toutes les instructions. Comme par exemple, pour mettre un terme à l'exécution dans un module standard ou encore, pour interrompre un événement Timer.

Les événements **GotFocus** et **LostFocus** sont décrits dans la section Les événements *GotFocus* et *LostFocus*, page 202.

Les événements **Click**, **DblClick**, **MouseDown**, **MouseUp**, **MouseMove**, **KeyPress**, **KeyUp**, **KeyDown**, **DragDrop** et **DragOver** sont décrits dans la section 4.6 Les événements communs des objets.

Les principales méthodes d'une feuille

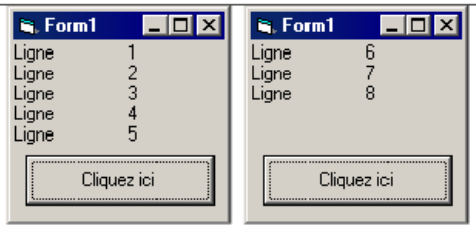
Dans la section 1.7, nous avons déjà traité de la syntaxe d'une méthode. Dans le cas d'une *feuille courante*, le mot clé **Me** peut être employé comme nom d'objet ou peut être omis.

Exemple : *objet.Méthode*

```
frmExemple.Print    frmExemple.PopupMenu
Me.Print            Me.PopupMenu
Print               PopupMenu
```

Cls	Efface de l'écran, les textes affichés par la méthode Print (Clear screen). Syntaxe : *objet*.**Cls**
Hide	Masque une feuille à l'utilisateur sans la décharger de la mémoire ; la valeur False est alors affectée à la propriété Visible de la feuille. Syntaxe: *objet*.**Hide**
PopupMenu	Affiche un menu contextuel pour la feuille. Les menus ainsi que la méthode PopupMenu sont décrits dans la section 4.7 Les menus.

Print	La méthode Print permet d'afficher un texte directement sur la feuille lors de l'exécution. Cette méthode est surtout utilisée lors des tests d'un projet.
	Syntaxe : *objet*.**Print** *liste d'impression*
	Exemple :
	`Private Sub cmdCliquer_Click()`
	`  Static I As Integer`
	`  If 0 = I Mod 5 Then`
	`  Cls`
	`  End If`
	`  I = I + 1`
	`  Print "Ligne  ", I`
	`End Sub`



À chaque exécution de la procédure, une ligne est affichée par la méthode **Print** et la variable *statique* I est augmentée de 1. L'opérateur **Mod** calcule le *reste de la division*. À toutes les 5 lignes, celui-ci vaut 0, provoquant ainsi un *effacement de l'écran* avec la méthode **Cls**.

La *liste d'impression* peut contenir plusieurs arguments. Pour plus de détails, consultez la section Les principales méthodes de l'objet Printer, page 369.

PrintForm

Imprime une image de la feuille et des objets visibles qu'elle contient. L'imprimante utilisée est désignée par les paramètres fixés dans le Panneau de configuration.

Syntaxe : *objet*.**PrintForm**

Exercice 8

Show
(instruction **Load**)

Affiche une feuille à l'écran. Si la feuille n'a pas été chargée en mémoire par l'instruction **Load**, elle sera chargée automatiquement avant d'être affichée.

Exemple : `Load frmFeuille` ou `frmFeuille.Show`
`frmFeuille.Show`

Syntaxe : *objet*.**Show** [*style*]

Le *style 0* (par défaut) spécifie que la feuille est *non modale* et le *style 1* (vbModal) spécifie que la feuille est *modale*.

Le terme **modale** désigne une fenêtre qui exige une action de la part de l'utilisateur, avant qu'une autre feuille de l'application puisse être activée. Par exemple, la boîte de dialogue affichée par *MsgBox est modale* ; l'utilisateur doit cliquer sur un bouton pour accéder à la fenêtre qui se trouve à l'arrière de la boîte de dialogue. Par défaut, une fenêtre est *non modale* ; l'utilisateur peut ainsi activer la ou les fenêtres affichées en arrière-plan.

Exemples :

```
frmFeuille.Show     ou   frmFeuille.Show 0
frmFeuille.Show 1   ou   frmFeuille.Show VbModal
```

Exercice 9, Exercice 11

Les méthodes **Move** et **SetFocus** sont décrites dans la section 4.5 Les méthodes communes des objets.

4.2 LES CONTRÔLES INTÉGRÉS DE LA BOÎTE À OUTILS

Dans cette section, seulement les propriétés, méthodes et événements *propres* à chacun des contrôles ainsi que certaines spécificités y sont décrits. Les propriétés communes sont regroupées dans la section 4.4, Les propriétés communes des objets. Les méthodes communes sont regroupées dans la section 4.5, Les méthodes communes des objets. Les événements communs sont décrits dans la section 4.6 Les événements communs des objets.

Propriété par défaut d'un contrôle

Les contrôles possèdent un grand nombre de propriétés. Parmi celles-ci, l'une d'entre elles est définie comme étant la **propriété par défaut**. C'est généralement la propriété la plus utilisée dans le code de programmation. Afin de réduire le code, la *syntaxe permet d'omettre cette propriété.*

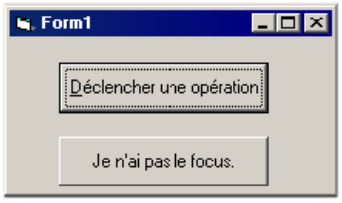
Exemple

La propriété par défaut d'une zone de texte est la propriété Text. Pour effacer une zone de texte, les deux lignes de code suivantes sont valides :

```
txtNomFamille.Text = "Entrez le nom de famille."
txtNomFamille = "Entrez le nom de famille."
```

Le contrôle *CommandButton*

Le contrôle **CommandButton** ou **Bouton de commande** permet à l'utilisateur de déclencher une opération. Lorsque le bouton est sélectionné, il donne une impression d'enfoncement, d'où l'expression *bouton-poussoir* qui est parfois utilisée. Sur le bouton, on retrouve un titre généralement constitué d'un *verbe d'action* (propriété Caption). Une *ligne pointillée* encadrant le bouton indique que celui-ci *a le focus*.



Un bouton de commande peut être sélectionné de plusieurs façons :

- Par un clic avec le bouton gauche de la souris ;

- Par le raccourci clavier, si celui-ci a été défini (exemple : Alt + D) ;
- Par la touche Espace ou Entrée lorsque le bouton a le focus ;
- Par la touche Entrée, s'il s'agit du *bouton par défaut* et que celui-ci a le focus ;
- Par la touche Échap (Escape), s'il s'agit du *bouton d'annulation*.

Lorsqu'un bouton est sélectionné, un événement Click est déclenché et la procédure événementielle correspondante est exécutée.

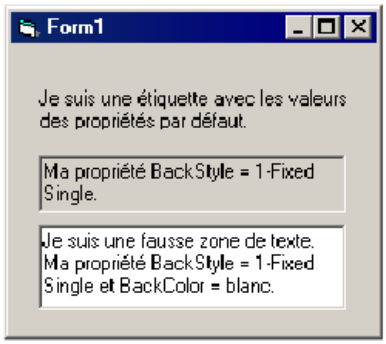
Quelques propriétés de l'objet CommandButton

(Name)	Le préfixe **cmd** identifie un objet CommandButton.
BackColor	Permet d'obtenir un bouton de couleur. La propriété **Style** doit être **1-Graphical**.
Cancel	Définit le **bouton d'annulation** d'une feuille. Ce bouton peut être sélectionné rapidement par la touche Échap (Escape). Un seul bouton peut être défini comme bouton d'annulation sur une feuille. Lorsque la propriété Cancel est fixée à True sur un bouton de commande, tous les autres boutons de la feuille reçoivent la valeur False. Il s'agit souvent d'un bouton comme Quitter, Fermer ou Annuler.
Caption	Désigne le titre affiché sur le bouton de commande. Celui-ci devrait généralement contenir un raccourci clavier défini par le caractère &.
Default	Définit le **bouton par défaut** d'une feuille. Ce bouton peut être sélectionné rapidement par la touche Entrée. Un seul bouton peut être défini comme bouton par défaut sur une feuille. Lorsque la propriété Default est fixée à True sur un bouton de commande, tous les autres boutons de commande de la feuille reçoivent la valeur False. Il s'agit généralement du bouton susceptible d'être le plus utilisé ou dans le cas de suppressions irréversibles, du bouton qui permet d'annuler la suppression. Un bouton de suppression ne doit pas être défini comme bouton par défaut car souvent, lorsqu'un utilisateur ne sait que faire, il presse la touche Entrée ou Echap. Un bouton peut être à la fois, le bouton d'annulation et le bouton par défaut.
Value	(Propriété par défaut) En *mode programmation*, cette propriété retourne une valeur spécifiant si oui ou non le bouton est sélectionné. Un bouton de commande peut être sélectionné par programmation, produisant ainsi le même résultat que si l'utilisateur avait cliqué sur le bouton. Exemple : `cmdBouton.Value = True`

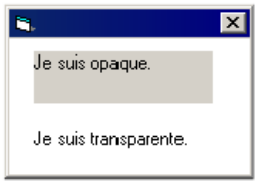
Exercice 2, Partie 1

Le contrôle *Label* A

Le contrôle **Label** ou **Étiquette** affiche un texte non modifiable par l'utilisateur.

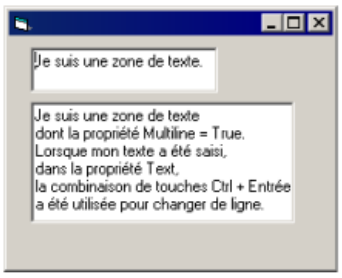


Quelques propriétés de l'objet Label

(Name)	Le préfixe **lbl** identifie un objet Label.
AutoSize	Ajuste ou non la taille de l'étiquette pour que le texte soit affiché en entier. Il est alors affiché horizontalement, sur une seule ligne. Pour que le texte soit réparti sur plusieurs lignes, fixez la valeur True à la propriété WordWrap.
BackStyle	Détermine si l'étiquette est opaque (1-Opaque, par défaut) ou transparente (0-Transparent). Exemple : 
BorderStyle	La valeur 0-None (par défaut) spécifie que l'étiquette ne possède pas de bordure et la valeur 1-Fixed Single donne une apparence d'enfoncement.
Caption	(Propriété par défaut) Définit le texte affiché dans l'étiquette. Notez que lorsque le texte est saisi dans la fenêtre Propriétés, il n'est pas possible de changer de ligne à un endroit précis. Il est parfois nécessaire de procéder à quelques ajustements pour répartir un texte sur plusieurs lignes.
WordWrap	Ajuste verticalement ou non la taille de l'étiquette pour que le texte soit affiché en entier.
Exercice 1, Partie 2 et Exercice 2, Partie 1	

Le contrôle *TextBox*

Le contrôle **TextBox** ou **Zone de texte** permet la saisie, l'édition et l'affichage d'un texte. Une zone de texte a le focus lorsque le curseur (point d'insertion) est placé à l'intérieur de celle-ci. Dans la figure qui suit, le curseur est placé au début de la première zone de texte.



Quelques propriétés de l'objet TextBox

(Name)	Le préfixe **txt** identifie un objet TextBox.
BorderStyle	La valeur 0-None spécifie que la zone de texte ne possède pas de bordure et la valeur 1-Fixed Single (par défaut) donne une apparence d'enfoncement à la zone de texte.
HideSelection	Détermine si oui ou non, un texte sélectionné demeure en surbrillance, lorsque la zone de texte perd le focus. Par défaut, il demeure en surbrillance.
Locked	Permet de verrouiller une zone de texte. Celle-ci peut recevoir le focus mais l'utilisateur ne peut en modifier le contenu.
MaxLength	Fixe un nombre maximal de caractères qui peuvent être saisis dans la zone de texte. Cette propriété est très utile pour valider les données. Par défaut, il n'y a pas de limite (MaxLength = 0). Exercice 7
Multiline	Autorise ou non la répartition du texte sur plusieurs lignes. Par défaut, une seule ligne est permise. Exercice 5
PasswordChar	Affiche un caractère spécifique (exemple : *) au lieu du contenu de la propriété Text afin de protéger la confidentialité des caractères affichés. Entrez le caractère de votre choix.

ScrollBars

Donne la possibilité d'ajouter une barre de défilement à la zone de texte.

0-None	Aucune (par défaut)
1-Horizontal	Barre horizontale
2-Vertical	Barre verticale
3-Both	Une barre horizontale et une barre verticale

Exercice 5

SelLength
SelStart
SelText

Ces propriétés permettent de *sélectionner dynamiquement* un texte ou une partie de texte ou encore de vérifier certaines propriétés d'un *texte sélectionné par l'utilisateur* (longueur, position de départ, contenu de la chaîne). Elles ne sont pas disponibles en mode création.

La propriété **SelLength** définit le nombre de caractères à sélectionner ou renvoie le nombre de caractères déjà sélectionnés.

La propriété **SelStart** définit la position de départ du texte à sélectionner ou renvoie la position de départ d'un texte sélectionné. Si aucun texte n'est sélectionné, la valeur 0 est retournée. Le premier caractère d'une zone de texte occupe la position 0.

La propriété **SelText** renvoie le contenu d'un texte sélectionné. Si aucun caractère n'est sélectionné, la propriété SelText retourne une chaîne de longueur nulle ("").

Syntaxe : *objet*.**SelLength** [**=** *longueur de la sélection*]
objet.**SelStart** [**=** *position de départ de la sélection*]
objet.**SelText** [**=** *chaîne de caractères*]

Exemples

Pour sélectionner tout le contenu d'une zone de texte :

```
txtZone.SelStart = 0
txtZone.SelLength = Len(txtZone.Text)
```

Pour vérifier les propriétés d'une sélection, on doit d'abord s'assurer qu'un texte est sélectionné :

```
If txtZone.SelText <> "" Then
    MsgBox "La sélection débute à la position " _
          & txtZone.SelStart & _
          " et a une longueur de " & _
          txtZone.SelLength
End If
```

Exercice 12, Exercice 20

Text

(Propriété par défaut)

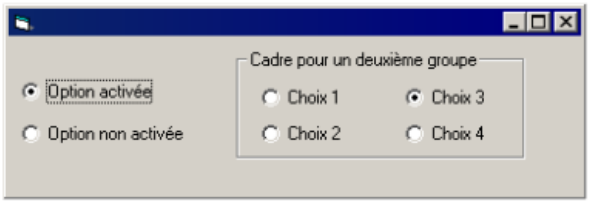
Contient une *chaîne de caractères* représentant le texte de la zone de texte. Sa limite est 2 048 caractères, sauf si la propriété MultiLine est True. Dans ce cas, la limite est 32 Ko.

Dans la fenêtre des propriétés, la combinaison de touches Ctrl+Entrée provoque un changement de ligne lorsque la propriété Multiline est fixée à True.

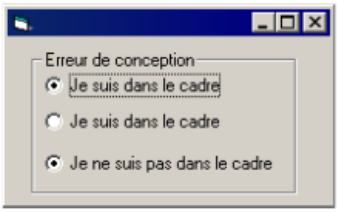
Exercice 3

Le contrôle *OptionButton*

Le contrôle **OptionButton** ou **Bouton d'option** permet à l'utilisateur de choisir une *seule option* parmi un groupe de boutons d'options. Ces contrôles doivent être regroupés dans un objet conteneur, comme un cadre (contrôle Frame) ou la feuille elle-même. Si un seul groupe de boutons d'options est nécessaire dans une feuille, l'usage d'un autre conteneur n'est pas nécessaire.



Le conteneur limite les choix à *un seul*. Dans la fenêtre qui suit, le troisième bouton n'a pas été dessiné à l'intérieur du cadre. Pour corriger l'erreur, il faut couper le troisième bouton et le coller dans le cadre.



Un bouton d'option a le focus lorsqu'une ligne pointillée entoure sa légende. Un bouton d'option est activé par un clic avec le bouton gauche de la souris ou par le raccourci clavier, si celui-ci a été défini (Alt + lettre soulignée).

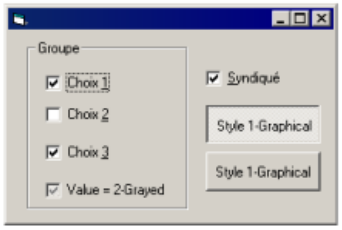
Lorsqu'un bouton d'option est activé, un événement Click est déclenché et la procédure événementielle correspondante est exécutée.

Quelques propriétés de l'objet OptionButton

(Name)	Le préfixe **opt** identifie un objet OptionButton.
Caption	Désigne le titre du bouton d'option.
Style	Définit le style du bouton d'option. Le style 0-None est le style par défaut et le style 1-Graphical donne une apparence semblable à un bouton de commande. Un bouton d'option activé est enfoncé et grisé lorsqu'il est sélectionné. Dans la fenêtre qui suit, le bouton rouge est le bouton actif et sa couleur rouge est estompée.
Value	(Propriété par défaut) Détermine l'état d'un bouton d'option : activé ou désactivé. Lorsque la propriété Value d'un bouton prend la valeur True, la propriété Value de tous les autres boutons du groupe prend la valeur False. En mode exécution, un bouton d'option peut être activé/désactivé, produisant ainsi le même résultat que si l'utilisateur avait cliqué sur le bouton. La procédure événementielle Click du bouton est alors exécutée. Au démarrage, l'un des boutons devrait toujours être activé. Exemple : `optChoix.Value = True`
Exercice 3	

Le contrôle Checkbox

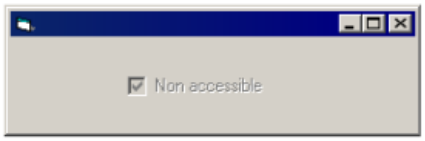
Le contrôle **Checkbox** ou **Case à cocher** permet à l'utilisateur d'activer ou de désactiver une option. Il est possible, mais non nécessaire, comme les boutons d'option, de regrouper visuellement plusieurs cases à cocher dans un conteneur.



Une case à cocher a le focus lorsqu'une ligne pointillée entoure sa légende. Elle est activée lorsqu'une coche apparaît dans la case. L'utilisateur active ou désactive une case à cocher en cliquant avec le bouton gauche de la souris ou par le raccourci clavier, si celui-ci a été défini par la propriété Caption.

Lorsqu'une case à cocher est cochée ou décochée, un événement Click est déclenché et la procédure événementielle correspondante est exécutée.

Quelques propriétés de l'objet CheckBox

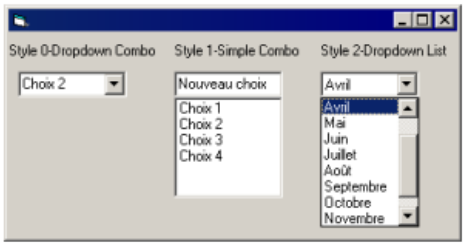
<table>
<tr><td>(Name)</td><td>Le préfixe chk identifie un objet Checkbox.</td></tr>
<tr><td>Caption</td><td>Détermine la légende de la case à cocher.</td></tr>
<tr><td>Style</td><td>Définit le style de la case à cocher. Le style 0-None est le style par défaut et le style 1-Graphical donne une apparence semblable à un bouton de commande. Une case à cocher ayant ce style, est active lorsqu'elle est enfoncée.</td></tr>
<tr><td>Value</td><td>(Propriété par défaut)
Détermine l'état de la case à cocher : activée, désactivée ou estompée.
<table>
<tr><td>0-Unchecked</td><td>non cochée, désactivée</td><td>vbUnChecked</td></tr>
<tr><td>1-Checked</td><td>cochée, activée</td><td>vbChecked</td></tr>
<tr><td>2-Grayed</td><td>grisée, estompée</td><td>vbGrayed</td></tr>
</table>
La valeur 2-Grayed devrait être utilisée conjointement avec la propriété Enabled = False. La case à cocher a ainsi une apparence d'inaccessibilité et de plus, elle est réellement inaccessible et sa légende est grisée.
 

En mode exécution, une case à cocher peut être activée/désactivée, produisant ainsi le même résultat que si l'utilisateur l'avait cochée/décochée lui-même. La procédure événementielle Click de la case à cocher est exécutée si l'état de la case à cocher est modifié.
Exemple :
<pre>chkSyndique.Value = vbChecked
chkSyndique.Value = vbUnChecked</pre></td></tr>
<tr><td colspan="2">Exercice 3</td></tr>
</table>

Les contrôles *ListBox* et *ComboBox*

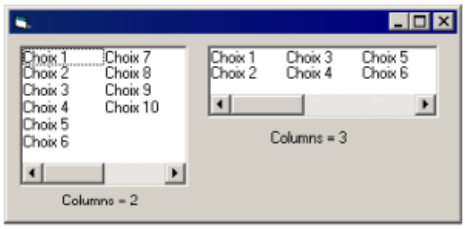
Le contrôle **ListBox** ou **Zone de liste** permet à l'utilisateur de sélectionner un ou plusieurs éléments dans une liste. Si le nombre d'éléments dépasse le nombre de lignes affichables, une barre de défilement est automatiquement ajoutée à droite.

Le contrôle **ComboBox** ou **Zone de liste modifiable** est une liste semblable à une zone de liste avec en plus une *zone de texte* qui permet à l'utilisateur d'ajouter des éléments dans la liste. En cliquant sur une pointe de flèche, l'utilisateur déroule la liste, affichant ainsi l'ensemble des éléments.

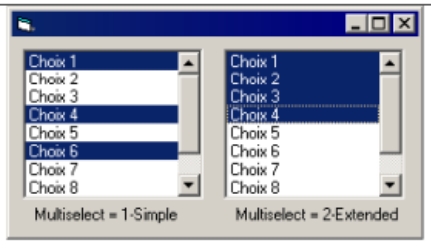


Dans une zone de liste, un élément est sélectionné par un clic ou par l'utilisation des flèches ↑ et ↓ à l'intérieur de la zone. Lorsqu'un élément est sélectionné, un événement Click est déclenché et la procédure événementielle correspondante est exécutée. Lorsqu'une zone de liste n'est pas modifiable, l'utilisateur peut taper la *première lettre d'un élément*, afin d'accéder plus rapidement à un élément précis dans une liste triée en ordre alphabétique.

Quelques propriétés des objets ListBox et ComboBox

(Name)	Le préfixe **lst** identifie un objet **ListBox** et le préfixe **cbo** identifie un objet **ComboBox**.
Columns (ListBox)	Permet d'afficher les éléments de la liste en plusieurs colonnes. Une barre de défilement horizontale est ajoutée au lieu d'une barre verticale. Par défaut, la valeur est 0 et une ligne contient un seul élément. 

<table>
<tr><td>List</td><td>Une liste est un tableau de chaînes de caractères, constitué de plusieurs éléments identifiés par un numéro d'index. La valeur d'index d'un élément est la valeur de sa position dans la liste – 1. Par exemple, le deuxième élément de la liste porte l'index 1.
En mode création, cette propriété permet d'inscrire les éléments de la liste. Chaque élément occupe une ligne et la combinaison de touches Ctrl+Entrée permet de changer de ligne avant d'entrer un autre élément.
Dans le code de programmation, la syntaxe suivante permet de référencer un élément de la liste :
objet.List(index) [= chaîne de caractères]
Exemples :
Pour modifier la quatrième valeur de la liste :
<pre>lstListe.List(3) = "Nouvelle valeur"</pre>Pour afficher la troisième valeur de la liste dans une étiquette :
<pre>lblTroisieme.Caption = lstListe.List(2)</pre></td></tr>
<tr><td>ListCount</td><td>En mode exécution, cette propriété retourne le nombre d'éléments dans la liste.</td></tr>
<tr><td>ListIndex</td><td>En mode exécution, cette propriété retourne l'index de l'élément actuellement sélectionné dans la liste.
Dans le cas d'un contrôle ComboBox modifiable, la valeur – 1 est retournée si aucun élément n'est sélectionné ou si l'utilisateur a entré une valeur dans la zone de texte.
Exemple : Afficher la position de l'élément sélectionné :
<pre>MsgBox "L'élément " & _
        lstListe.ListIndex + 1 & _
        " est sélectionné.", , ""</pre>
</td></tr>
<tr><td>MultiSelect
(ListBox)</td><td>Autorise ou non la sélection multiple d'éléments.
<table>
<tr><td>0-None</td><td>La sélection multiple n'est pas autorisée. Un seul élément peut être sélectionné dans la liste. (par défaut)</td></tr>
<tr><td>1-Simple</td><td>La sélection multiple simple est autorisée. L'utilisateur sélectionne et annule une sélection par un clic ou par la touche Espace.</td></tr>
<tr><td>2-Extended</td><td>La sélection multiple étendue est autorisée. L'enfoncement de la touche Maj pendant la sélection permet de sélectionner des éléments consécutifs. La touche Ctrl combinée à un clic de souris sélectionne un élément dans la liste ou annule une sélection. La touche Espace annule une sélection multiple.</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>NewIndex</td><td>En mode exécution, cette propriété retourne l'index du dernier élément ajouté dans la liste.</td></tr>
<tr><td>Selected
(ListBox)</td><td>En mode exécution, cette propriété permet de vérifier si un ou plusieurs éléments sont sélectionnés. La propriété Selected est constituée d'un tableau de valeurs booléennes (True/False) comprenant le même nombre d'éléments que la propriété List. Dans le cas d'une sélection multiple, cette propriété est utile pour déterminer quels sont les éléments sélectionnés.</td></tr>
<tr><td>Sorted</td><td>Détermine si oui ou non les éléments de la liste sont triés en ordre alphabétique.
Lorsque la méthode AddItem est utilisée pour ajouter un élément à un endroit spécifique de la liste, il est possible que l'ordre de tri soit perturbé.</td></tr>
<tr><td>Style</td><td>Permet de modifier l'apparence d'une zone de liste.
Le contrôle ListBox comporte deux styles : 0-Standard (par défaut) et 1-CheckBox.
Le style CheckBox ajoute une case à cocher en avant des éléments de la liste. L'utilisateur peut sélectionner plusieurs éléments en cochant dans chaque case, même si la propriété MultiSelect est fixée à 0-None. Lorsque l'utilisateur coche ou décoche la case, un événement ItemCheck est déclenché.
Le contrôle ComboBox comporte trois styles :
<table>
<tr><td>0-Dropdown Combo</td><td>Liste déroulante modifiable (par défaut).
L'utilisateur sélectionne un élément dans la liste déroulée ou il entre un nouvel élément dans la zone de texte.</td></tr>
<tr><td>1-Simple Combo</td><td>Liste modifiable simple non déroulante.
L'utilisateur sélectionne un élément dans la liste ou il entre un nouvel élément dans la zone de texte.</td></tr>
<tr><td>2-Dropdown List</td><td>Liste déroulante non modifiable.
L'utilisateur sélectionne un élément dans la liste déroulée mais n'a pas la possibilité d'ajouter de nouveaux éléments.</td></tr>
</table></td></tr>
<tr><td>Text</td><td>(Propriété par défaut)
Représente le contenu de la zone de texte dans une liste modifiable ou l'élément sélectionné dans une liste non modifiable.
En mode création, pour définir un élément sélectionné par défaut, tapez la chaîne de caractères de l'élément. Une chaîne de caractères identique doit être inscrite dans la propriété List.</td></tr>
</table>

	En *mode programmation*, pour sélectionner le premier élément d'une liste, n'affectez pas de valeur à la propriété Text en mode création et codez comme suit :
	`lstListe.Text = lstListe.List(0)`
	`cboListe.Text = cboListe.List(0)`

TopIndex	En mode exécution, cette propriété retourne ou définit une valeur spécifiant quel élément d'une liste doit apparaître en première position. L'élément concerné n'est cependant pas sélectionné. Syntaxe : *objet*.**TopIndex** [**=** *valeur d'index*]
Exercice 3	

Quelques méthodes des objets ListBox et ComboBox

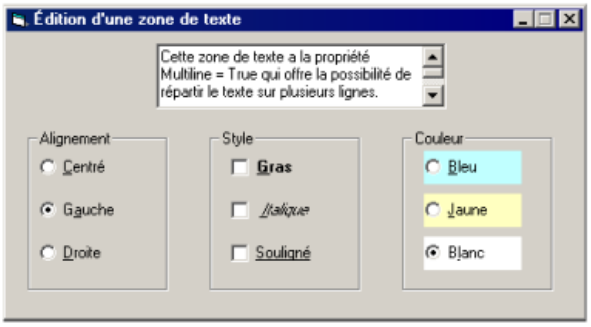
AddItem	Ajoute un élément à une zone de liste. Syntaxe : *objet*.**AddItem** *nouvel élément*[, *index*] L'argument *index* spécifie une position dans la liste. S'il est omis, le nouvel élément est ajouté à la fin de la liste ou à la position appropriée si la propriété Sorted est fixée à True. Exemple : `cboListe.AddItem "Quatrième valeur", 3` `cboListe.AddItem "Nouvelle valeur"`
Clear	Efface tous les éléments d'une zone de liste. Syntaxe : *objet*.**Clear** Exemple : `cboListe.Clear`

RemoveItem	Supprime un élément d'une zone de liste.
	Syntaxe : *objet*.**RemoveItem** *index*
	L'argument *index* spécifie la position de l'élément à supprimer dans la liste.
	Exemples :
	Pour effacer le deuxième élément de la liste :
	`cboListe.RemoveItem 1`
	Pour effacer l'élément sélectionné :
	`cboListe.RemoveItem cboListe.ListIndex`

Exercice 8	

Le contrôle *Frame*

Le contrôle **Frame** ou **Cadre** est un **conteneur** qui permet de regrouper visuellement des contrôles. Certains contrôles comme les boutons d'option sont regroupés logiquement dans un cadre, afin qu'un seul des boutons du groupe soit activé. Dans la fenêtre qui suit, trois cadres permettent de regrouper des éléments semblables :



Pour regrouper des contrôles, tracez d'abord le cadre, puis *dessinez* les contrôles à l'intérieur de celui-ci. *Un contrôle ajouté à un cadre par un double-clic n'est pas inclus dans celui-ci.* Lorsqu'un cadre est déplacé, tous les contrôles qu'il contient sont aussi déplacés. Pour ajouter des contrôles existants dans un cadre, *coupez* les *contrôles* et *collez-les* à l'intérieur du cadre.

Quelques propriétés de l'objet Frame

(Name)	Le préfixe **fra** identifie un objet Frame.
BorderStyle	Spécifie si le cadre a une bordure (1-Fixed Single, par défaut) ou non (0-None).
Caption	(Propriété par défaut) Spécifie le titre du cadre qui devrait identifier l'ensemble des contrôles que le cadre contient.
Enabled	Lorsqu'un cadre n'est pas accessible à l'utilisateur (Enabled = False), tous les contrôles qu'il contient sont inaccessibles.
Visible	Lorsqu'un cadre est masqué (Visible = False), tous les contrôles qu'il contient ne sont pas visibles.
Exercice 3, Exercice 5	

Le contrôle *Timer*

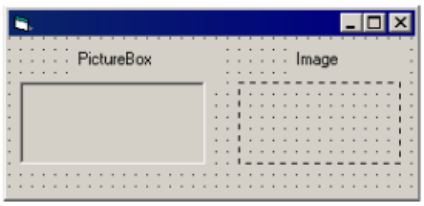
Le contrôle **Timer** ou **Minuterie** permet l'exécution d'une procédure événementielle **Timer**, à un intervalle de temps fixé par la propriété **Interval**. Le contrôle Timer ne reconnaît qu'un seul événement : l'événement **Timer**. Le contrôle Timer n'est pas visible à l'exécution et plusieurs minuteries peuvent être utilisées sur une même feuille.

Quelques propriétés de l'objet Timer

(Name)	Le préfixe **tmr** identifie un objet Timer.
Enabled	(Propriété par défaut) La valeur True spécifie de mettre la minuterie en marche et la valeur False spécifie d'arrêter la minuterie.
Interval	Spécifie un intervalle de temps exprimé en millisecondes. Par exemple, la valeur 1 000 équivaut à une seconde, la valeur 100 à un dixième de seconde et la valeur 10 000 à 10 secondes. La valeur 0 désactive la minuterie et la valeur maximale est 65 535 (un peu plus d'une minute).
Exercice 4	

Les contrôles *PictureBox* et *Image*

Les **Zones d'image, PictureBox** et **Image** permettent d'ajouter des images sur une feuille. Il peut s'agir d'images bitmap, d'icônes, de métafichiers, de métafichiers étendus, de fichiers JPEG ou GIF.

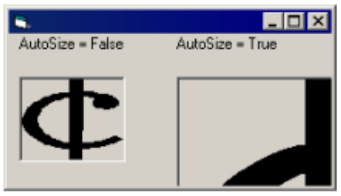
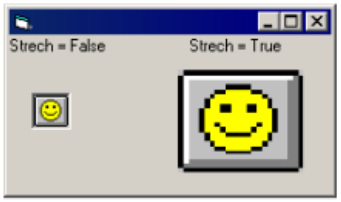


Le contrôle **Image** est une version allégée du contrôle PictureBox. Il ne s'emploie que pour afficher des images et utilise beaucoup moins de ressources système que le contrôle PictureBox.

En plus de l'affichage des images, le contrôle **PictureBox** peut être utilisé comme conteneur pour regrouper des boutons d'option ou pour afficher des dessins réalisés par des méthodes graphiques, comme Circle et Line, et des textes affichés par la méthode Print.

Exemple : `picZone.Print "texte affiché dans un contrôle PictureBox"`

Quelques propriétés des objets PictureBox et Image

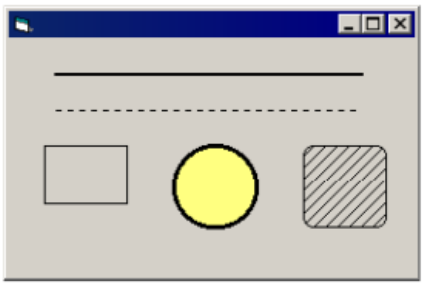
(Name)	Le préfixe **pic** identifie un objet **PictureBox** et le préfixe **img** identifie un objet **Image**.
AutoSize (PictureBox)	La valeur True redimensionne automatiquement la zone d'image en fonction de la taille de l'image. Si la valeur est False (par défaut), l'image est découpée et seulement une partie de l'image sera visible. 
BorderStyle	Spécifie si la zone d'image a une bordure (1-Fixed Single) ou non (0-None). Par défaut, un contrôle PictureBox a une bordure et un contrôle Image n'a pas de bordure.
Picture	(Propriété par défaut) Désigne l'image affichée dans la zone d'image. En *mode création*, cette propriété affiche la boîte de dialogue "Charger une image" qui permet de parcourir les dossiers afin de retrouver le fichier image. En *mode programmation*, utilisez la fonction LoadPicture pour charger un fichier image dans la zone d'image. Exemple : `picZone.Picture = LoadPicture("C:\Windows\Happy.bmp")`
Stretch (Image)	La valeur False (par défaut), redimensionne automatiquement la zone d'image en fonction de la taille de l'image. Si la valeur est True, c'est l'image qui est redimensionnée selon la taille du contrôle Image, ce qui peut engendrer une déformation de l'image. 
Exercice 2 et Exercice 16, parties 2 et 4	

Le contrôle *Line* et le contrôle *Shape*

Le contrôle **Line** ou **Ligne** est un contrôle graphique qui permet d'ajouter des lignes horizontales, verticales ou diagonales sur une feuille.

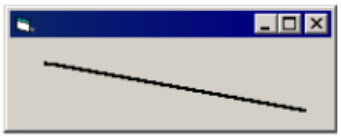
Le contrôle **Shape** ou **Formes** est un contrôle graphique qui permet d'ajouter des formes sur une feuille.

Les contrôles Line et Shape ne réagissent à aucun événement. Ils sont strictement des éléments décoratifs.



Quelques propriétés des objets Line et Shape

(Name)	Le préfixe **lin** identifie un objet **Line** et le préfixe **shp** identifie un objet **Shape**.
BorderColor	Désigne la couleur de la bordure (par défaut, noire).
BorderStyle	Spécifie le style d'une ligne ou le style de la bordure d'une forme (par défaut, 1-Solid, ligne continue).
BorderWitdth	Définit la largeur de la bordure par un nombre entier. Plus le nombre est élevé, plus la bordure est large (par défaut, 1).
DrawMode	Détermine l'aspect du contrôle ou le résultat produit par une méthode graphique. Par défaut, la valeur 13-Copy Pen spécifie que la couleur est celle spécifiée par la propriété FillColor.
FillColor (Shape)	Spécifie la couleur de remplissage d'une forme (par défaut, noire). Pour une forme pleine, modifiez la valeur par défaut de la propriété FillStyle.
FillStyle (Shape)	Spécifie le motif de remplissage d'une forme. Le style par défaut 1-Transparent est incolore et uni. Le style 0-Solid est coloré et uni. Les autres styles contiennent différents motifs.

<table>
<tr><td>Shape
(Shape)</td><td>(Propriété par défaut)
Spécifie l'apparence de la forme.<table><tr><td>0-Rectangle</td><td>Rectangle</td></tr><tr><td>1-Square</td><td>Carré</td></tr><tr><td>2-Oval</td><td>Ovale</td></tr><tr><td>3-Circle</td><td>Cercle</td></tr><tr><td>4-Rounded rectangle</td><td>Rectangle avec coins arrondis</td></tr><tr><td>5-Rounded square</td><td>Carré avec coins arrondis</td></tr></table></td></tr>
<tr><td>X1, X2, Y1, Y2
(Line)</td><td>Représentent les coordonnées horizontales et verticales d'une ligne.
Exemple :
 
X1 : Point de départ horizontal (360)
Y1 : Point de départ vertical (240)
X2 : Point de fin horizontal (3 000)
Y2 : Point de fin vertical (720)
En mode exécution, ces coordonnées peuvent être utilisées pour déplacer une ligne ou pour modifier sa taille.</td></tr>
<tr><td>Visible</td><td>Propriété par défaut du contrôle Line. Par défaut, une ligne est visible.</td></tr>
<tr><td colspan="2">Exercice 6 (Shape) et 7 (Line)</td></tr>
</table>

Les barres de défilement *VScrollBar* et *HScrollBar*

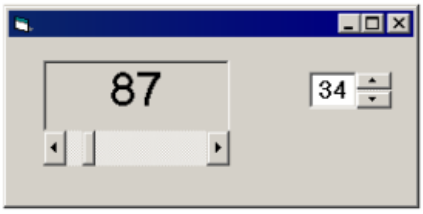
Les contrôles **VScrollBar** (**Barre de défilement verticale**) et **HScrollBar** (**Barre de défilement horizontale**) permettent de se déplacer rapidement dans de longues listes d'éléments. Elles servent aussi à augmenter ou à diminuer la valeur d'un indicateur dans une plage de valeurs. La position du curseur de défilement dans la barre représente la valeur actuelle de l'indicateur. Il peut s'agir, par exemple, d'un indicateur de vitesse, de quantité, de niveau sonore ou de temps écoulé.

Quelques propriétés des objets VScrollBar et HScrollBar

(Name)	Le préfixe **vsb** identifie un objet **VScrollBar** et le préfixe **hsb** identifie un objet **HScrollBar**.
LargeChange	Spécifie la valeur d'augmentation ou de diminution (incrément) de l'indicateur, lorsque l'utilisateur clique sur la zone entre le curseur de défilement et la flèche de défilement. Valeurs possibles : 1 à 32 767, valeur par défaut : 1.
Max	Représente la valeur maximale de la propriété Value. C'est la plus grande valeur que l'indicateur peut prendre. Le curseur de défilement est alors placé en bas ou à droite de la barre selon son orientation. Valeurs possibles : –32 768 à 32 767, valeur par défaut : 32 767.
Min	Représente la valeur minimale de la propriété Value. C'est la plus petite valeur que l'indicateur peut prendre. Le curseur de défilement est alors placé en haut ou à gauche de la barre selon son orientation. Valeurs possibles : -32 768 à 32 767, valeur par défaut : 0.
SmallChange	Spécifie la valeur d'augmentation ou de diminution (incrément) de l'indicateur lorsque l'utilisateur clique sur la flèche de défilement. Valeurs possibles : 1 à 32 767, valeur par défaut : 1.
Value	(Propriété par défaut) Indique la position actuelle du curseur dans la barre de défilement. Cette valeur est nécessairement dans la plage définie par les propriétés Max et Min.

Exemple d'utilisation

La fenêtre qui suit contient une étiquette jumelée avec une barre de défilement horizontale (Min = 100, Max = 0) et une zone de texte jumelée avec une barre de défilement verticale (Min = 0, Max = 50).



```
Private Sub Text1_Change()
  vsbBarre.Value = Val(txtNombre.Text)
End Sub

Private Sub hsbBarre_change()
  lblNombre.Caption = hsbBarre.Value
End Sub

Private Sub vsbBarre_change()
  txtNombre.Text = vsbBarre.Value
End Sub
```

À chaque fois que la valeur de la barre de défilement horizontale change, la nouvelle valeur est affichée dans l'étiquette correspondante, par exemple, 87 (procédure `hsbBarre_Change`).

À chaque fois que la valeur de la barre de défilement verticale change, la nouvelle valeur est affichée dans la zone de texte correspondante, par exemple, 34 (procédure `vsbBarre_Change`). Lorsque l'utilisateur tape une valeur dans la zone de texte, la barre de défilement est synchronisée avec la valeur entrée (procédure `Text1_Change`).

Un contrôle UpDown est utilisé dans l'exercice 28. Son fonctionnement est semblable à celui des contrôles VScrollBar et HScrollBar.

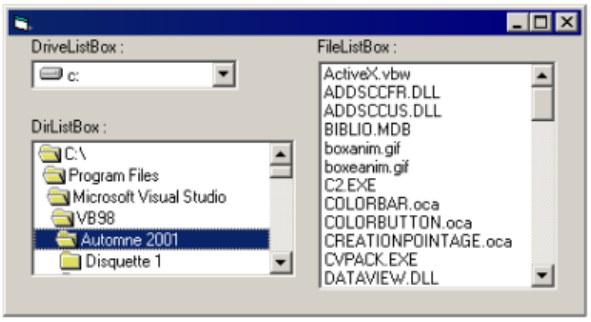
Les contrôles *DriveListBox* , *DirListBox* et *FileListBox*

Un contrôle **DriveListBox** ou **Liste de lecteurs** permet à l'utilisateur de sélectionner un lecteur valide parmi tous les lecteurs de son système. Pour sélectionner un lecteur, cliquez sur son nom ou utilisez les flèches ↓ ou ↑ dans la liste.

Un contrôle **DirListBox** ou **Liste de dossiers** affiche une liste hiérarchique des dossiers et des chemins d'accès. Pour sélectionner un dossier, double-cliquez sur son nom dans la liste.

Un contrôle **FileListBox** affiche la **Liste des fichiers** du dossier spécifié par la propriété **Path**. Pour sélectionner un fichier, cliquez sur son nom ou utilisez les flèches ↓ ou ↑ dans la liste.

En combinant ces trois contrôles, il est possible, par exemple, de créer une boîte de dialogue qui permet à l'utilisateur de spécifier le chemin d'accès d'un fichier (lecteur/dossier/nom du fichier).



Quelques propriétés des objets DriveListBox, DirListBox et FileListBox

(Name)	Le préfixe **drv** identifie un objet **DriveListBox**. Le préfixe **dir** identifie un objet **DirListBox**. Le préfixe **fil** identifie un objet **FileListBox**.
Archive, Hidden, Normal, System (FileListBox)	Ces propriétés spécifient, si oui ou non le contrôle FileListBox affiche les fichiers ayant les attributs Archivé, Caché, Normal ou Système. Par défaut, seulement les fichiers Archive et Normal sont affichés.
Drive (DriveListBox)	En mode exécution, cette propriété retourne le lecteur sélectionné. Par défaut, le lecteur sélectionné est celui en cours. Le lecteur sélectionné est renvoyé sous l'une des formes suivantes : Disquettes : a: Support fixe : c: [*nom du volume*] Connexion réseau : x: \\serveur\ressource
FileName (FileListBox)	Représente le nom du fichier sélectionné dans un contrôle FileListBox. Si aucun fichier n'est sélectionné, la valeur retournée est nulle ("").
List	Cette propriété représente un *tableau* de chaînes de caractères, constitué de plusieurs éléments identifiés par un *index*. La valeur d'index d'un élément est la valeur de sa position dans la liste – 1. Le contrôle **DriveListBox** contient la liste de tous les lecteurs connectés. L'index 0 désigne le premier lecteur de la liste. Le contrôle **DirListBox** contient la liste de tous les dossiers. L'index –1 désigne le dossier sélectionné dans la liste. Un dossier situé à un niveau supérieur du dossier sélectionné porte un index négatif, et un dossier situé à un niveau inférieur porte un index d'une valeur de 0 et plus. La propriété List retourne le chemin d'accès du dossier sélectionné (exemple : C:\Windows). Le contrôle **FileListBox** contient les noms des fichiers d'un dossier spécifique. L'index 0 désigne le premier fichier de la liste.

<table>
<tr><td>ListCount</td><td>En mode exécution, cette propriété retourne le nombre d'éléments dans une liste.</td></tr>
<tr><td>ListIndex</td><td>En mode exécution, cette propriété retourne l'index de l'élément actuellement sélectionné dans la liste.</td></tr>
<tr><td>MultiSelect
(FileListBox)</td><td>Autorise ou non la sélection multiple des fichiers.
<table>
<tr><td>0-None</td><td>La sélection multiple n'est pas autorisée. Un seul élément peut être sélectionné dans la liste.</td></tr>
<tr><td>1-Simple</td><td>La sélection multiple simple est autorisée. L'utilisateur sélectionne et annule une sélection par un clic ou par la touche Espace.</td></tr>
<tr><td>2-Extended</td><td>La sélection multiple étendue est autorisée. L'enfoncement de la touche Maj pendant la sélection permet de sélectionner des éléments consécutifs. La touche Ctrl combinée à un clic de souris sélectionne un élément dans la liste ou annule une sélection. La touche Espace annule une sélection multiple.</td></tr>
</table></td></tr>
<tr><td>Path
(DirListBox)
(FileListBox)</td><td>En mode exécution, cette propriété retourne le chemin d'accès du dossier ou du fichier sélectionné dans un contrôle DirListBox ou FileListBox.</td></tr>
<tr><td>Pattern
(FileListBox)</td><td>Cette propriété permet de filtrer les fichiers affichés dans une liste. Par défaut cette propriété contient la valeur *.* (tous les fichiers). Le point-virgule permet de séparer deux filtres. Exemple : *.icol;*.cur;*.bmp</td></tr>
<tr><td>ReadOnly
(FileListBox)</td><td>Détermine si oui ou non les fichiers en lecture seule sont affichés dans la liste. Par défaut, ils sont affichés.</td></tr>
<tr><td>Selected
(FileListBox)</td><td>En mode exécution, cette propriété permet de vérifier si un ou plusieurs éléments sont sélectionnés. Cette propriété constitue un tableau de valeurs booléennes (True/False) comprenant le même nombre d'éléments que la propriété List.</td></tr>
<tr><td colspan="2"> Exercice 18</td></tr>
</table>

Le contrôle *OLE*

Le contrôle conteneur **OLE** (Object Linking and Embedding) ou **Liaison et incorporation d'objets** permet d'ajouter des objets à une application Visual Basic, en provenance d'une autre application Windows.

Un objet, comme par exemple, une feuille de calcul Excell ou un document Word, provient d'une **application source** qui a créé l'objet (Excell, Word) et il peut être ajouté à une **application destination** (Visual Basic).

Lors de l'échange d'un objet entre deux applications, celui-ci peut être **incorporé** ou **lié**. Lorsqu'un objet est *incorporé*, il n'est plus modifiable par l'application source, seulement par l'application destination. Lorsqu'un objet est *lié*, celui-ci peut être modifié par l'application source et par l'application destination. L'application Visual Basic conserve l'emplacement de l'objet au moyen d'un pointeur.

Le contrôle *Data*

Le contrôle **Data** ou **Contrôle de données** est un contrôle graphique qui permet un accès rapide aux données enregistrées dans une base de données. Il est muni de quatre boutons de navigation permettant à l'utilisateur de se déplacer dans les différents enregistrements d'une table de données.

Dans la version 6.0 de Visual Basic, ce contrôle a été remplacé avantageusement par le contrôle graphique ADO Data (ActiveX Data Objects). Il est recommandé de n'utiliser que ce dernier dans de nouveaux projets, celui-ci étant beaucoup plus performant que son ancêtre.

> **Note :**
>
> La *gestion des bases de données* fera l'objet d'un prochain volume. actuellement en

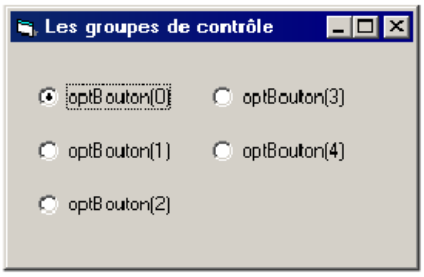
4.3 LES GROUPES DE CONTRÔLES (PROPRIÉTÉ INDEX)

Un **groupe de contrôles** est un ensemble de contrôles qui font partie de la même classe (exemple : OptionButton, TextBox, CommandButton) et qui partagent le *même nom* (exemple : optBouton) et les *mêmes procédures événementielles*.

La propriété Index

La valeur de la **propriété Index** de chaque contrôle du groupe permet de différencier les éléments d'un groupe de contrôles. Par défaut, la valeur des index varie de 0 à $n-1$, n étant le nombre d'éléments dans le groupe. Par exemple, si le groupe compte

5 contrôles optBouton, les contrôles seront identifiés par optBouton(0) à optBouton(4). Toutefois, il est possible d'ajuster les valeurs d'index selon des besoins spécifiques dans la fenêtre Propriétés.



L'utilisation de groupes de contrôles permet de réduire le code de programmation, puisque plusieurs contrôles partagent une *procédure événementielle commune*. Lorsqu'une procédure commune est ajoutée à un projet, l'argument **Index** apparaît automatiquement à la suite du nom de la procédure.

Exemple

```
Private Sub optBouton_Click(Index As Integer)
  optBouton(Index).Caption = "Bouton cliqué"
End Sub
```

La valeur de l'argument Index est mise à jour automatiquement, lorsque le bouton est cliqué. Si le deuxième bouton est cliqué, l'argument Index prend la valeur 1. Dans la figure précédente, une seule ligne de code permet de modifier le titre de tous les boutons d'option.

Lorsque le nom d'un contrôle groupé est cité, il doit *toujours être suivi d'un index entre parenthèses*. Par exemple, Visual Basic ne reconnaît pas la ligne de code suivante :

```
optBouton.Caption = "Bouton cliqué"
```

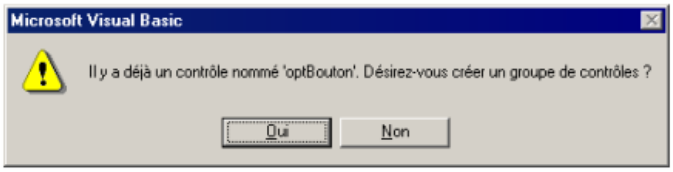
Création d'un groupe de contrôles

En **mode création**, un groupe de contrôles peut être créé :

- En affectant le même nom à plusieurs contrôles (propriété Name) ;
- En copiant un contrôle existant ;
- En modifiant la propriété Index d'un contrôle.

Affecter le même nom

Lorsque l'on tente d'attribuer un nom existant à un contrôle ou de copier un contrôle, Visual Basic affiche le message suivant :

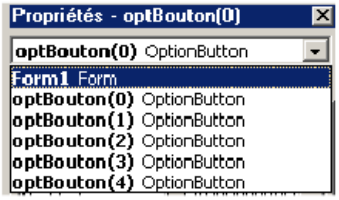


En répondant Oui à ce message, le contrôle existant sera nommé optBouton(0) et le nouveau, optBouton(1). La propriété Index de chacun des contrôles sera ajustée automatiquement.

Copier un contrôle existant

La copie d'un contrôle existant est une façon rapide de créer un groupe de contrôles et d'attribuer les mêmes propriétés à tous les contrôles du groupe. Procédez comme suit :

- Créez le premier contrôle, nommez-le et fixez les propriétés qui doivent être communes à tous les contrôles, avant de copier. Par exemple, la police de caractères et la couleur d'arrière-plan.
- Copiez le contrôle par Ctrl+C, sélectionnez le conteneur (forme ou cadre) et collez-y le contrôle par Ctrl+V.
- Répondez Oui au message pour créer un groupe de contrôles.
- Placez le contrôle à l'endroit désiré.
- Copiez les autres contrôles.
- Observez le résultat dans la liste de sélection des objets de la fenêtre Propriétés :



Modifier la propriété Index

Dans la fenêtre Propriétés, aussitôt qu'une valeur est attribuée à la propriété Index, un groupe de contrôles est créé automatiquement. Cette façon de procéder est souvent une *source d'erreur* puisque l'attribution d'un index à un contrôle modifie le nom de celui-ci. Par exemple, le contrôle txtNom qui porte l'index 3 est reconnu seulement par l'expression txtNom(3) dans le code de programmation.

Exercice 5, Exercice 6 et Exercice 9.

Ajout et suppression dynamique d'un contrôle dans un groupe

En **mode exécution**, un contrôle peut être créé dynamiquement par l'instruction **Load** et supprimé par l'instruction **Unload**.

Syntaxe : **Load** *objet(Index)*

Unload *objet(Index)*

Exemple

```
Load optBouton(5)

Unload optBouton(5)
```

L'instruction Load ajoute un sixième bouton au groupe et l'instruction Unload le supprime. L'instruction Unload ne peut supprimer que des contrôles créés par l'instruction Load.

4.4 LES PROPRIÉTÉS COMMUNES DES OBJETS

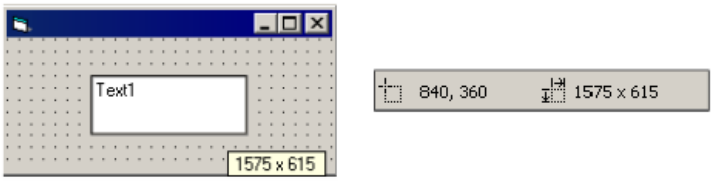
Plusieurs propriétés sont communes aux mêmes objets (forme et contrôles). Cette section regroupe la plupart d'entre elles, avec toutefois certaines spécificités que l'on peut retrouver d'un objet à l'autre.

Les propriétés *Left*, *Top*, *Height* et *Width*

- La propriété **Left** désigne la distance entre l'extrémité *gauche* d'un objet et l'extrémité *gauche* de son conteneur.
- La propriété **Top** désigne la distance entre l'extrémité supérieure d'un objet et l'extrémité *supérieure* de son conteneur.
- La propriété **Height** désigne la *hauteur* d'un objet.
- La propriété **Width** désigne la *largeur* d'un objet.

Dans le cas d'une feuille, le conteneur est l'écran. Pour les contrôles, le conteneur peut être la feuille ou un cadre. Ces propriétés sont exprimées en nombre de **twips**. Un *twip* est une unité de mesure d'écran égale à 1/20 point. Un pouce contient environ 1 440 twips et un centimètre contient approximativement 567 twips.

Lorsqu'un contrôle est dessiné sur la feuille, la largeur et la hauteur de celui-ci sont affichées dans une info-bulle (exemple : 1 575 x 615). Ces dimensions sont aussi affichées, à droite de la barre d'outils Standard, avec la valeur des propriétés Left et Top (exemple : 840, 360).



En mode exécution, la modification des propriétés Left et Top déplace les objets à l'écran et la modification des propriétés Height et Width change la taille des objets.

En mode création, il peut arriver qu'un contrôle ne soit plus visible dans l'interface utilisateur. Pour le retrouver, sélectionnez-le dans la liste de sélection de la fenêtre Propriétés et fixez ses propriétés Top et Left à 0. Le contrôle apparaîtra dans le coin supérieur gauche de la feuille.

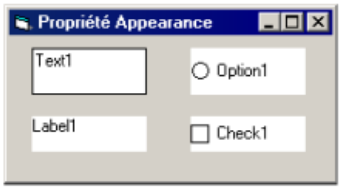
Exercice 1, partie 2 et Exercice 6.

Les propriétés *Alignment* et *Appearance*

La propriété **Alignment** permet d'aligner un texte dans un contrôle ou de modifier l'apparence des boutons d'option et des cases à cocher.

0-Left Justify	Alignement à gauche	vbLeftJustify
1-Right Justify	Alignement à droite	vbRightJustify
2-Center	Alignement au centre	vbCenter

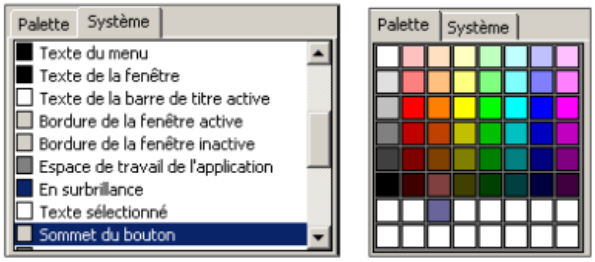
- La propriété **Appearance** permet de modifier le style de présentation d'un objet. Le style 1-3D (par défaut) a une apparence 3D et le style 0-Flat a une apparence 2D avec l'arrière-plan blanc.



Les propriétés *BackColor*, *ForeColor* et *Font*

- La propriété **BackColor** spécifie la *couleur de l'arrière-plan* d'un objet et la propriété **ForeColor** spécifie la *couleur de premier plan* des textes et éléments graphiques d'un objet.

Dans la fenêtre des propriétés, les couleurs peuvent être sélectionnées à partir de deux onglets. L'onglet **Système** contient les couleurs système aussi accessibles par l'icône Affichage/onglet Apparence du Panneau de configuration. L'onglet **Palette** contient les couleurs accessibles par le menu Affichage /Palette de couleurs.



Les couleurs sont représentées par des codes hexadécimaux. Exemple :

En mode programmation, les couleurs peuvent être modifiées comme suit :

- Par le code de couleur hexadécimal : `cmdBouton.BackColor = &H80000008 ;`

- Par une constante couleur : `cmdBouton.BackColor = vbBlack` ;
- Par une constante couleur système : `cmdBouton.BackColor = vbDesktop` ;
- Par la fonction RGB : cmdBouton.BackColor = RGB(122, 13, 7).

Pour utiliser une constante VB représentant une couleur, consultez l'Annexe A qui répertorie les constantes couleurs et les constantes couleurs système de Visual Basic.

La fonction RGB est décrite dans la section Autres fonctions, page 303.

En *mode création*, la propriété **Font** affiche la boîte de dialogue Police qui permet de choisir une police de caractères, un style et une taille.

En *mode programmation*, les valeurs **Bold**, **Charset**, **Italic**, **Name**, **Size**, **Strikethrough**, **Underline**, **Weight** de la propriété **Font** peuvent être modifiées comme suit :

```
txtZone.Font.Bold = True
txtZone.Font.Italic = False
txtZone.Font.Name = "Courier New"
txtZone.Font.Size = 18
```

Autres propriétés

FontName	Nom de la police de caractères.
FontSize	Nombre de points représentant la taille de la police de caractères.
FontBold	Style **Gras**.
FontItalic	Style *Italic*.
FontStrikethru	Style ~~Barré~~.
FontUnderline	Style Souligné.

Exercice 1, partie 2, Exercice2, partie 1 et Exercice 5.

La propriété *Caption*

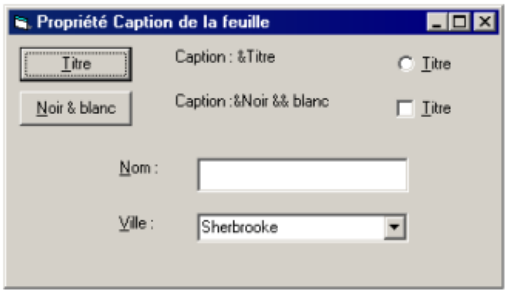
La propriété **Caption** permet d'attribuer un titre à un objet.

Dans le cas d'une feuille, le titre est celui qui apparaît dans la barre de titre et il est affiché sur la barre des tâches lorsque la feuille est réduite.

Dans le cas d'un contrôle, il est possible d'affecter un **raccourci clavier** au contrôle ou à celui qui le suit dans l'ordre de tabulation. Pour se faire, placez un caractère & en avant d'une lettre ou d'un chiffre qui sera souligné dans le titre. Le caractère souligné indique à l'utilisateur qu'il peut utiliser la touche **Alt** avec ce caractère pour placer le

focus sur ce contrôle ou l'activer. Pour qu'un caractère & apparaisse dans un titre, entrez deux caractères & consécutifs.

Exemple



Pour affecter un raccourci clavier à un bouton de commande, une case à cocher ou un bouton d'option, il suffit d'utiliser le caractère & dans sa propriété Caption.

Pour une zone de texte ou une zone de liste, il faut avoir recours à *une étiquette* généralement située à *gauche* ou *au-dessus* du contrôle à identifier. Le caractère & est placé dans la propriété Caption de l'étiquette et non dans celle du contrôle que l'utilisateur désire atteindre avec le raccourci clavier. Pour associer logiquement l'étiquette avec le contrôle à identifier, la propriété TabIndex de l'étiquette doit être égale à celle du contrôle – 1. Par exemple, la propriété TabIndex de l'étiquette Nom : a la valeur 5, et la propriété TabIndex de la zone de texte du nom, qui lui est associée, a la valeur 6.

Consultez la section Les propriétés *TabIndex* et *TabStop*, page 195.

Dans la fenêtre illustrée précédemment, le raccourci Alt+T est utilisé trois fois et le raccourci Alt+N est utilisé deux fois, ce qui causerait problème à un utilisateur. Dans la conception de vos interfaces utilisateur, soyez vigilants !

Les propriétés *Enabled* et *Visible*

- La propriété **Enabled** permet d'activer ou de désactiver un objet de manière qu'il soit accessible ou non à l'utilisateur. Dans le cas d'un objet conteneur, si celui-ci n'est pas accessible, tous les objets qu'il contient seront inaccessibles.

Exemple

```
fraCadre.Enabled = False
```

- La propriété **Visible** permet d'afficher ou de masquer un objet. Dans le cas d'un objet conteneur, si celui-ci n'est pas visible, tous les objets qu'il contient seront masqués.

Exemple `fraCadre.Visible = False`

Les propriétés *MousePointer* et *MouseIcon*

- La propriété **MouseIcon** est utilisée pour personnaliser l'aspect du pointeur de la souris. Un nom de fichier (.cur ou .ico) doit être spécifié et la propriété MousePointer doit avoir la valeur 99-Custom.

  Plusieurs styles de pointeurs sont fournis avec Visual Basic. Ils sont stockés dans le dossier :

  ...\Program Files \Microsoft Visual Studio \Common \Graphics \Cursors

- La propriété **MousePointer** détermine l'apparence du pointeur de la souris lorsque celui-ci atteint un objet. Les styles de pointeurs suivants sont disponibles dans la fenêtre des propriétés :

0-Default	Forme déterminée par l'objet (par défaut).
1-Arrow	Flèche blanche.
2-Cross	Croix.
3-I-Beam	Forme d'un I.
4-Icon	Icône (petit carré dans un autre carré).
5-Size	Flèche à 4 directions : Nord, Sud, Est, Ouest.
6-Size NE SW	Flèche à 2 directions : Nord-Est, Sud-Ouest.
7-Size N S	Flèche à 2 directions : Nord, Sud.
8-Size NW SE	Flèche à 2 directions : Nord-Ouest, Sud-Est.
9-Size WE	Flèche à 2 directions : Ouest, Est.
10-Up Arrow	Flèche vers le haut.
11-HourGlass	Sablier.
12-No Drop	Cercle avec une barre diagonale (zone interdite).
13-Arrow and Hourglass	Sablier avec une flèche.
14-Arrow and Question	Point d'interrogation avec une flèche.
15-Size All	Flèche à 4 pointes.
99-Custom	Icône personnalisée fixée par la propriété MouseIcon.
Une constante VB est disponible pour chaque style de pointeur. À partir de l'index de MSDN, consultez la rubrique MousePointer, Constantes.	

Exercice 16, partie 1

Les propriétés *TabIndex* et *TabStop*

- La propriété **TabIndex** détermine **l'ordre de tabulation** des contrôles d'une feuille, c'est-à-dire, l'ordre dans lequel les contrôles reçoivent le focus, lorsque la touche **Tab** est pressée. Par défaut, Visual Basic affecte la valeur 0 au premier contrôle, la valeur 1 au deuxième et ainsi de suite. Lorsqu'une valeur est modifiée dans la fenêtre Propriétés, les autres valeurs seront ajustées automatiquement. Notez que la combinaison de touches Maj+Tab déplace le focus en sens inverse de l'ordre de tabulation.
- La propriété **TabStop** détermine si un contrôle fait partie de l'ordre de tabulation ou non. En affectant la valeur False à la propriété TabStop d'un contrôle, le contrôle ne peut pas recevoir le focus avec la touche Tab. Notez que l'utilisateur peut quand même y accéder en cliquant à l'intérieur de celui-ci.

Lorsque tous les contrôles sont dessinés sur une feuille, exécutez le projet et vérifiez les déplacements avec la touche Tab ainsi que le fonctionnement des raccourcis clavier avec la touche Alt. Apportez des changements en ajustant les propriétés TabIndex et TabStop de vos contrôles.

Exemple pratique : Exercice 3

La propriété *Tag*

Contrairement aux autres propriétés, la propriété **Tag** n'est pas utilisée par Visual Basic. Elle est disponible au programmeur et celui-ci peut l'utiliser pour y conserver une valeur qu'il pourra éventuellement vérifiée.

Exemple

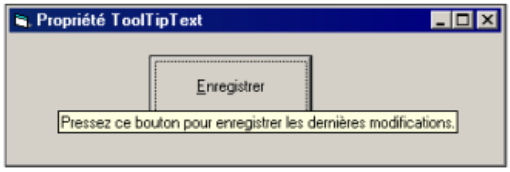
```
Private Sub cmdBouton_Click(Index As Integer)
  If cmdBouton(Index).Tag = 1 Then
    cmdBouton(Index).Tag = 2
    cmdBouton(Index).BackColor = vbMenuBar
  Else
    cmdBouton(Index).Tag = 1
    cmdBouton(Index).BackColor = vbWhite
  End If
End Sub
```

La propriété Tag est utilisée pour modifier la couleur d'un bouton lors d'un clic sur celui-ci. Un bouton gris devient blanc et un bouton blanc devient gris. Les valeurs 1 et 2 ont été affectées à la propriété Tag en mode création. Celles-ci sont vérifiées et modifiées en mode exécution.

Exercice 27

La propriété *ToolTipText*

La propriété **ToolTipText** permet d'afficher une info-bulle afin de renseigner l'utilisateur sur le contrôle où se trouve le pointeur de la souris.



Les propriétés Picture, DisabledPicture et DownPicture

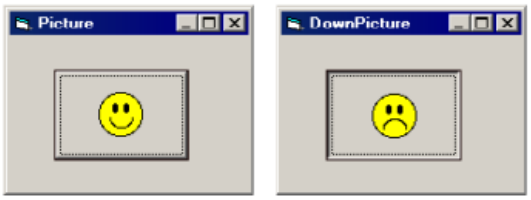
- La propriété **Picture** permet d'afficher une image dans un contrôle.
- La propriété **DisabledPicture** permet d'afficher une image dans un contrôle, lorsque celui-ci est désactivé (propriété Enabled = False).
- La propriété **DownPicture** permet d'afficher une image dans un contrôle, lorsque celui-ci est sélectionné.

Les propriétés DisabledPicture et DownPicture sont disponibles pour les contrôles CommandButton, OptionButton et CheckBox dont la propriété Style = 1-Graphical.

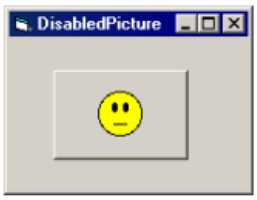
Exemple

Une image souriante a été affectée à la propriété Picture d'un bouton de commande et une image triste a été affectée à la propriété DownPicture. En cliquant plusieurs

fois sur le bouton, l'effet est sensiblement le même que dans le projet "Jean qui pleure et Jean qui rit".



Une image neutre a été affectée à la propriété DisabledPicture et la propriété Enabled du bouton de commande est False. Le bouton n'est donc pas accessible à l'utilisateur. La figure demeurera inchangée jusqu'à la rencontre d'une instruction cmdBouton.Enabled = True ; l'image souriante sera alors affichée.



4.5 LES MÉTHODES COMMUNES DES OBJETS

La méthode *SetFocus*

La méthode **SetFocus** permet de donner le **focus** a un objet (feuille, contrôle), celui-ci devenant ainsi la *feuille active* ou le *contrôle actif*. On reconnaît une feuille active à sa barre de titre en surbrillance. Un contrôle actif est identifié par différents signes. Par exemple, une zone de saisie active contient un curseur, un bouton de commande actif est encadré d'une ligne pointillée et une zone de liste modifiable est affichée en surbrillance. Le focus est donné à un objet par l'utilisateur (clic ou touche Tab) ou par la méthode SetFocus.

Syntaxe : *objet*.SetFocus

Exemple `txtNomFamille.SetFocus`

Pour recevoir le focus par la méthode SetFocus, un objet doit être accessible (Enabled = True) et visible (Visible = True).

La méthode SetFocus ne peut être invoquée dans un événement Form_Load puisque les contrôles ne sont pas encore dessinés. Pour donner le focus à un contrôle au démarrage, utilisez une procédure événementielle subséquente à Form_Load, comme par exemple, Form_Paint ou Form_Activate.

Exercice 3, Exercice 16, partie 3

La méthode *Move*

La méthode **Move** permet de modifier la position et la taille d'un objet en mode exécution. Elle modifie les propriétés Left, Top, Width et Height d'un objet.

Syntaxe: [*objet.*]**Move** *left* [, *top*[, *width*[, *height*]]]

L'argument *left* est le seul argument obligatoire. Cependant, pour utiliser les arguments suivants, vous devez citer tous les arguments qui apparaissent en avant de celui que vous voulez utiliser.

Exemple

```
Private Sub imgAvion_Click()
  Dim I As Integer
  imgAvion.Move 0, 0
  For I = 0 To 2000 Step 1
    imgAvion.Move I, I
  Next
End Sub
```

Au démarrage, l'avion est placé dans le coin inférieur gauche de la feuille. À la rencontre de la première méthode Move, celui-ci est déplacé dans le coin supérieur gauche de la feuille (Left = 0, Top = 0). Par la suite, l'avion vole en diagonale jusqu'au coin inférieur droit. Le premier déplacement est un *déplacement absolu* (valeurs fixes : 0, 0) et le second est un *déplacement relatif* (selon la valeur de la variable I).

En codant la méthode : `imgAvion.Move I, I, I, I` dans la boucle For, l'avion grossira graduellement en volant.

La méthode *Drag*

La méthode **Drag** commence, termine ou annule une opération glisser/déplacer d'un contrôle. Cette méthode s'utilise généralement lorsque la propriété DragMode du contrôle est fixée à 0-Manual. Elle n'est pas nécessaire dans le cas où la propriété DrragMode est fixée à 1-Automatic.

Syntaxe : *objet*.**Drag** [*code d'action*]

Trois codes d'action :

0	Annule l'opération glisser/déplacer.	vbCancel
1	Commence à faire glisser le contrôle. (par défaut)	vbBeginDrag
2	Arrête de faire glisser le contrôle et le dépose sur la feuille.	vbEndDrag

L'opération Glisser/Déplacer est traitée dans la section Les événements DragDrop et DragOver, page 204.

4.6 LES ÉVÉNEMENTS COMMUNS DES OBJETS

Les événements *Click* et *DblClick*

L'événement **Click** se produit lorsque l'utilisateur clique et relâche un bouton de la souris sur un objet. Il peut aussi se produire lorsque la valeur d'un contrôle est modifiée par programmation.

Sur une feuille, l'événement Click se produit lorsqu'un clic est effectué sur une zone vide de la feuille ou sur un contrôle non accessible (bouton droit ou gauche).

Sur un contrôle, l'événement Click est déclenché par le *bouton gauche et le bouton droit* de la souris, sauf sur les contrôles CheckBox, CommandButton, OptionButton, ComboBox et ListBox, où l'événement est déclenché seulement avec le bouton gauche de la souris.

L'utilisateur déclenche l'événement dans les cas suivants :

- Lors d'une sélection d'un élément dans un contrôle ComboBox ou ListBox (flèches ou clic) ;
- Lorsque la touche Espace est pressée sur un contrôle CommandButton, OptionButton et CheckBox qui a le focus ;

- Lorsque la touche Entrée est pressée sur un bouton de commande par défaut ou que la touche Échap est pressée et qu'un bouton d'annulation a été défini ;
- Lorsqu'un raccourci clavier est utilisé sur un bouton de commande.

L'événement peut être déclenché par programmation :

- En affectant la valeur True à la propriété Value des boutons de commande et des boutons d'option ou en modifiant la valeur de la propriété Value d'une case à cocher.

Exemple

```
cmdBouton.Value = True
optBouton.Value = True
chkCase.Value = vbChecked
chkCase.Value = vbUnChecked
```

- L'exemple qui suit provoque une **boucle d'événements** puisque la procédure est appelée indéfiniment :

```
Private Sub cmdBouton_Click()
  cmdBouton.Value = True
End Sub
```

Les boucles d'événements sont traitées dans la section L'événement *Change*, page 201.

Syntaxe de la procédure événementielle Click

Private Sub Form_Click( )

Private Sub *objet*_**Click(**[*index* **As Integer**]**)**

Dans le cas d'une feuille, aucun argument ne doit être spécifié et dans le cas d'un contrôle, l'argument *index* spécifie que l'objet fait partie d'un groupe de contrôles.

Lorsque les événements Click et DblClick sont codés pour le même objet, l'événement DblClick n'est pas déclenché, parce que l'événement Click est le premier à se produire.

Exercice 1

L'événement **DblClick** se produit lorsque l'utilisateur clique rapidement deux fois de suite sur un objet.

Sur une feuille, l'événement DblClick se produit lorsqu'un double-clic est effectué sur une zone vide de la feuille ou sur un contrôle non accessible (bouton droit ou gauche).

Sur un contrôle, l'événement est déclenché par un double-clic sur un élément contenu dans un contrôle ComboBox (style = 1-Simple Combo), FileListBox, ListBox, DataCombo ou DataList.

Syntaxe de la procédure événementielle DblClick

Private Sub Form_DblClick( )

Private Sub *objet*_**DblClick(**[*index* **As Integer**]**)**

Dans le cas d'une feuille, aucun argument ne doit être spécifié et dans le cas d'un contrôle, l'argument *index* spécifie que l'objet fait partie d'un groupe de contrôles.

Exercice 18

L'événement *Change*

L'événement **Change** se produit lorsque le contenu d'un contrôle est modifié par l'utilisateur ou par programmation.

Spécifications selon le contrôle

TextBox	L'événement se produit lorsque l'utilisateur modifie le contenu d'une zone de texte, aussitôt qu'un nouveau caractère est entré ou effacé. L'événement est aussi déclenché lorsque la propriété **Text** est modifiée par programmation.
ComboBox	L'événement se produit si celui-ci contient une zone de texte (propriété **Text**) et que le contenu de cette zone de texte est modifié par l'utilisateur ou par programmation.
Label	L'événement se produit lorsque la propriété Caption est modifiée par programmation.
DirListBox	L'événement se produit lors d'un double-clic sur un dossier dans la liste ou lorsque la propriété **Path** de la liste est modifiée par programmation.
DriveListBox	L'événement se produit lorsque l'utilisateur sélectionne un lecteur dans la liste ou lorsque la propriété **Drive** de la liste est modifiée par programmation.
HScrollBar **VScrollBar**	L'événement se produit lorsque l'utilisateur utilise le curseur de défilement ou les pointes de flèches ou lorsque la propriété **Value** est modifiée par programmation.
PictureBox	L'événement se produit lorsque la propriété **Picture** est modifiée par programmation.

Syntaxe de la procédure événementielle Change

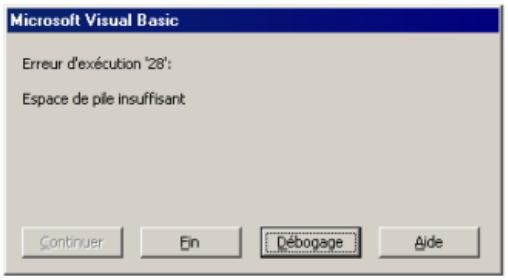
Private Sub *objet*_**Change (**[*index* **As Integer**]**)**

L'argument *index* spécifie que l'objet fait partie d'un groupe de contrôles.

Exercice 8, Exercice 18, Exercice 28.

Boucle d'événements

Une **boucle d'événements** se produit lorsqu'une procédure s'appelle elle-même ou lorsqu'une procédure s'appelle indirectement. Dans ce cas, l'espace mémoire où tous les appels sont enregistrés devient insuffisant. Le message qui suit est alors affiché :



Exemple d'une procédure qui s'appelle elle-même

```
Private Sub txtZone_Change()
  Call txtZone_Change
End Sub
```

La procédure Change est appelée indéfiniment.

Exemple d'une procédure appelée indirectement

```
Private Sub txtZone_Change()
    txtZone.Text = ""
   txtZone.Text = "Contenu"
End Sub
```

La première instruction efface la zone de texte et la seconde lui affecte une chaîne de caractères modifiant ainsi sa valeur. Aussitôt que la valeur de la zone de texte change, la procédure `txtZone_Change` est appelée de nouveau. Elle sera ainsi appelée indéfiniment.

Lorsqu'une erreur d'exécution "Espace de pile insuffisant" se produit, pressez le bouton Débogage et affichez la fenêtre "Pile des appels" par le menu Affichage /Pile des appels ou les touches Ctrl+L.

Les événements *GotFocus* et *LostFocus*

L'événement **GotFocus** se produit lorsqu'un objet reçoit le focus et l'événement **LostFocus** se produit lorsqu'un objet perd le focus. Une feuille peut recevoir le focus lorsque tous les contrôles qu'elle contient sont inaccessibles.

Syntaxe des procédures événementielles GotFocus et LostFocus

Private Sub Form_XxxxFocus( )

Private Sub *objet*_**XxxxFocus(**[*index* **As Integer**]**)**

L'argument *index* spécifie que l'objet fait partie d'un groupe de contrôles.

Dans le cas d'une zone de saisie, l'événement LostFocus est un moment approprié pour coder des instructions de validation et afficher des messages en cas de données invalides.

Exercice 13.

Les événements *MouseDown*, *MouseUp* et *MouseMove*

- L'événement **MouseDown** se produit lorsque l'utilisateur enfonce un bouton de la souris.
- L'événement **MouseUp** se produit lorsque l'utilisateur relâche un bouton de la souris.
- L'événement **MouseMove** se produit pour un objet lorsque le pointeur de la souris se déplace sur lui.

Syntaxe des procédures événementielles

Private Sub Form_MouseXxxx (*bouton* **As Integer, _**

shift **As Integer**, *x* **As Single**, *y* **As Single**)

Private Sub *object*_**MouseXxxx (**[*index* **As Integer,**] _

bouton **As Integer,** *shift* **As Integer,** *x* **As Single,** *y* **As Single)**

L'argument *index* spécifie que l'objet fait partie d'un groupe de contrôles.

L'argument *bouton* identifie le bouton de la souris.

1	Bouton gauche	vbLeftButton
2	Bouton droit	vbRightButton
4	Bouton central	vbMiddleButton

L'argument *shift* spécifie si une ou plusieurs des touches Maj, Ctrl et Alt sont enfoncées.

0	Aucune touche enfoncée.	
1	La touche Maj est enfoncée.	vbShiftMask
2	La touche Ctrl est enfoncée.	vbCtrlMask
4	La touche Alt est enfoncée.	vbAltMask
Dans le cas de plusieurs touches enfoncées, utilisez la somme des valeurs de chaque touche. Exemple : Pour les touches Ctrl+Alt, utilisez la valeur 6 et pour les 3 touches, utilisez 7.		

Les arguments *x* et *y* retourne la position du pointeur de la souris.

Les événements DragDrop et DragOver (DragMode, DragIcon)

- L'événement **DragDrop** se produit à la fin d'une opération glisser/déplacer parce l'utilisateur a fait glisser un contrôle sur un objet et a relâché le bouton de la souris. L'événement est aussi déclenché lorsque la méthode Drag affecte la valeur 2 (vbEndDrag) à un contrôle en déplacement.
- L'événement **DragOver** se produit lorsqu'une opération glisser/déplacer est en cours. Cet événement est approprié pour contrôler le pointeur de la souris lorsqu'il atteint ou quitte la cible ou encore lorsqu'il s'immobilise dessus.
- La **propriété DragMode** d'un contrôle spécifie si une opération glisser/déplacer est effectuée en mode manuel ou automatique.

0-Manual (vbManual)	Le mode manuel (par défaut) requiert que la *méthode Drag* (réf. La méthode *Drag*) soit invoquée pour débuter une opération glisser/déplacer sur le contrôle à déplacer.
1-Automatic (vbAutomatic)	En mode automatique, une opération glisser/déplacer débute automatiquement par un clic sur le contrôle à déplacer.

- La **propriété DragIcon** spécifie une icône à afficher comme pointeur de souris lors d'une opération glisser/déplacer. Le fichier icône peut être spécifié dans la fenêtre Propriétés ou par programmation.

Exemple : L'icône d'un autre objet peut être utilisée ou la fonction LoadPicture.

```
imgGlisse.DragIcon = txtZone.Icon
imgGlisse.DragIcon = LoadPicture(…)
```

Syntaxe de la procédure événementielle DragDrop

Private Sub Form_DragDrop(*source* **As Control**, _

x **As Single**, *y* **As Single)**

Private Sub *objet*_**DragDrop(**[*index* **As Integer,**] _

source **As Control,** *x* **As Single,** *y* **As Single)**

L'argument *index* spécifie que l'objet fait partie d'un groupe de contrôles.

L'argument *source* identifie l'objet en déplacement. Le nom de l'argument peut être utilisé dans le code à la place du nom de l'objet (exemple : `source.Visible = False`).

Les arguments *x* et *y* retourne les positions horizontale et verticale du pointeur de la souris.

Syntaxe de la procédure événementielle DragOver

Private Sub Form_DragOver(*source* **As Control**, _

x **As Single**, *y* **As Single,** *state* **As Integer)**

Private Sub *objet*_**DragOver(**[*index* **As Integer,**] _

source **As Control,** *x* **As Single,** *y* **As Single**, *state* **As Integer)**

L'argument *index* spécifie que l'objet fait partie d'un groupe de contrôles.

L'argument *source* identifie l'objet en déplacement.

Les arguments *x* et *y* retourne les positions horizontale et verticale du pointeur de la souris.

L'argument *state* retourne un entier indiquant la position du contrôle en déplacement.

0	Le contrôle source entre dans la cible.
1	Le contrôle source sort de la cible.
2	Le contrôle est dans la cible.

Exercice 16

L'événement *Scroll*

L'événement **Scroll** se produit lorsque l'utilisateur utilise une barre de défilement indépendante (exemple : VScrollBar, HScrollBar, FlatScrollBar, Slider) ou une barre de défilement intégrée à un objet par sa propriété ScrollBars (exemple : DataGrid, ComboBox).

Syntaxe de la procédure événementielle Scroll

Private Sub *objet*_**Scroll( )**

Exercice 28

Les événements clavier

Les événements clavier **KeyPress**, **KeyDown** et **KeyUp** se produisent lorsque l'utilisateur presse ou relâche une touche du clavier. Ils permettent au programmeur d'intercepter les touches, dans le but de les vérifier, de les refuser ou encore de les modifier. Ces trois événements peuvent être codés pour un contrôle ou pour une feuille. Dans le cas d'une feuille, l'événement est déclenché pour l'ensemble de ses contrôles et la propriété **KeyPreview** de la feuille doit avoir la valeur **True**, afin que les événements clavier de la feuille soient reconnus **avant** ceux de chacun des contrôles qu'elle contient.

Les trois événements *ne sont pas déclenchés* dans les cas suivants :

- Lorsque la touche Tab est pressée ;
- Lorsque la touche Entrée est pressée et qu'un *bouton par défaut* a été défini (Default = True) ;
- Lorsque la touche Échap est pressée et qu'un *bouton d'annulation* a été défini (Cancel = True).

L'événement KeyPress

L'événement **KeyPress** se produit lorsqu'une touche ANSI (American National Standards Institute) a été pressée et relâchée. Chaque touche du clavier correspond à un code numérique variant de 0 à 255.

Consultez l'Annexe B Tableau des codes ANSI.

Syntaxe de la procédure événementielle KeyPress

Private Sub Form_KeyPress(*KeyAscii* **As Integer)**

Private Sub *objet*_**KeyPress([***Index* **As Integer**,] *KeyAscii* **As Integer)**

L'argument *KeyAscii* reçoit une valeur de 0 à 255 représentant le code de la touche pressée.

L'affectation *KeyAscii = 0* annule la touche qui a été pressée, la valeur 0 représentant une chaîne de caractères nulle ("").

***Exemple 1* : Validation numérique**

```
Private Sub txtNombre_KeyPress(KeyAscii As Integer)
  Select Case KeyAscii
    Case vbKey0 To vbKey9, Asc(","), Asc("-")
    Case Asc(".")
      KeyAscii = Asc(",")  'point changé en virgule
    Case Else
      KeyAscii = 0  'tous les autres touches sont refusées
  End Select
End Sub
```

Seulement les nombres, la virgule et le signe moins (-) sont acceptés. Si l'utilisateur presse un point, il sera converti en virgule.

Pour vérifier la touche pressée, les constantes vbKey0 et vbKey9 sont utilisées. Si une constante n'est pas disponible pour représenter un caractère, la fonction **Asc** est employée afin d'obtenir le code (0 à 255) du caractère spécifié. Il est aussi possible d'utiliser la fonction **Chr** afin de convertir l'argument *KeyAscii* en caractère. Par exemple :

```
Private Sub txtNombre_KeyPress(KeyAscii As Integer)
  Select Case Chr(KeyAscii)
    Case 0 To 9, ",", "-"
    Case "."
      KeyAscii = Asc(",")
    Case Else
      KeyAscii = 0
  End Select
End Sub
```

Les codes ANSI peuvent aussi être utilisés, mais sont peu efficaces pour une compréhension rapide du code de programmation. Par exemple :

```
Private Sub txtNombre_KeyPress(KeyAscii As Integer)
  Select Case KeyAscii
    Case 48 To 57, 44, 45
    Case 46
      KeyAscii = 44 'point changé en virgule
    Case Else
      KeyAscii = 0  'tous les autres touches sont refusées
  End Select
End Sub
```

Les fonctions Chr et Asc sont décrites dans la section Les fonctions de chaînes de caractères, page 287.

***Exemple 2* : Validation alphabétique**

```
Private Sub txtNom_KeyPress(KeyAscii As Integer)
  Select Case KeyAscii
    Case vbKeyA To vbKeyZ
    Case Asc("a") To Asc("z")
    Case Asc("-"), vbKeySpace
    Case vbKeyBack
    Case vbKeyEscape
      txtNom.Text = ""
    Case Else
      KeyAscii = 0
  End Select
End Sub
```

Seulement les touches suivantes sont acceptées : les lettres majuscules, les lettres minuscules, le tiret, le caractère d'espacement (vbKeySpace), la touche Retour Arrière (vbKeyBack) et la touche Échap (vbKeyEscape). Lorsque l'utilisateur presse la touche Échap, la zone de texte est complètement effacée.

Lorsqu'un événement KeyPress est programmé, la touche *Retour Arrière* n'est plus fonctionnelle. Il faut donc en permettre l'accès.

Notez que dans l'exemple précédent, les *caractères accentués* ne sont pas acceptés (exemple : à, É, è, ...).

Exercice 8, Exercice 10, Exercice 11

Les événements KeyDown et KeyUp

L'événement **KeyDown** se produit lorsqu'une touche est pressée et l'événement **KeyUp** se produit lorsqu'une touche est relâchée.

Syntaxe des procédures événementielles KeyDown et KeyUp

Private Sub Form_KeyXxxx(*Keycode* **As Integer,** *Shift* **As Integer)**

Private Sub *objet*_KeyXxxx ([*Index* **As Integer,**] *Keycode* **As _**

Integer, *Shift* **As Integer)**

L'argument *KeyCode* reçoit la valeur d'une touche qui correspond à une constante Visual Basic. Les constantes font partie de la classe KeyCodeConstants (exemple : vbKeyF1).

Consultez l'Annexe A.1 Les constantes des codes de touches.

L'argument *Shift* spécifie si une ou plusieurs des touches Maj, Ctrl et Alt sont enfoncées.

0	Aucune touche enfoncée.	
1	La touche Maj est enfoncée.	vbShiftMask
2	La touche Ctrl est enfoncée.	vbCtrlMask
4	La touche Alt est enfoncée.	vbAltMask
Dans le cas de plusieurs touches enfoncées, utilisez la somme des valeurs de chaque touche. Exemple : Pour les touches Ctrl+Maj, utilisez la valeur 3 et pour les 3 touches, utilisez la valeur 7.		

Les événements KeyDown et KeyUp permettent de vérifier :

- Les touches de fonction (F1 à F12) ;
- Les touches de navigation (exemple : flèche à droite →) ;
- La distinction entre un nombre du clavier standard et un nombre du clavier numérique ;
- Les combinaisons de touches avec Maj, Ctrl et Alt ;
- La distinction entre une minuscule et une majuscule.

Exemple

```
Private Sub txtZone_KeyDown(KeyCode As Integer, _
                                Shift As Integer)
  Select Case KeyCode
    Case vbKeyPageUp
      Debug.Print "Touche Page précédente"
    Case vbKeyRight
      Debug.Print "Touche Flèche droite"
    Case vbKey5
      Debug.Print "Touche 5 du clavier standard"
    Case vbKeyNumpad5
      Debug.Print "Touche 5 du clavier numérique"
    Case Shift = 1 And vbKeyA
      Debug.Print "A majuscule"
```

```
    Case vbKeyA
      Debug.Print "a minuscule"
    'Case "j"
      'Erreur 13 : type incompatible
    Case vbKeyF2
      Debug.Print "Touche F2"
    Case vbKeyF3
      Select Case Shift
        Case 7
          Debug.Print "Touches Shift + Ctrl + Alt + F3"
        Case 6
          Debug.Print "Touches Ctrl + Alt + F3"
        Case 5
          Debug.Print "Touches Shift + Alt + F3"
        Case 4
          Debug.Print "Touches Alt + F3"
        Case 3
          Debug.Print "Touches Shift + Ctrl + F3"
        Case 2
          Debug.Print "Touches Ctrl + F3"
        Case 1
          Debug.Print "Touches Shift + F3"
        Case 0
          Debug.Print "Touche F3"
      End Select
    End Select
End Sub
```

Lorsqu'une touche est pressée dans la zone de texte txtZone, l'événement KeyDown est déclenché et la touche pressée est vérifiée ainsi que les touches Maj, Ctrl et Alt.

La méthode **Print** de l'objet **Debug** affiche une chaîne de caractères dans la fenêtre Exécution. L'objet Debug est un moyen très pratique de tester certaines parties de code.

L'instruction *SendKeys*

L'instruction **SendKeys** envoie une ou plusieurs touches à la fenêtre active comme si les touches avaient été directement saisies du clavier. Pour utiliser cette instruction, la propriété **KeyPreview** de la feuille doit obligatoirement avoir la valeur **True**.

Syntaxe : **SendKeys** *string*

L'argument *string* est une chaîne de caractères qui contient les touches envoyées.

Exemple

```
SendKeys "A"
SendKeys "ABC"
```

Certaines touches doivent être encadrées d'accolades. Par exemple, la touche {Tab}, la touche {Enter}, la touche Retour Arrière {BS} et la touche Échap {Esc}.

Consultez dans l'index de l'aide MSDN, la rubrique SendKeys, instruction. Vous y trouverez un tableau de toutes les touches du clavier.

Exemple

```
Private Sub txtZone_KeyPress(KeyAscii As Integer)
  If KeyAscii = vbKeyReturn Then
    SendKeys "{tab}"
    KeyAscii = 0
  End If
End Sub
```

Si l'utilisateur presse la touche Entrée, l'instruction SendKeys envoie la touche Tab, et la touche Entrée est annulée (KeyAscii = 0).

Exercice 27

Créer un gestionnaire de clavier au niveau d'une feuille

Un gestionnaire de clavier au niveau d'une feuille, permet de créer une procédure événementielle *commune* pour l'ensemble des contrôles d'une feuille. Les événements du clavier, Form_KeyPress, Form_KeyDown ou Form_KeyUp, sont déclenchés pour toutes les zones de saisie d'une feuille.

Afin que l'événement de niveau feuille (exemple : Form_KeyDown) soit déclenché *avant* celui d'un contrôle (exemple : txtNom_KeyDown), fixez la valeur **True** à la propriété **KeyPreview** de la feuille, en mode création ou en mode programmation.

Exemple 1 : La touche Entrée doit être équivalente à la touche Tab pour toutes les zones de saisie de la feuille.

```
Private Sub Form_Load()
  KeyPreview = True
End Sub

Private Sub Form_KeyPress(KeyAscii As Integer)
  If KeyAscii = vbKeyReturn Then
    SendKeys "{tab}"
    KeyAscii = 0
  End If
End Sub
```

Exemple 2 : La touche Échap permet d'effacer le contenu d'une zone de texte quand celle-ci est active.

```
Private Sub Form_KeyPress(KeyAscii As Integer)
  If TypeOf Me.ActiveControl Is TextBox Then
    If KeyAscii = vbKeyEscape Then
      Me.ActiveControl.Text = ""
    End If
  End If
End Sub
```

Exemple 3 : Le raccourci clavier Alt + x ou Alt + X permet d'effacer le contenu de toutes les zones de texte de la feuille, en une seule opération.

```
Private Sub Form_KeyDown(KeyCode As Integer, _
                                 Shift As Integer)
  Dim Controle As Control
  If KeyCode = vbKeyX And Shift = 4 Then
    For Each Controle In Me.Controls
      If TypeOf Controle Is TextBox Then
        Controle.Text = ""
      End If
    Next
  End If
End Sub
```

Exercice 27

L'événement *Validate*

L'événement **Validate** se produit avant que le focus passe au contrôle suivant, dont la propriété **CausesValidation** a pour valeur **True**. Cet événement est déclenché juste avant l'événement LostFocus.

L'événement Validate est utile pour s'assurer de la validité d'une donnée que l'utilisateur vient juste d'entrer dans une zone de saisie. La donnée est vérifiée juste avant que l'utilisateur atteigne un autre contrôle par la touche Tab ou par un clic de souris.

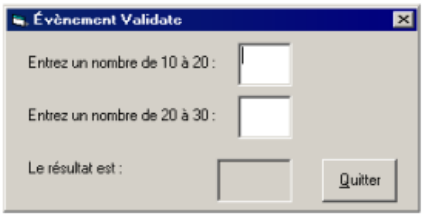
Syntaxe de la procédure événementielle Validate

Private Sub *objet*_**Validate(***Cancel* **As Boolean)**

L'argument *Cancel* détermine si le contrôle perd le focus ou non. En lui affectant la valeur True, le contrôle est redonné à la zone, empêchant ainsi l'utilisateur d'atteindre un autre contrôle.

Par défaut, la propriété CausesValidation de tous les contrôles a la valeur True. Si celle-ci n'est pas modifiée, l'événement Validate est déclenché sur l'ensemble des contrôles, en autant que l'utilisateur tente de donner le focus à un autre contrôle. Par exemple, s'il clique sur un choix au menu ou sur le bouton Fermer de la barre de titre, l'événement ne sera pas déclenché.

Exemple




```
Private Sub txtNb1_Validate(Cancel As Boolean)
  If Val(txtNb1) < 10 Or Val(txtNb1) > 20 Then
    MsgBox "Le nombre 1 est invalide", , "Erreur"
    Cancel = True
  End If
End Sub

Private Sub txtNb2_Validate(Cancel As Boolean)
  If Val(txtNb1) < 20 Or Val(txtNb1) > 30 Then
    MsgBox "Le nombre 2 est invalide", , "Erreur"
    Cancel = True
  End If
End Sub
```

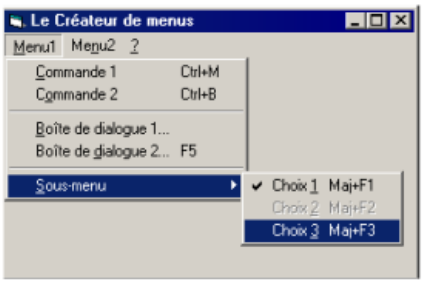
Lorsqu'un nombre est erroné, un message est affiché et Cancel = True redonne le focus à la zone en erreur. L'utilisateur doit **obligatoirement** entrer une donnée valide avant de pouvoir passer à un autre contrôle (txtNb2 ou cmdQuitter). Par exemple, si l'utilisateur clique sur le bouton Quitter, le message est affiché et l'utilisateur ne peut pas quitter sans entrer un nombre valide... Toutefois, il peut utiliser le bouton Fermer de la barre de titre.

Utilisez l'événement Validate lorsqu'une donnée valide est absolument nécessaire à un moment précis. Une autre méthode de validation consiste à valider les données en bloc, après que l'utilisateur ait pressé un bouton "Soumettre les données" ou "Enregistrer les données".

Exercice 13

4.7 LES MENUS

Les menus permettent de regrouper un ensemble de commandes accessibles à l'utilisateur sans encombrer l'interface utilisateur. Dans une feuille, la **barre de menus** est affichée sous la barre de titre. Elle contient un ou plusieurs menus identifiés par un titre (exemple : Menu1, Menu2 et ?).



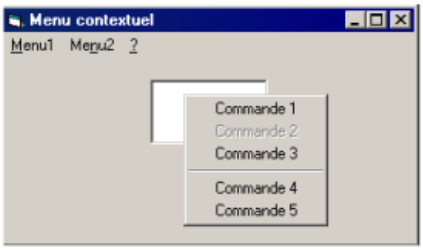
Un menu peut contenir :

- Des titres de *commandes* (Commande 1);
- Des lignes séparatrices ;
- Des titres de *sous-menus* suivis d'une pointe de flèche ► indiquant à l'utilisateur la présence d'un sous-menu ;
- Des *points de suspension* (...) qui signalent qu'une *boîte de dialogue* sera affichée à la suite de ce choix (exemple : Enregistrer sous...) ;
- Des choix *non accessibles* (exemple : Choix 2) ;
- Des choix avec une *coche* (exemple : Choix 1) ;
- Des *touches de raccourci* (exemple : Ctrl+M, F5) ;
- Des *lettres soulignées* dans les titres pour un accès avec la touche Alt (exemple : Alt+M donne accès à Menu1).

En plus de la barre de menus, des **menus contextuels** permettent de regrouper un ensemble de commandes spécifiques à un objet. Un menu contextuel est affiché lorsque le pointeur de la souris se trouve sur un objet et que l'utilisateur clique avec le bouton droit de la souris.

Certains contrôles, comme le contrôle TextBox, possèdent déjà un menu contextuel. Dans ce cas, un second menu contextuel peut être affiché en cliquant de nouveau avec

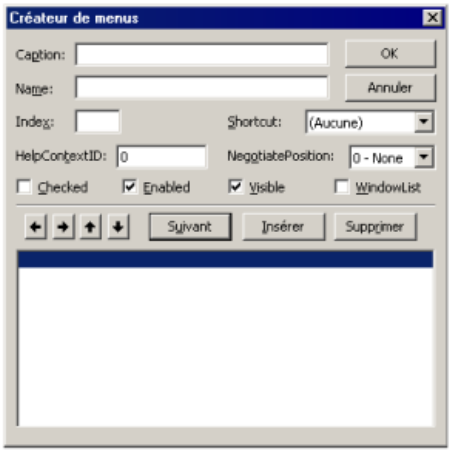
le bouton gauche de la souris. Un menu contextuel est affiché avec la méthode **PopupMenu**.



La barre de menus ainsi que les menus contextuels sont créés à l'aide du **Créateur de menus** qui permet d'ajouter des contrôles **Menu** à un projet. Chacun des choix de menu définis est une nouvelle instance de la *classe d'objet Menu* ayant ses propres caractéristiques (exemple : Name, Caption, Enabled) et ses propres comportements.

Le Créateur de menus

Le **Créateur de menus** est accessible par le menu **Outils /Créateur de menus**, le raccourci clavier **Ctrl+E** ou par l'icône "Créateur de menus" de la barre d'outils Standard. La boîte de dialogue "Créateur de menus" permet de définir rapidement les différents objets d'un menu. Une fois que le menu est créé, les propriétés de chaque objet peuvent être modifiées dans la fenêtre Propriétés. Le Créateur de menus est accessible seulement lorsque la feuille de conception de feuille est activée.



Les propriétés Name et Caption

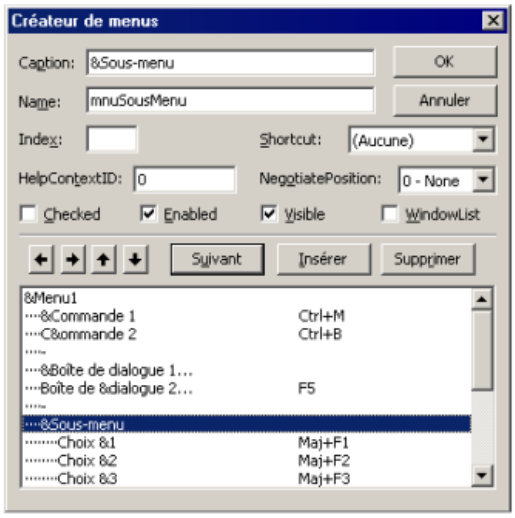
Les propriétés **Name** et **Caption** sont *obligatoires* pour chacun des éléments d'un menu.

- La propriété **Name** définit le nom du contrôle Menu. Le préfixe **mnu** identifie le contrôle Menu.
- La propriété **Caption** représente le titre d'un élément de menu. Utilisez le caractère & en avant d'un caractère pour souligner celui-ci et ainsi définir un raccourci clavier avec la touche Alt. Pour définir une barre séparatrice, entrez un *seul tiret* (-).

Pour ajouter un nouvel élément de menu, fixez les propriétés Name et Caption et terminez l'ajout en pressant le bouton **Suivant** pour ajouter un autre élément. Lorsque le dernier élément est créé, pressez le bouton **OK** pour fermer la fenêtre "Créateur de menus".

La zone de liste des éléments d'un menu

Dans le bas de la fenêtre Créateur de menus, une zone de liste affiche les éléments du menu qui sont définis.



- Un élément de menu qui apparaît à l'extrémité gauche de la zone de liste est un *titre de menu* dans la barre de menus (exemple : &Menu1).

- Un élément de menu en *retrait*, c'est-à-dire précédé de 4 petits points (….), s'affiche dans le menu lorsque l'utilisateur clique sur le titre de menu précédent. (exemple : ….&Commande 1, ….&Sous-Menu).
- Un élément de menu en *double retrait* (……..) est un élément d'un sous-menu. (exemple : ……..Choix &1).

Évitez les sous-menus et les *sous-sous-menus* qui imposent à l'utilisateur de sélectionner plusieurs choix avant d'atteindre la commande désirée. Dans plusieurs cas, l'utilisation d'une boîte de dialogue se révèle plus pratique puisque l'utilisateur peut sélectionner plusieurs choix dans une seule fenêtre.

- Un *tiret* (-) représente une *barre séparatrice* qui sert à diviser les éléments d'un menu en groupes logiques. Notez que la propriété Name d'une barre séparatrice est aussi obligatoire (exemple : ….-).

La zone de liste comporte des boutons de commande :

La flèche **Droite** met *en retrait* un élément du menu et la flèche **Gauche** supprime un retrait.
Les flèches **Haut** et **Bas** permettent de se déplacer dans la zone de liste.

Le bouton **Suivant** sélectionne l'élément suivant dans la liste.

Le bouton **Insérer** permet d'ajouter un nouvel élément au menu. Il insère une ligne dans la zone de liste *au-dessus* de celle sélectionnée.

Le bouton **Supprimer** supprime l'élément sélectionné dans la zone de liste.

Quelques autres propriétés

- La propriété **ShortCut** permet d'affecter un *raccourci clavier* à une commande d'un menu. Un raccourci clavier permet à l'utilisateur de sélectionner une commande en une seule opération. Une zone de liste affiche toutes les combinaisons de touches qui sont permises.
- La propriété **Checked** ajoute ou non une coche en avant d'une commande. Une coche indique à l'utilisateur que l'élément coché est actuellement activé. Cette propriété est modifiable par programmation.

Exemple `mnuChoix1.Checked = False`

- La propriété **Enabled** active ou désactive un élément du menu. Le texte d'un élément désactivé est affiché en grisé. Cette propriété est modifiable par programmation.

Exemple

```
mnuChoix1.Enabled = True
```

- La propriété **Index** permet de créer des groupes de contrôles, comme tous les autres contrôles. Pour créer un groupe de contrôles avec certains éléments d'un menu, il suffit de leur attribuer le même nom (propriété Name).

Exemple

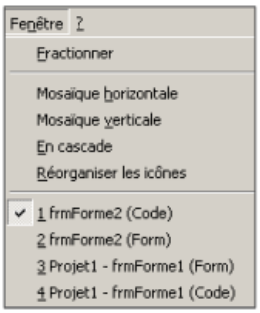
Name	Caption	Index
mnuChoix	Choix &1	0
mnuChoix	Choix &2	1
mnuChoix	Choix &3	2

```
Private Sub mnuChoix_Click(Index As Integer)
  instructions communes pour les 3 choix
End Sub
```

- La propriété **Visible** spécifie si oui ou non l'élément du menu est visible. Lors de la création d'un *menu contextuel*, affectez la valeur **False** à cette propriété pour qu'il ne soit pas affiché dans la *barre de menus*.

- La propriété **WindowList** est spécifique aux feuilles de type MDI. Elle détermine si oui ou non, une liste des feuilles MDI filles ouvertes est affichée. Par exemple, dans l'environnement de développement de Visual Basic, une liste des fenêtres ouvertes est affichée dans le menu Fenêtre.

Exemple

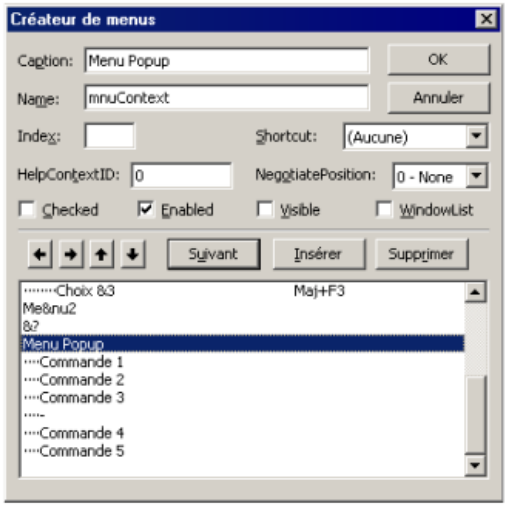


Le contrôle Menu ne réagit qu'à l'événement **Click**. Une procédure événementielle Click doit être codée pour chacune des commandes du menu pour que celles-ci soient exécutables.

L'exercice 11 vise à créer un premier menu, l'exercice 12 permet de créer un second menu et l'exercice 22 démontre comment synchroniser les éléments d'une barre de menus avec ceux d'un menu contextuel et les icônes correspondantes sur une barre d'outils.

Les menus contextuels (méthode *PopupMenu*)

De façon générale, un **menu contextuel** est créé à l'aide du Créateur de menu et la valeur False est affectée à la propriété Visible du titre du menu. Par exemple, dans la fenêtre qui suit, le titre du menu est Menu Popup, son nom est mnuContext et la case Visible n'est pas cochée.



La méthode PopupMenu

La méthode **PopupMenu** permet d'afficher un menu contextuel.

Syntaxe : *objet*.**PopupMenu** *menuname*[, *flags*][, *x*, *y*][, *boldcommand*]

L'argument *menuname* est le nom du menu contextuel à afficher (exemple : mnuContext).

L'argument *flags* précise l'emplacement et les boutons de sélection du menu contextuel.

Emplacement	
vbPopupMenuLeftAlign	Le pointeur de la souris se trouve à gauche du menu contextuel (par défaut).
vbPopupMenuCenterAlign	Le menu contextuel est centré sur le pointeur de la souris.
vbPopupMenuRightAlign	Le pointeur de la souris se trouve à droite du menu contextuel.
Boutons de sélection de la souris	
vbPopupMenuLeftButton	L'utilisateur ne peut utiliser que le bouton gauche pour sélectionner un élément du menu contextuel (par défaut).
vbPopupMenuRightButton	L'utilisateur peut sélectionner un élément du menu contextuel avec le bouton droit et le bouton gauche de la souris.

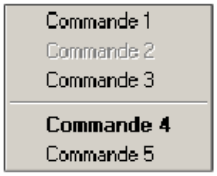
Les arguments *x* et *y* permettent de désigner un endroit précis dans l'écran où le menu contextuel doit être affiché, au lieu d'afficher le menu selon la position du pointeur de la souris.

L'argument *boldcommand* permet d'afficher l'un des éléments du menu en gras. Le nom de cet élément doit être spécifié.

Exemple

```
Private Sub lblEtiq_MouseDown(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
  If Button = vbRightButton Then
    PopupMenu mnuContext, vbPopupMenuRightButton + _
                          vbPopupMenuCenterAlign, , , _
                          mnuCommande4
  End If
End Sub
```

Lorsque que le pointeur de la souris est placé sur une étiquette `lblEtiq` et que l'utilisateur enfonce l'un des boutons de la souris, la procédure événementielle `lblEtiq_MouseDown` est exécutée. Si le bouton enfoncé est le bouton droit de la souris (`Button = vbRightButton`), le menu contextuel `mnuContext` est affiché. Il est centré sur le pointeur de la souris (`vbPopupMenuCenterAlign`). L'utilisateur peut sélectionner une commande du menu avec le bouton gauche et le bouton droit de la souris (`vbPopupMenuRightButton`). La Commande 4 est affichée en gras dans le menu.



Exercice 17

4.8 LES MASQUES DE SAISIE (CONTRÔLE MASKEDBOX)

Le contrôle **MaskEdBox** (MaskEditBox) ou **Édition avec masque** offre la possibilité de limiter les données saisies et d'afficher celles-ci selon un format prédéfini. Il comporte une propriété **Text** (propriété par défaut) et se comporte de la même façon qu'un contrôle TextBox si un masque n'a pas été défini dans la propriété **Mask**. Il est accessible par **Projet /Composants /Microsoft Masked Edit Control 6.0** et le préfixe **msk** l'identifie.

La propriété Mask

La propriété **Mask** permet de définir un masque de saisie qui peut être constitué de plusieurs caractères.

Caractères les plus utiles :

#	Représente un chiffre (0 à 9).
&	Représente un caractère ANSI compris entre 32 à 126 et 128 à 255.
A	Représente un caractère alphanumérique. Exemple : a à z, A à Z ou 0 à 9.
?	Représente une lettre. Exemple : a à z, A à Z.
/	Sépare les éléments dans une date.
:	Sépare les éléments dans une heure.
> <	Convertit les lettres qui suivent le caractère > en majuscules, et celles qui suivent le caractère < en minuscules.
autres caractères	Les autres caractères sont affichés comme tels. Exemple : tiret, espace.

Exemple

date	##/##/####	jj/mm/aaaa
heure	##:##	hh:mm
heure	##:## ??	hh:mm AM
code postal	?#? #?#	j1h 2g3
code régional	(###)	(514)
téléphone	###-####	846-4545
code postal	>?#? #?#	J1H 2G3 (lettres en majuscules)
nom	>?<??????????????	

(Les noms DEMERS ou demers sont convertis en Demers.)

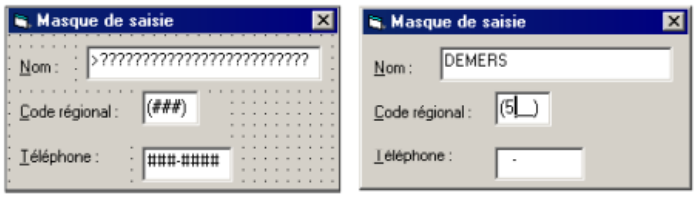
L'utilisateur doit taper des caractères de même type que ceux spécifiés dans le masque. Tout caractère invalide est rejeté et un événement **ValidationError** est déclenché. Par exemple, cet événement peut être utilisé pour coder un message comme celui-ci : "Seulement les nombres sont permis dans cette zone".

Les propriétés PromptChar et AllowPrompt

La propriété **PromptChar** permet de modifier le caractère d'invite. Par défaut, un *trait de soulignement* apparaît sous le caractère que l'utilisateur doit entrer, lorsque le contrôle a le focus.

La propriété **AllowPrompt** spécifie si oui ou non (par défaut), l'utilisateur peut entrer le caractère d'invite.

Exemple



Le nom est entré en minuscules et est convertit en majuscules. Dans la zone de saisie du code régional, des traits de soulignement sont affichés.

Un seul caractère d'invite est permis et il ne doit pas être un caractère susceptible d'être accepté, à moins que la propriété **AllowPrompt** ait la valeur **True**. Dans ce cas, la propriété **PromptInclude** doit aussi avoir la valeur **True**.

Exemple Code régional : Mask = (###)
PromptChar = 0
Masque affiché : (000)

Si l'utilisateur entre 800 et que AllowPrompt = False, seul le chiffre 8 est accepté.

La propriété PromptInclude

La propriété **PromptInclude** spécifie si les caractères de l'invite font partie ou non de la valeur de la propriété Text. Par défaut, les caractères font partie intégrante de la valeur de la propriété Text (PromptInclude = True).

Exemple

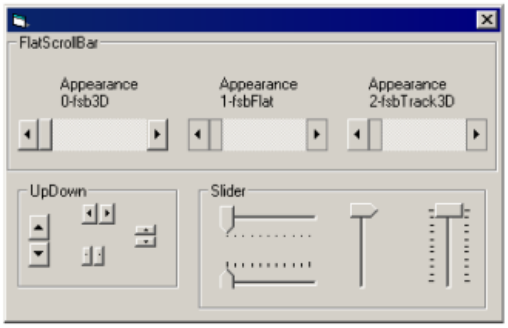
Les codes 819, 514, 450 et 418 sont valides et un code absent est accepté.

```
'Propriété PromptInclude = True (valeur par défaut)
Private Sub mskCodeReg_Validate(Cancel As Boolean)
  Select Case mskCodeReg.Text
    Case "(819)", "(514)", "(450)", "(418)", "(___)"
    Case Else
      MsgBox "Le code régional est inexistant!", _
                      vbExclamation, "Erreur"
      Cancel = True
  End Select
End Sub

'Propriété PromptInclude = False
Private Sub mskCodeReg_Validate(Cancel As Boolean)
  Select Case mskCodeReg.Text
    Case "819", "514", "450", "418", ""
    Case Else
      MsgBox "Le code régional est inexistant!", _
                      vbExclamation, "Erreur"
      Cancel = True
  End Select
End Sub
```

Pour *effacer le contenu* de la propriété Text d'un contrôle MaskEdBox, la propriété **PromptInclude** doit avoir la valeur **False**, sinon, une erreur d'exécution 380 est générée.

Exercice 27 et Exercice 31

4.9 LES CONTRÔLES FLATSCROLLBAR, SLIDER ET UPDOWN



- Le contrôle **FlatScrollBar** est une version améliorée des barres de défilement VScrollBar et HScrollBar. Ses propriétés **Min**, **Max**, **SmallChange**, **LargeChange** et **Value** sont semblables à celles des barres de défilement intégrées et il réagit de la même façon aux événements Change et Scroll (voir section Les barres de défilement *VScrollBar* et *HScrollBar*).

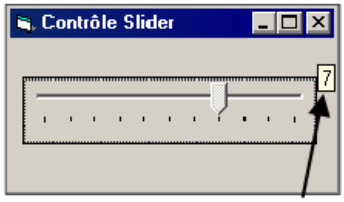
  Le préfixe **fsb** identifie le contrôle FlatScrollBar et celui-ci est accessible par **Projet /Composants /Microsoft Windows Common Controls-2 6.0**.

  La propriété **Appearance** offre trois choix :

0-fsb3D	Aspect 3D comme les barres de défilement de la boîte à outils.
1-fsbFlat	Aspect 2D (par défaut).
2-fsbTrack3D	Aspect 2D ; le curseur et les flèches apparaissent en 3D lorsqu'ils sont pointés par la souris.

  La propriété **Arrows** permet d'activer ou de désactiver les flèches et la propriété **Orientation** permet d'obtenir une barre horizontale ou verticale.

- Le contrôle **Slider** ou **Curseur** permet à l'utilisateur de sélectionner une valeur entière parmi une plage de valeurs en déplaçant un curseur sur une échelle graduée. L'utilisateur peut glisser le curseur, cliquer sur la fente ou utiliser les flèches de direction du clavier, déclenchant ainsi les événements **Change** et **Scroll**. Par défaut, la plage de valeurs est un nombre entre 0 et 10. Une info-bulle affiche la valeur du curseur. (Exemple : 7, voir l'illustration à la page suivante.)



Le préfixe **sld** identifie le contrôle Slider et celui-ci est accessible par **Projet /Composants /Microsoft Windows Common Controls 6.0**.

Quelques propriétés

Value	(Propriété par défaut) Valeur numérique entière qui correspond à la position du curseur.
Min **Max**	Valeur minimale et maximale de l'échelle de graduation. Par défaut, Min = 0 et Max = 10.
LargeChange **SmallChange**	Facteur d'incrémentation de la propriété Value lorsque l'utilisateur clique sur la fente (LargeChange) ou lorsqu'il utilise une flèche de direction du clavier (SmallChange). Par défaut, la valeur de LargeChange est 5 et celle de SmallChange est 1.
Orientation	Le curseur est orienté verticalement ou horizontalement.
TickFrequency	Fréquence des graduations sur l'échelle. Par défaut, la valeur est 1 spécifiant ainsi qu'une graduation apparaît pour chaque valeur de l'échelle. Par exemple, une valeur de 3 spécifie d'afficher une graduation à toutes les trois valeurs de la plage.
TickStyle	Présence et positionnement des graduations. Par défaut, les graduations sont affichées en bas ou à droite du curseur, selon l'orientation du contrôle.

- Le contrôle **UpDown** ou **Incrément** permet à l'utilisateur de sélectionner une valeur entière en cliquant sur des flèches afin d'augmenter ou de diminuer une valeur qui est généralement affichée dans un contrôle associé au contrôle UpDown.

Exemple

Dans la feuille qui suit, un contrôle UpDown est associé à un contrôle TextBox par la propriété **BuddyControl** et est synchronisé avec la propriété **Text** de ce même contrôle, par la propriété **SyncBuddy**. L'utilisateur peut utiliser les flèches pour modifier la valeur dans la zone de texte ou il peut entrer une nouvelle valeur directement dans la zone de texte.

Le préfixe **upd** identifie le contrôle UpDown et celui-ci est accessible par **Projet /Composants /Microsoft Windows Common Controls-2 6.0**.

Quelques propriétés

AutoBuddy	Définit automatiquement un contrôle associé. Il s'agit généralement du contrôle précédent dans l'ordre de tabulation fixé par la propriété TabIndex.
BuddyControl	Définit un contrôle associé en spécifiant son nom.
BuddyProperty	Spécifie la propriété associée du contrôle. La valeur de cette propriété est affectée à la propriété Value du contrôle UpDown, en autant que la propriété SyncBuddy est fixée à True. Si aucune propriété n'est sélectionnée, la propriété associée est la propriété par défaut du contrôle associé (exemple : Text pour un contrôle TextBox).
Increment	Détermine la valeur d'augmentation ou de diminution de la propriété Value lors d'un clic sur l'une des flèches. Par défaut, Increment = 1.
Min **Max**	Désigne la valeur minimale ou maximale de la plage de valeurs. Par défaut, Min = 0 et Max = 10.
Orientation	Spécifie si le contrôle est orienté verticalement ou horizontalement.
SyncBuddy	Détermine si oui ou non, le contrôle UpDown synchronise sa propriété Value avec une propriété du contrôle associé, spécifiée par BuddyProperty.
Value	(Propriété par défaut) Représente une valeur numérique entière qui correspond à la valeur actuelle du défilement. À chaque fois que la propriété Value est modifiée, l'événement Change est déclenché.
Wrap	Détermine si oui ou non, la propriété Value du contrôle est réinitialisée lorsque la valeur Min ou Max est atteinte.

Lorsque l'utilisateur clique sur l'une des flèches du contrôle UpDown, l'événement **UpClick** est déclenché. Celui-ci se produit après l'événement Change.

Exercice 28

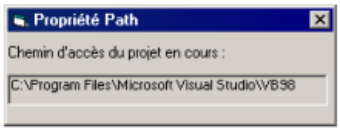
4.10 L'OBJET APP

L'objet **App** est un objet global qui représente une application, c'est-à-dire un projet Visual Basic. Il comporte plusieurs propriétés spécifiques au projet comme par exemple, le titre de l'application, le numéro de la version, le chemin d'accès, le nom de son fichier exécutable ainsi que le nom de son fichier d'aide.

La propriété **Path** de l'objet App est la propriété la plus utile de cet objet. Elle retourne le chemin d'accès du projet (fichier.vbp) lorsque l'application est exécutée à partir de l'environnement de développement. Lorsque l'application est exécutée à partir d'un fichier exécutable (fichier.exe), le chemin d'accès est celui de ce fichier.

Syntaxe : *objet*.**Path**

Exemple 1




```
lblChemin.Caption = App.Path
```

Exemple 2

Chemin d'accès d'un fichier Soleil.bmp stocké dans un dossier IMAGES sous-jacent au dossier du projet.

```
imgZoneImage = LoadPicture(App.Path & "\IMAGES\Soleil.bmp")
```

Exemple 3

Chemin d'accès d'un fichier Soleil.bmp stocké dans un dossier IMAGES sous-jacent au dossier du projet. Le projet est enregistré à la racine d'une disquette.

```
imgZoneImage = LoadPicture(App.Path & "IMAGES\Soleil.bmp")
```

Il ne faut pas spécifier un caractère \ en avant du nom de dossier parce que la propriété Path de l'objet App retourne la chaîne A:\. Le chemin d'accès ainsi constitué, A:\\IMAGES\Soleil.bmp, n'est pas un chemin valide.

Exercice 16, partie 4 et Exercice 17

4.11 L'OBJET CLIPBOARD (LE PRESSE-PAPIERS DE WINDOWS)

L'objet **Clipboard** permet d'accéder au Presse-papiers du système et de coder des opérations Copier/Couper/Coller dans une application Visual Basic. L'objet ClipBoard compte cinq méthodes, mais aucune propriété.

Clear	Vide le contenu du Presse-papiers.
GetData	Récupère un élément graphique du Presse-papiers.
GetText	Récupère une chaîne de caractères du Presse-papiers.
SetData	Insère un élément graphique dans le Presse-papiers.
SetText	Insère un texte dans le Presse-papiers.

Exemple

Insérer une chaîne de caractères dans le Presse-papiers :

```
Clipboard.SetText "chaîne de caractères"
```

Insérer un texte sélectionné d'une zone de texte dans le Presse-papiers :

```
Clipboard.SetText txtZone.SelText
```

Insérer une image dans le Presse-papiers :

```
Clipboard.SetData imgZoneImage
```

Récupérer un texte du Presse-papiers :

```
txtZone.Text = Clipboard.GetText
```

Récupérer une image du Presse-papiers :

```
imgZoneImage.Picture = Clipboard.GetData
```

Exercice 12, Exercice 20

4.12 AJOUTER ET SUPPRIMER DYNAMIQUEMENT UN CONTRÔLE

La méthode **Add** de la collection Controls (ensemble des contrôles d'une feuille) permet d'ajouter dynamiquement un nouveau contrôle à une feuille.

Syntaxe : **Me.Controls.Add (***classe*, *nom* [, *conteneur*]**)**

L'argument *classe* représente le nom de la bibliothèque d'objets et de la classe du nouveau contrôle. (Exemple : VB.TextBox pour créer une nouvelle zone de texte.)

L'argument *nom* désigne le nom du nouveau contrôle. Pour en créer plusieurs, il faut prévoir l'attribution d'un nom unique pour chaque instance (exemple : txtZone1, txtZone2, txtZone3 ...).

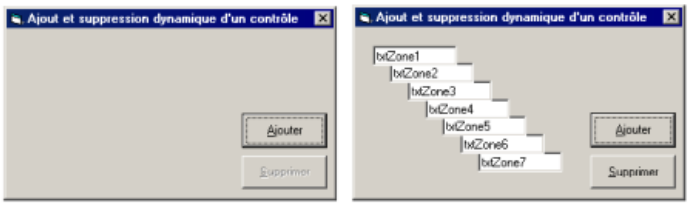
La méthode **Remove** de la collection Controls permet de supprimer un contrôle qui a été créé par la méthode Add.

Syntaxe : **Me.Controls.Remove** *nom*

L'argument *nom* désigne le nom du contrôle à supprimer.

Exemple

Une feuille contient un bouton Ajouter et un bouton Supprimer. Au démarrage, le bouton Supprimer n'est pas accessible (feuille de gauche). Après quelques clics successifs sur le bouton Ajouter, de nouvelles zones de texte sont ajoutées. Chacune d'elles porte le nom txtZone suivi d'un identificateur numérique afin que chaque nouvelle zone de texte porte un *nom unique* (exemple : txtZone1, txtZone2, ...).




```
Dim I As Integer
Dim mstrNom As String
Dim txtNom As TextBox

Private Sub cmdAjouter_Click()
  I = I + 1
  mstrNom = "txtZone" & I
  Set txtNom = Me.Controls.Add("VB.Textbox", mstrNom)
  With txtNom
    .Text = mstrNom
    .Height = 200
    .Visible = True
    .Move I * 250, I * 250
  End With
  If I = 1 Then
    cmdSupprimer.Enabled = True
  End If
End Sub
```

- La variable I est utilisée comme identificateur numérique dans le nom.
- La variable mstrNom contient le nom de la nouvelle zone de texte.
- La variable d'objet de type TextBox est utilisée par l'instruction **Set** pour référencer le nouvel objet créé par la méthode Add. Par la suite, la variable txtNom est utilisée à la place du nom de l'objet.
- Le nom est affecté à la propriété Text, la hauteur est fixée à 200, la propriété Visible est modifiée parce que, par défaut, un nouveau contrôle n'est pas visible.
- La méthode Move place le contrôle à une position relative, afin que les nouvelles zones de texte ne soient pas empilées au même endroit.
- Lorsqu'un ajout est effectué, le bouton Supprimer devient accessible.

```
Private Sub cmdSupprimer_Click()
  Me.Controls.Remove "txtZone" & I
  I = I - 1
  If I = 0 Then
    cmdSupprimer.Enabled = False
  End If
End Sub
```

- La méthode Remove supprime la dernière zone de texte qui a été créée.
- La valeur de I est réduite afin de supprimer les zones de texte en ordre décroissant (exemple : txtZone3, txtZone2, txtZone1).
- Lorsque toutes les nouvelles zones de texte sont détruites (I = 0), le bouton Supprimer n'est plus accessible.

5 Les éléments du langage de programmation

5.1 LES MODULES

Un projet, qui lorsque terminé deviendra une application, est formé d'un ensemble d'instructions appelé **code de programmation**. Le code de programmation est stocké dans des modules qui contiennent des **déclarations** et des **procédures**. Il existe trois types de modules de code : **module de feuille**, **module standard** et **module de classe**.

- Un **module de feuille** est subdivisé en **procédures événementielles**, qui contiennent des instructions qui sont exécutées, suite à des événements spécifiques. Un module de feuille peut aussi contenir des **procédures personnalisées** et des **fonctions personnalisées** qui sont exécutées en réponse à l'appel d'une autre procédure. Dans un projet, on retrouve un module de feuille pour chacune des feuilles du projet. Un module de feuille est stocké dans un fichier **.frm** avec les propriétés de la feuille. Par défaut, il est nommé **Form1**.
- Un **module standard** contient des déclarations et des procédures générales, de niveau *global*, pour l'ensemble du projet. Un module standard ne contient que du code, pas d'interface graphique. Un code de programmation commun à plusieurs feuilles devrait être placé dans une procédure d'un module standard, plutôt que de répéter le code sur chacune des feuilles. L'exécution d'un projet peut débuter par un module standard en y codant une procédure **Sub Main**. Dans ce cas, cette procédure doit être définie comme **objet de démarrage** (réf. 5.3). Un module standard porte l'extension **.bas** et par défaut, il est nommé **Module1**.

Exercice 10 et Exercice 24 (Sub Main).

- Un **module de classe** permet de créer de nouveaux objets avec ses propriétés, méthodes et événements. Un module de classe porte l'extension **.cls** et par défaut, il est nommé **Class1**.

- L'Édition Professionnelle de Visual Basic permet de créer des contrôles ActiveX dans des modules de type **Contrôle utilisateur**. Un module Contrôle utilisateur porte l'extension **.ctl** et par défaut, il est nommé **UserControl1**.

Chapitre 8, Création d'un contrôle utilisateur ActiveX.

5.2 LES TYPES DE FICHIER D'UN PROJET VISUAL BASIC

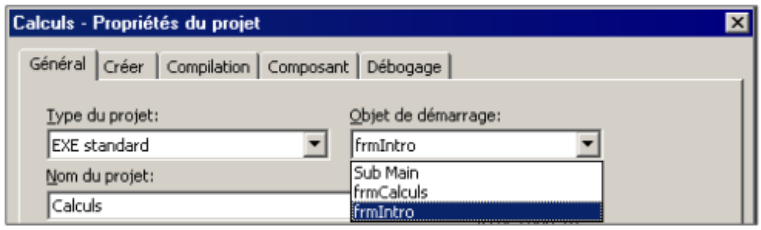
Dans un projet Visual Basic, on peut retrouver plusieurs types de fichiers :

- Un **fichier de projet (.vbp)** qui contient la liste de tous les fichiers et objets d'un projet ainsi que des paramètres d'environnement. Dans l'Explorateur de Windows, un projet Visual Basic peut être démarré par ce type de fichier.
- Un **fichier d'espace de travail (.vbw)** du même nom que le fichier d'un projet.
- Un **fichier de feuille (.frm)** pour chaque feuille d'un projet qui contient les propriétés des contrôles, les caractéristiques graphiques de la feuille et le code de programmation.
- Un **fichier binaire (.frx)** du même nom que le fichier .frm qui contient les propriétés des icônes et des images de la feuille correspondante.
- Un **fichier module standard (.bas)** qui contient le code de programmation d'un module standard.
- Un **fichier exécutable (.exe)** qui permet d'exécuter une application en dehors de l'environnement de développement Visual Basic.
- Un **fichier d'erreur (.log)** au format texte qui contient des indications sur les erreurs survenues lors du chargement d'un fichier de feuille. Un fichier d'erreur peut être consulté dans le Bloc-Notes de Windows.
- Un **fichier module de classe (.cls)** qui contient le code de création d'une nouvelle classe d'objet.
- Un **fichier contrôle utilisateur (.ctl)** qui contient le code de création d'un contrôle ActiveX.
- Un **fichier binaire (.ctx)** du même nom que le fichier .ctl qui contient les propriétés des icônes et des images du contrôle ActiveX.
- Un **fichier contrôle ActiveX (.ocx)** qui contient un contrôle ActiveX compilé.

- Un **fichier de ressources (.res)** qui contient des chaînes de caractères, des images ou d'autres données nécessaires à une application. Par exemple, pour afficher des informations en plusieurs langues sans modifier l'application.
- Un **fichier groupe de projets (.vbg)** qui permet de démarrer plusieurs projets .vbp dans l'environnement de développement Visual Basic. Ce type de fichier est très pratique pour tester un nouveau contrôle ActiveX.

5.3 L'OBJET DE DÉMARRAGE

Dans un projet de type EXE Standard, la première feuille d'un projet est utilisée comme **objet de démarrage**, c'est-à-dire la feuille qui est affichée automatiquement au démarrage du projet. L'exécution du projet débute par la première procédure événementielle de la feuille, Form_Initialize. Il est possible de modifier l'objet de démarrage en sélectionnant au menu Projet /Propriétés du projet /onglet Général /Objet de démarrage. La zone de liste "Objet de démarrage" contient le nom de toutes les feuilles du projet ainsi que l'expression **Sub Main**.



Démarrer avec Sub Main

Il est possible de créer un projet sans feuilles ou un projet qui ne démarre pas par une feuille en déclarant une procédure **Sub Main** dans un *module standard*. La procédure Sub Main doit alors être définie comme *objet de démarrage*.

Exemple d'un projet qui ne démarre pas avec une feuille

```
Sub Main()
  Call ChargerFichierStats
  frmFeuille.Show
  Call EnregistrerStats
  End
End Sub
```

Lors du démarrage du projet, la procédure `Sub Main` du module standard est exécutée. Par la suite, la procédure `ChargerFichierStats` du module standard est exécutée et

une feuille est affichée. Lorsque la feuille est déchargée, la procédure `EnregistrerStats` du module standard est exécutée et l'instruction `End` termine l'exécution du projet.

Dans l'exercice 24, un projet sans feuille est créé.

5.4 LES PROCÉDURES

Un module de code est subdivisé en **procédures**. Une procédure est une unité logique qui regroupe des instructions qui effectuent une tâche (exemple : effacer les données, calculer le salaire brut). L'utilisation de procédures permet de diviser un programme en plus petites unités qui peuvent être codées et testées individuellement, facilitant ainsi le travail du programmeur. Les types de procédures :

- Les procédures événementielles Sub ;
- Les procédures personnalisées Sub ;
- Les procédures personnalisées Function.

Déclaration et appel d'une procédure événementielle *Sub*

Une **procédure événementielle Sub** est formée d'instructions exécutées lorsqu'un événement se produit.

Déclaration de la procédure

Syntaxe : [**Private**|**Public**][**Static**] **Sub** *nomprocédure* (*liste arguments*)

instructions

[**Exit Sub**]

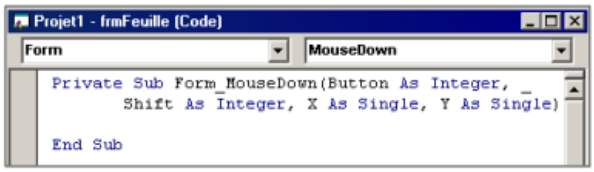
End Sub

L'argument *nomprocédure* est formé du nom d'un objet suivi d'un trait de soulignement (_) et du nom d'un événement. Par exemple, dans le nom de procédure cmdQuitter_Click, cmdQuitter est le nom de l'objet et Click est le nom de l'événement.

Les termes Private, Public et Static sont décrits dans la section La portée des procédures page 238 et l'argument *liste arguments* est traité dans la section Transmission des arguments, page 239.

L'instruction **Exit Sub** interrompt l'exécution des instructions d'une procédure. L'exécution se poursuit à partir de l'instruction qui suit celle qui a appelé la procédure Sub.

Lors de la déclaration d'une procédure événementielle, ne codez pas vous-mêmes la ligne de déclaration de la procédure, afin d'éviter des erreurs de syntaxe. Dans la fenêtre de code, sélectionnez un objet dans la zone de gauche, **Objet** (exemple : Form) et un événement dans la zone de droite, **Procédure** (exemple : MouseDown). Si la procédure contient des arguments, ceux-ci seront automatiquement ajoutés.



Appel à la procédure

Syntaxe : **Call** *nomprocédure*(*argument1*,*argument2*,*argument-n*)

nomprocédure argument1,*argument2*,*argument-n*

L'instruction **Call** fait appel à une procédure. Lorsque le mot Call est employé, les arguments doivent être entre parenthèses et si le mot Call n'est pas cité, les arguments ne doivent pas être entre parenthèses.

Exemple

```
Call cmdQuitter_Click
cmdQuitter_Click
Call txtZone_Validate(Arg1)
txtZone_Validate Arg1
```

Exercice 2, partie 2

Déclaration et appel d'une procédure personnalisée *Sub*

Une **procédure personnalisée Sub** est formée d'instructions qui visent à accomplir une tâche particulière. Une procédure personnalisée est très utile pour regrouper une série d'instructions qui doivent être exécutées par plusieurs procédures événementielles. En codant ces instructions dans une procédure personnalisée appelée par les procédures événementielles, vous éviterez de répéter inutilement des lignes de code.

Déclaration de la procédure

Syntaxe : [**Private**|**Public**][**Static**] **Sub** *nomprocédure* (*liste arguments*)
instructions
[**Exit Sub**]
End Sub

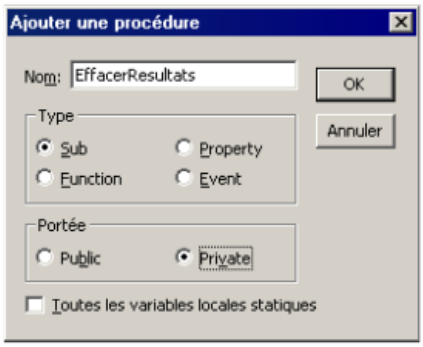
L'argument *nomprocédure* suit les mêmes règles de syntaxe qu'un nom de variable (réf. Définition d'une variable). Voici quelques normes de programmation à respecter lors de l'attribution d'un nom à une procédure personnalisée :

- Attribuez un nom qui est représentatif de la tâche ;
- Débutez le nom par une majuscule ;
- Si plusieurs mots sont employés, chacun d'eux doit débuter par une majuscule.

Exemple : EffacerDonnees, EffacerResultats, ValiderZoneSaisie.

Les termes Private, Public et Static sont décrits dans la section La portée des procédures, page 238 et l'argument *liste arguments* est traité dans la section Transmission des arguments, page 239.

Pour ajouter une procédure à votre projet, vous pouvez coder la ligne de déclaration ou sélectionner au menu Outils /Ajouter une procédure. Entrez le nom de la procédure et sélectionnez une portée Private ou Public selon vos besoins. Par défaut, le type de procédure est le type Sub.



Appel à la procédure

Syntaxe : **Call** *nomprocédure*(*argument1,argument2,argument-n*)

nomprocédure argument1,argument2,argument-n

L'appel d'une procédure personnalisée se fait comme l'appel d'une procédure événementielle (réf. Déclaration et appel d'une procédure événementielle *Sub*).

Exemple

```
Call EffacerResultats
EffacerResultats
Call Validation(Keyascii, txtCout)
Validation Keyascii, txtCout
```

Exercice 8, Exercice 14

Déclaration et appel d'une procédure *Function*

Visual Basic contient une centaine de fonctions intégrées (réf. 5.20). Il est cependant possible de coder des procédures **Function** personnalisées.

Déclaration d'une fonction

Syntaxe : [**Private**|**Public**][**Static**] **Function** *nomfontion* (*liste argument*s) [**As** *type*]

instructions

[**Exit Function**]

End Function

De façon générale, une procédure personnalisée Sub est utilisée pour effectuer une série d'instructions tandis qu'une procédure Function est utilisée dans le but d'obtenir *un résultat que la procédure Function renvoie*. Le résultat renvoyé est transmis par le nom de la fonction. La valeur retournée possède un type de données et celui-ci est défini par la clause **As** de la ligne de déclaration. Si le type est omis, la valeur de retour est de type Variant.

L'argument *nomfonction* suit les mêmes règles de syntaxe qu'un nom de variable (réf. Définition d'une variable). Utilisez des noms courts et significatifs.

Les termes Private, Public et Static sont décrits à la page 238 et l'argument *liste arguments* est traité à la page 239.

Pour ajouter une fonction à votre projet, vous pouvez coder la ligne de déclaration ou sélectionner au menu Outils /Ajouter une procédure. Entrez le nom de la fonction, sélectionnez le type Function et la portée selon vos besoins. Les arguments doivent être ajoutés par la suite.

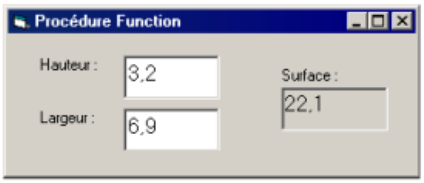
Appel d'une fonction

Une procédure Function est généralement appelée de la même façon qu'une fonction intégrée c'est-à-dire en *utilisant son nom suivi de ses arguments dans une expression*. En affectant la valeur de retour à une variable, celle-ci sera conservée.

Syntaxe : *nom* (*liste d'arguments*)

variable = *nom* (*liste d'arguments*)

Exemple




```
Private Sub lblSurface_Click()
  Dim sngHauteur As Single
  Dim sngLargeur As Single
  Dim sngSurface As Single

  sngHauteur = CSng(txtHauteur)
  sngLargeur = CSng(txtLargeur)
  sngSurface = Surface(sngHauteur, sngLargeur)
  lblSurface = FormatNumber(sngSurface, 1)
End Sub

Public Function Surface(Hauteur As Single, _
                        Largeur As Single) As Single
  Surface = Hauteur * Largeur
End Function
```

La fonction **`Surface`** est appelée lorsque son nom est cité dans le code. La fonction calcule alors la surface et le résultat de type Single est affecté à la variable `sngSurface`.

Exercice 14

La portée des procédures

La **portée d'une procédure** spécifie l'accessibilité de celle-ci par rapport aux autres procédures d'un projet.

Le mot clé **Public** indique que la procédure Sub ou Function est accessible à *toutes les procédures du projet*, donc pour l'ensemble des modules. Par défaut, une procédure est publique. Regrouper les procédures Sub et Function publiques ou globales dans un *module standard* est une bonne habitude de programmation. Attribuez un nom unique à chacune des procédures publiques de manière à ce que tous les modules puissent les appeler sans spécifier le nom du module dans lequel la procédure est codée.

Exemple

Call *nommodule.nomprocédure*

appelle une procédure codée dans un autre module ;

nommodule.nomfonction

appelle une fonction codée dans un autre module.

Si le nom d'une procédure est unique, le nom de module n'est pas nécessaire lors de l'appel.

Le mot clé **Private** indique que la procédure Sub ou Function n'est accessible qu'aux procédures du module où elle est déclarée.

Le mot clé **Static** spécifie que la valeur de toutes les variables déclarées dans la procédure (variables locales) est conservée entre les appels. Par défaut, avant l'exécution d'une procédure, toutes les variables locales sont initialisées.

Transmission des arguments

Les procédures Sub ou Function ont souvent besoin d'échanger des informations avec la procédure qui en font l'appel. Ces informations sont transmises par des variables appelées **arguments**. Les arguments sont énumérés et séparés par une virgule dans une liste d'arguments.

Syntaxe d'un argument lors de la déclaration

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *nomvariable* [**As** *type*] [= *valeur*]

Le mot clé **Optional** spécifie que l'argument est facultatif. S'il est utilisé, les arguments suivants doivent aussi être facultatifs et précédés du mot clé Optional.

Le mot clé **ByVal** indique que l'argument est transmis *par valeur*. Lorsqu'un argument est transmis par valeur, la valeur originale de la variable transmise ne peut être modifiée par la procédure à laquelle elle est passée, cette dernière disposant d'une copie de la variable plutôt que de la variable proprement dite.

Le mot clé **ByRef** indique que l'argument est transmis *par référence*. Lorsqu'un argument est transmis par référence, l'adresse de la variable est transmise. La procédure à laquelle elle est passée peut ainsi accéder à la variable proprement dite et modifier sa valeur. *Par défaut, les arguments sont transmis par référence.*

Le mot clé **ParamArray** spécifie qu'un argument est un tableau optionnel d'éléments de type Variant. Ce mot clé ne peut être utilisé que sur le dernier argument d'une liste.

L'argument *nomvariable* est le nom de la variable transmise. Celui-ci peut être différent du nom cité lors de l'appel.

L'argument **As** *type* spécifie le type de données de la variable transmise. S'il n'est pas spécifié, le type de données Variant est assumé par défaut.

L'argument facultatif = *valeur* est une valeur par défaut spécifiée par une constante, dans le cas d'un argument Optional.

En général, si les mots clé Optional et ParamArray ne sont pas utilisés, le *nombre d'arguments et l'ordre dans lequel ils sont cités doivent être identiques* lors de l'appel et lors de la déclaration de la procédure Sub ou Function. Le nom des variables transmises peut être différent mais leur type de données doit être le même.

***Exemple 1* : Transmission des arguments par référence**

Déclaration d'une procédure Sub :

```
Private Sub Validation(CodeTouche As Integer, _
                               ZoneTexte As TextBox)
```

Appel d'une procédure Sub :

```
Call Validation(Keyascii, txtCout)
```

Lors de la déclaration, les mots clé `ByVal` et `ByRef` ne sont pas cités. Dans ce cas, la transmission est faite *par référence* et les variables peuvent éventuellement être modifiées par la procédure `Validation`. La correspondance est faite entre les deux premiers arguments cités (`Keyascii` et `CodeTouche`) et les seconds (`txtCout` et `ZoneTexte`). La ligne de code suivante est équivalente à la déclaration précédente :

```
Private Sub Validation(ByRef CodeTouche As Integer, _
                       ByRef ZoneTexte As TextBox)
```

***Exemple 2* : Transmission des arguments par valeur**

Déclaration d'une procédure Function :

```
Private Function Surface(ByVal Hauteur As Single, _
                         ByVal Largeur As Single) As Single
  Hauteur = Hauteur * 10
  Largeur = Largeur * 20
  Surface = Hauteur * Largeur
End Function
```

Appel d'une procédure Function :

```
sngSurface = Surface(sngHauteur, sngLargeur)
```

Après l'exécution de la procédure Function `Surface`, les variables `sngHauteur` et `sngLargeur` ont la même valeur que lors de l'appel de la fonction, même si la valeur de `Hauteur` et de `Largeur` ont été modifiées par la fonction.

Exercice 14

5.5 LES INSTRUCTIONS D'ARRÊT (END, UNLOAD ET STOP)

L'instruction **End** termine l'exécution d'un projet. Elle peut être placée n'importe où dans une procédure. Aussitôt qu'elle est rencontrée, l'exécution est interrompue, les fichiers ouverts par une instruction Open sont fermés et les variables sont effacées de la mémoire.

Syntaxe : **End**

L'instruction **Unload** permet de mettre fin à l'exécution d'un projet en déchargeant de la mémoire chacune des feuilles du projet. Lorsqu'une feuille est déchargée, les événements QueryUnload, Unload et Terminate sont successivement déclenchés. Les deux premiers événements peuvent interrompre le déchargement en affectant la valeur True à l'argument Cancel.

Syntaxe : **Unload** *feuille*

Exemple

```
Unload Me
Unload frmFeuille
```

L'instruction **Stop** interrompt l'exécution d'un projet de la même façon qu'un point d'arrêt. L'exécution peut être reprise par la touche F5 ou le bouton Continuer de la barre d'outils Standard. Cette instruction ne doit être utilisée que lors des tests d'un projet.

Syntaxe : **Stop**

5.6 LES ÉLÉMENTS DE SYNTAXE

Répartition d'une instruction sur plusieurs lignes

Pour répartir une instruction sur plusieurs lignes, utilisez le **caractère de continuité de ligne** à la fin d'une ligne avant de continuer le code sur la ligne suivante. Le caractère de continuité de ligne est constitué d'un **trait de soulignement** (_) et il est précédé d'une espace. Il ne peut être suivi d'un commentaire.

Exemple

```
strMess = "Donnée non numérique ou absente." & _
                vbNewLine & "Seulement les caractères " & _
                "0 à 9 , + et - sont permis."
MsgBox "Impossible d'effectuer le calcul." & vbNewLine _
              & strMess, vbExclamation, "Erreur!"
```

Plusieurs instructions sur une même ligne

La syntaxe de Visual Basic permet de coder plusieurs instructions sur une même ligne en séparant chacune d'elles par deux-points (:).

Exemple

```
txtZone.Text = "" : txtZone.Enabled = True : intVar = 1
```

Les normes de programmation imposent de coder *une seule instruction par ligne* pour faciliter la compréhension du code.

Ajout de commentaires

Des **commentaires** sont des précisions placées dans le code de programmation par le programmeur, pour clarifier certaines parties du code. Ces précisions sont très utiles aux autres programmeurs susceptibles de lire le code de programmation.

Un commentaire est précédé d'une **apostrophe** ('). Il peut occuper une ligne entière ou suivre une instruction sur la même ligne. Un commentaire ne peut être placé à la suite du caractère de continuité de ligne.

Exemple

```
Dim intNb1 As Integer       'Nombre aléatoire de l'auto 1
Dim intNb2 As Integer       'Nombre aléatoire de l'auto 2

'Tirage de 2 nombres aléatoires entiers compris entre 0 et 10
intNb1 = Int(11 * Rnd)
intNb2 = Int(11 * Rnd)
```

Attribution d'un nom en Visual Basic

Plusieurs éléments qui sont utilisés dans le code de programmation doivent être nommés (exemple : procédures, variables, constantes, feuille, contrôles). Les noms qui sont attribués aux **procédures**, **variables** et **constantes** doivent respecter les **règles de syntaxe** suivantes :

- Ils doivent commencer par une lettre.
- Ils ne peuvent pas contenir de points (.) ou des caractères de déclaration de type (%, &, !, #, @, $, réf. Les caractères de déclaration de type de données).

- Ils ne peuvent comporter plus de 255 caractères ; les noms des contrôles, des feuilles, des classes et des modules sont cependant limités à 40 caractères maximum.
- Ils ne peuvent être identiques à un **mot clé**. Un mot clé est un mot qui appartient au langage Visual Basic. Exemple : If, Or, Len, And, etc.

5.7 LES VARIABLES ET LES TYPES DE DONNÉES

Définition d'une variable

Une **variable** est un espace en mémoire qui permet de conserver temporairement une valeur modifiable. Toute variable doit porter un *nom significatif* et possède un *type de données* qui détermine son contenu.

Tableau des types de données

Le tableau qui suit présente les différents types de données avec le préfixe qui identifie chacun, l'espace mémoire occupé et la plage des valeurs.

Type Préfixe	Contenu	Taille	Plage de valeurs
Byte **byt**	valeur binaire	1 octet	0 à 255
Boolean **bln**	valeur logique	2 octets	True (-1) ou False (0)
Integer **int**	entier	2 octets	-32 768 à 32 767
Long **lng**	entier long	4 octets	+/- 2 milliards
Single **sng**	nombre en virgule flottante, simple précision	4 octets	+/- 1 E-45 à 3 E38
Double **dbl**	nombre en virgule flottante, double précision	8 octets	+/- 5 E-324 à 1,8 E308
Currency **cur**	15 chiffres en avant du séparateur décimal et 4 après le séparateur. (Type utilisé pour une plus grande précision dans les calculs monétaires.)	8 octets	+/- 9 E14
Date/Time **dtm**	date et heure	8 octets	1 janvier 100 au 31 décembre 9999

String **str**	chaîne de caractères de longueur fixe dont la longueur est définie par *	1 octet par caractère	1 à 65 400 caractères
String **str**	chaîne de caractères de longueur variable (par défaut)	10 octets + la longueur de la chaîne	0 à 2 milliards de caractères
Variant **vnt**	nombres	16 octets	idem type Double
Variant **vnt**	caractères	22 octets + la longueur de la chaîne	idem type String, longueur variable
Object **obj**	référence à un objet	4 octets	adresse 32 bits
Type défini par l'utilisateur **udt**	structure de données formée par la combinaison d'éléments de plusieurs types, créée par l'instruction Type	somme des éléments	plage de valeurs de chaque type de données

Le *type de données par défaut* est le type **Variant**. Ce dernier peut contenir tous les autres types de données et Visual Basic se charge d'effectuer les conversions. Cependant, le type Variant occupe *plus d'espace en mémoire* que les autres types et demande *plus de temps de traitement*.

En général, pour les données numériques, choisissez le type de données qui occupe le *moins d'octets en mémoire* tout en tenant compte de la plage des valeurs. Pour les données autres que numériques, choisissez le type de données String pour conserver des *chaînes de caractères*. Une chaîne de caractères est délimitée par des guillemets et peut contenir tous les caractères.

Exemple

Nom	String * 30
Adresse	String * 40
Age	Integer
Sexe	Boolean
Date de naissance	Date
Taux horaire	Single
Salaire net	Single ou Currency
Population	Long

La portée des variables

La portée d'une variable spécifie l'accessibilité de celle-ci pour les différentes procédures et modules d'un projet. On retrouve des variables globales, des variables de portée module et des variable locales déclarées par les mots clés Public, Dim, Private et Static.

Les mots clés Public, Dim, Private et Static sont détaillés dans la section Déclaration de variables, page 246.

Les variables globales (préfixe **g**)

Les **variables globales** ou **publiques** sont accessibles à tous les modules d'un projet, donc à toutes les procédures. Elles résident en mémoire durant l'exécution de tout le projet.

Une variable globale est déclarée avec le mot clé **Public** dans la partie Déclarations d'un *module standard*.

Utilisez une variable globale lorsqu'une variable est utilisée par plusieurs feuilles.

Exercice 10

Les variables de portée module (préfixe **m**)

Les **variables de portée module** sont accessibles à toutes les procédures d'un même module. Les variables de portée module résident en mémoire durant l'exécution de tout le projet. Dans le cas d'un module de feuille, si une feuille est chargée en mémoire plusieurs fois, *il peut être nécessaire d'initialiser les variables lors du chargement de la feuille*.

Une variable de portée module est déclarée avec le mot clé **Dim** ou **Private** dans la section Déclarations du module.

Utilisez une variable de portée module lorsqu'une variable est utilisée par plusieurs procédures dans un même module.

Exercice 4

Les variables locales (aucun préfixe)

Les **variables locales** ou **de portée procédure** sont accessibles seulement par la procédure où elles sont déclarées.

Une variable locale est déclarée avec le mot clé **Dim** ou **Static** à l'intérieur d'une procédure. Une variable déclarée avec le mot clé Dim réside en mémoire pour la durée d'exécution de la procédure tandis qu'une variable déclarée avec le mot Static demeure en mémoire durant tout le projet.

Utilisez une variable locale lorsqu'une variable est utilisée uniquement dans une procédure. Il est possible de déclarer une *variable locale ayant le même nom* dans plusieurs procédures (exemple : I). Chacune des procédures peut modifier sa variable I sans affecter la variable I d'une autre procédure.

Exercice 4

Noms de variables selon les normes de programmation

Une variable doit porter un nom qui l'identifie et qui respecte les règles de syntaxe vues dans la section Attribution d'un nom en Visual Basic, page 242. Afin de faciliter la compréhension du code de programmation, respectez les **standards de programmation** suivants :

- Débutez le nom par un préfixe formé d'une lettre minuscule qui précise la portée de la variable ;

Préfixe	Portée
g	globale
m	module
aucun	locale

- À la suite du préfixe, spécifiez le type de données par un préfixe de 3 lettres minuscules (réf. Tableau des types de données) ;
- Attribuez un *nom significatif* débutant par une lettre majuscule ; si plus d'un mot est nécessaire, débutez chaque mot par une majuscule.

Un nom de variable doit être unique à l'intérieur d'une même portée.

Exemple

```
gintNombre          variable de type Integer de portée globale
strNom              variable de type String de portée locale
msngSalaireBrut     variable de type Single de portée module
blnTemp             variable de type Boolean de portée locale
```

Déclaration des variables

La déclaration des variables utilisées dans un programme n'est pas obligatoire. Lorsque Visual Basic rencontre un nom qui respecte les règles de syntaxe des noms de

variables, il déclare automatiquement une variable de type Variant, le type de données le plus gourmand en mémoire. Aussi, la déclaration automatique des variables est parfois une source d'erreurs qui sont difficilement repérables. La meilleure façon de procéder consiste à placer l'instruction suivante dans la section Déclarations de chacun des modules de code :

L'instruction **Option Explicit** force le programmeur à déclarer les variables qu'il utilise dans son code. Au démarrage d'un projet, si une variable n'est pas déclarée, un message d'erreur est affiché.

Pour que l'instruction Option Explicit soit ajoutée automatiquement dans chaque module, sélectionnez au menu Outils /Options /onglet Éditeur /Déclaration des variables obligatoire.

Syntaxe : {**Dim**|**Private**|**Public**} [**Static**] *nom de la variable* [**As** *type de données*]

L'instruction **Dim** déclare une variable de **portée locale** ou **module** selon que l'instruction est placée à l'intérieur d'une procédure ou au début d'un module de code.

Exemple

```
Dim strNom As String                          portée locale
Dim mstrNom As String                         portée module
Dim intNb1 As Integer, intNb2 As Integer      plusieurs par ligne
Dim strChaine As String                       chaîne de longueur variable
Dim strChaine As String * 25                  chaîne d'une longueur fixe de 25
Dim strNom                                    type Variant par défaut
```

L'instruction **Private** déclare une variable de **portée module** de la même façon que l'instruction Dim dans la section Déclarations d'un module. Le mot Private est plus significatif que le mot Dim parce qu'il contraste directement avec le mot Public.

Exemple : Les deux lignes de code suivantes sont équivalentes :

```
Private mstrNom As String                     portée module
Dim mstrNom As String                         portée module
```

L'instruction **Public** déclare une variable de portée **globale** ou **publique**. L'instruction doit être placée dans la section Déclarations d'un *module standard*.

Exemple

```
Public gstrNom As String          portée globale
```

L'instruction **Static** déclare une **variable locale** qui conserve sa valeur entre les appels de la procédure où elle est déclarée. Sa valeur est conservée durant toute l'exécution du projet.

Exemple

```
' 3 appels à la procédure personnalisée Exemple
' à partir d'une procédure Main d'un module standard
Sub Main()
  Call Exemple
  Call Exemple
  Call Exemple
End Sub
Private Sub Exemple()
  Static intVar1 As Integer
  Dim intVar2 As Integer
  intVar1 = intVar1 + 2
  intVar2 = intVar2 + 2
End Sub
```

Premier appel : intVar1 = 2, intVar2 = 2

Deuxième appel : intVar1 = 4, intVar2 = 2

Troisième appel : intVar1 = 6, intVar2 = 2

Initialisation des variables lors de la déclaration

Lors de la déclaration d'une variable, une *valeur initiale* lui est automatiquement affectée par Visual Basic.

Initialisation des variables selon le type de données :

- Les variables **numériques** (Byte, Integer, Long, Single, Double, Currency) prennent la valeur numérique 0.
- Les **chaînes de caractères** (type String) de **longueur variable** prennent la valeur "" qui représente une chaîne de caractères de longueur nulle.
- Les **chaînes de caractères** (type String) de **longueur fixe** ne sont pas initialisées.
- Les variables de type **Variant** contiennent la valeur **Vide** (= Empty).

Exemple : Pour savoir si une variable a été modifiée depuis la déclaration :

```
If varNombre <> Empty Then ...
```

- Les variables de type **Boolean** prennent la valeur **False**.

Les caractères de déclaration de type de données

Il est possible de déclarer une variable avec un **caractère de déclaration de type de données**. Un caractère de déclaration de type de données se place à la fin du nom d'une variable et spécifie le type de données de la variable.

Type de données	Caractère de déclaration
Integer	%
Long	&
Single	!
Double	#
Currency	@
String	$

Exemple : Les deux lignes de code suivantes sont équivalentes :

```
Dim lngNombre As Long
Dim Nombre&
```

Cette dénomination est basée sur le langage Basic et est maintenant peu utilisée, puisqu'elle vient à l'encontre des normes de programmation et qu'il n'existe pas de caractère de déclaration pour tous les types de données.

5.8 LES CONSTANTES

Une **constante** représente une valeur spécifique et n'est pas modifiable. Comme une variable, une constante porte un *nom significatif* et un *type de données*. Par leurs noms significatifs, les constantes améliorent la lisibilité du code de programmation. Par exemple, le nom vbWhite est plus représentatif que la valeur hexadécimale FFFFFF représentée par le code de couleur &H00FFFFFF. Dans Visual Basic, on retrouve deux types de constantes : les **constantes intégrées** et les **constantes symboliques**.

Les constantes intégrées de Visual Basic

Le nom des **constantes intégrées**, issues de la bibliothèque d'objets de Visual Basic, débute par le préfixe **vb** (exemple : vbRed, vbChecked, etc.). On retrouve une liste complète des constantes intégrées de Visual Basic dans l'Explorateur d'objets (F2) et le détail de celles-ci dans l'Index de l'aide MSDN, à la rubrique Constantes Visual Basic.

L'Annexe A présente 5 tableaux des principales constantes Visual Basic.

Les constantes symboliques ou définies par le programmeur

Les **constantes symboliques** ou **définies par le programmeur** sont déclarées à l'aide de l'instruction **Const**.

Syntaxe : [**Public**|**Private**] **Const** *nom de constante* [**As** *type de données*] = *valeur*

L'instruction **Public Const** définit une constante de **portée globale** et doit être placée dans la section Déclarations d'un module standard.

L'instruction **Const** ou **Private Const** définit une constante de **portée locale** si elle est placée dans une procédure et une constante de **portée module** si elle est placée dans la section Déclarations d'un module de code.

Le **nom d'une constante** suit les mêmes règles que le nom d'une variable et débute par le préfixe **c**. Cette lettre permet de différencier rapidement une constante d'une variable.

Lors de la déclaration, une **valeur** doit obligatoirement être affectée à une constante.

Exemple

```
Const cmstrMessage As String = "Message d'erreur"
Private Const cmstrMessage As String = "Message d'erreur"

Public Const cgsngTauxTps As Single = 0.07
Const intMinimum As Integer = 10, intMaximum As Integer = 200
```

Les constantes dont la valeur est susceptible d'être modifiée (exemple : taux de TPS lors d'un nouveau budget), sont généralement déclarées publiques et elles sont regroupées dans la section Déclarations d'un module standard afin de pouvoir les modifier rapidement.

Exercice 8

L'énumération de constantes (instruction *Enum*)

L'instruction **Enum** permet de déclarer une énumération de constantes dans le but de faciliter la compréhension du code de programmation.

Syntaxe : [**Public** | **Private**] **Enum** *noménumération*
nom1 [= *valeur*]
nom2 [= *valeur*]
nom-n [= *valeur*]
End Enum

Le mot clé **Public** spécifie une portée globale (par défaut) et le mot clé **Private** spécifie une portée de niveau module.

L'argument *noménumération* identifie le type Enum. Il est obligatoire et permet de référencer tous les autres noms (*nom1* à *nom-n*).

Les arguments *nom1* à *nom-n* représentent des constantes et doivent porter un *nom significatif* de la valeur qu'ils représentent.

L'argument *valeur* représente une valeur initiale facultative qui peut être affectée à une constante. Si aucune valeur n'est spécifiée, la valeur affectée est 0 pour la première constante, 1 pour la deuxième, 2 pour la troisième, etc. Les valeurs autorisées sont des nombres positifs ou négatifs.

Exemple 1

```
Public Enum Province
  Quebec
  Ontario
  NouveauBrunswick
  .
  .
End Enum

If intProv = 0 then…          (sans Enum)
If Quebec then…               (plus significatif avec Enum)
```

Exemple 2

```
Public Enum Departement
  Comptabilite = 50
  RessourcesHumaines = 75
  Informatique = 35
  .
  .
End Enum
```

Exercice 17

5.9 L'AFFECTATION PAR L'OPÉRATEUR =

L'**affectation** consiste à affecter une valeur à une propriété, une variable ou une constante à l'aide de l'**opérateur d'affectation =**. La valeur qui est affectée doit être du *même type de données* que la variable, la constante ou la propriété de l'objet.

Exemple

```
txtNom.Text = ""
txtPrenom.Text = "Jacinthe"
intVar = 0
intNombre = 33000     (erreur 6, dépassement de capacité)
intNombre = 5.8       (le contenu en mémoire est 5)
sngVariable = ""      (erreur 13, type incompatible)
```

```
sngVariable = ""      (erreur 13, type incompatible)
sngTauxTps = .07      (le contenu en mémoire est ,07)
sngTauxTvq = ,075     (erreur de compilation)
sngTauxTvq = .075     (le contenu en mémoire est ,075)
```

La virgule est le **symbole décimal** défini par les Paramètres régionaux de Windows (réf. 1.2). Cependant, Visual Basic utilise la virgule comme *séparateur d'arguments*. Il faut ainsi *utiliser un point au lieu d'une virgule* dans les affectations ou dans les formules mathématiques.

Les valeurs qui sont fournies par l'utilisateur doivent contenir *une virgule et non un point* pour être affectées à des variables numériques.

Exemple `sngVariable = txtTaux.Text` (affectation à une variable numérique)

La valeur 7,5 est entrée dans la zone de texte txtTaux. sngVariable = 7,5

La valeur 7.5 est entrée dans la zone de texte txtTaux. sngVariable = 75

Il est *très important de valider toute donnée saisie au clavier* pour éviter des résultats aussi imprévisibles que dans l'exemple précédent.

Le caractère # permet d'encadrer une date ou une heure lors d'une affectation à une variable de type **Date**. Pour les dates, utilisez le caractère / (barre oblique) ou - (tiret) comme séparateur et pour les heures, utilisez le caractère : (deux-points).

Exemple

```
Dim dtmDate As Date
Dim dtmHeure As Date

dtmDate = #12/2/2002#              ' #12/2/02# a été tapé
dtmDate = #12/2/2002#              ' #2002-12-2# a été tapé
dtmDate = "2 Décembre 2002"
dtmDate = "2 Déc. 2002"
dtmDate = "Déc 2 2002"
dtmDate = "2002 Déc 2"
dtmDate = "2 Dcembre 2002"         ' Erreur 13, type incompatible
dtmdate = #13/14/2002#             ' Erreur de compilation
dtmHeure = #2:08:45 AM#            ' #2:08:45# a été tapé
dtmHeure = "12:08:45"
dtmHeure = #11:56:00 PM#           ' #23:56# a été tapé
dtmheure = #24:39#                 ' Erreur de compilation
```

Au moment de la saisie d'une date ou d'une heure avec le caractère #, la date ou l'heure sont validées et converties aux formats #mm/jj/aaaa# ou hh:mm:ss AM. Les

dates et heures qui sont entrées sous forme de chaîne de caractères (entre "") ne sont pas vérifiées au moment de la saisie et elles génèrent une erreur d'exécution si elles ne sont pas valides.

5.10 LES EXPRESSIONS MATHÉMATIQUES

Les calculs sont effectués à l'aide **d'expressions mathématiques** formées **d'opérateurs mathématiques**, de variables et de constantes numériques. Les opérations mathématiques à l'intérieur d'une expression mathématique sont effectuées selon un ordre de priorité.

Les opérateurs mathématiques et leur priorité

Les opérateurs mathématiques

Opérateur	Opération	Ordre de priorité
^	Élévation à une puissance	1
-	Négation	2
*, /	Multiplication, Division	3
\	Division entière	4
Mod	Modulo (reste de la division)	5
+, -	Addition, soustraction	6

Des parenthèses peuvent être utilisées pour modifier la priorité des opérations.

Exemple
```
sngRep = (sngVar1 + intVar2) / (lngVar3 * dblVar4) - 50
```

La **division entière** renvoie comme résultat un nombre entier obtenu après que le dividende et le diviseur aient été arrondis, si nécessaire, à une valeur entière.

Exemple
```
sngReponse = 19 / 44,75
intReponse = 19 \ 4          4
intReponse = 19 \ 4.6        3        (19 \ 5)
intReponse = 19.5 \ 4.2      5        (20 \ 4)
```

L'opérateur *modulo* **Mod** permet d'obtenir le reste de la division dans une division entière.

Exemple
```
intReste = 211 Mod 4        3        (reste de la division)
```

L'opérateur ^ sert à élever un nombre à une puissance. Ce nombre peut être négatif, mais seulement dans le cas où il s'agit d'une évaluation à une puissance entière.

Exemple

```
intReponse = 2 ^ 4          16      (2 à la puissance 4)
intReponse = -9 ^ 3         -729    (-9 à la puissance 3)
intReponse = -9 ^ -3        0
```

Dans une expression mathématique, il ne faut pas utiliser le symbole monétaire ($), la virgule (,) ou le symbole de pourcentage (%).

Exemple

codez :	`59.99`	au lieu de 59.99 $
codez :	`.05`	au lieu de 5 %
codez :	`56.95`	au lieu de 56,95

Conversion numérique des données

Avant de calculer avec des valeurs issues de zones de texte qui sont des chaînes de caractères, il faut d'abord s'assurer que leur contenu est numérique et par la suite les convertir à un type de données numérique par des fonctions de conversion de types de données.

Consultez la section Les fonctions de conversion de types de données, page 286.

Exemple

```
sngTaux = Csng(txtTaux.Text)
sngTps = Csng(txtMontant.Text) * Csng(txtTaux.Text)
curSalaireBrut = Ccur(txtTauxHoraire) * Csng(txtHeures)
```

La fonction **Csng** (Converted to Single) convertit au type de données Single et la fonction **CCur** (Converted to Currency) convertit au type de données Currency. Notez que chaque chaîne de caractères est convertie individuellement. La ligne de code suivante serait inexacte :

```
sngTps = Csng(txtMontant.Text * txtTaux.Text)
```

La fonction **Val** qui convertit une chaîne de caractères en nombre n'est pas appropriée pour les nombres comportant des décimales. Dans le cas de nombres entiers, la fonction Val est cependant équivalente à la fonction **CInt** (Converted to Integer).

Exemple

```
txtNombreEtudiant.Text = 25
txtMontant.Text = 58,99

intNombre = Val(txtNombreEtudiant.Text)      (intNombre = 25)
intNombre = CInt(txtNombreEtudiant.Text)     (intNombre = 25)

sngMontant = Val(txtMontant.Text)            (sngMontant = 58)
sngMontant = CSng(txtMontant.Text)           (sngMontant = 58,99)
```

La fonction de conversion **Val** convertit correctement les nombres entiers et décimaux lorsque le **symbole décimal** défini dans les Paramètres régionaux de Windows est le **point**.

Exercice 7

5.11 LA CONCATÉNATION DES CHAÎNES DE CARACTÈRES

La **concaténation** consiste à assembler deux chaînes de caractères à l'aide de l'**opérateur de concaténation &**.

Syntaxe : *chaîne1* **&** *chaîne2*

Exemple
```
strChaine = "AAAA" & "BBBBBBB" & "CC"
strChaine = AAAABBBBBBCC
```

L'opérateur & peut aussi être utilisé pour assembler une chaîne de caractères et une variable numérique.

Exemple
```
strChaine = "La valeur de I est : " & I
strChaine = La valeur de I est : 7
```

L'opérateur + permet aussi d'assembler deux chaînes de caractères. Cependant, il ne peut joindre une variable numérique.

Exemple `strChaine = "La valeur de I est : " + I` (Erreur 13!)

Utilisez l'opérateur + seulement comme opérateur d'addition pour éviter toute confusion.

Exercice 4

5.12 AFFICHAGE DE BOÎTES DE DIALOGUE PRÉDÉFINIES

Les fonctions **MsgBox** et **InputBox** permettent l'affichage de **boîtes de dialogue prédéfinies** qui permettent une interaction entre une application et son utilisateur. Les boîtes de dialogue prédéfinies sont *modales*, ce qui signifie qu'elles doivent être fermées avant que l'utilisateur puisse accéder à une autre fenêtre de l'application.

La fonction *MsgBox*

La fonction **MsgBox** affiche un message dans une boîte de dialogue avec un ou plusieurs boutons de réponse. Lorsque l'utilisateur clique sur un bouton, une valeur de type *Integer* est retournée au programme précisant le bouton que l'utilisateur a pressé. Cette valeur n'étant pas toujours nécessaire au programme, la syntaxe de la fonction MsgBox peut différer.

Syntaxe d'un message sans valeur de retour (sans parenthèses)

MsgBox *message*[, *boutons*] [, *titre*] [, *fichier d'aide*, *contexte*]

Syntaxe d'un message avec une valeur de retour (avec parenthèses)

réponse = **MsgBox (***message*[, *boutons*] [, *titre*] [, *fichier d'aide*, *contexte*]**)**

L'argument *message* spécifie le message affiché dans la boîte de dialogue. Si le message occupe plus d'une ligne, utilisez l'une des constante suivantes : vbNewLine, vbCrLf, vcCr ou vbLf.

Ces constantes sont décrites dans la section A.5 Autres constantes.

L'argument *boutons* est la somme d'une à quatre constantes intégrées, chacune d'elles faisant partie d'un groupe différent. Il faut donc choisir *un seul élément par groupe*. Si l'argument est omis, seulement le bouton OK est présent et aucune icône n'est affichée.

Groupe 1 : Boutons de commande		
vbOKOnly	0	Bouton OK seulement (par défaut).
vbOKCancel	1	Boutons OK et Annuler.
vbAbortRetryIgnore	2	Boutons Abandonner, Réessayer et Ignorer.
vbYesNoCancel	3	Boutons Oui, Non et Annuler.
vbYesNo	4	Boutons Oui et Non.
vbRetryCancel	5	Boutons Réessayer et Annuler.

Groupe 2 : Icônes		
vbCritical	16	Icône message critique.
vbQuestion	32	Icône d'interrogation.
vbExclamation	48	Icône d'avertissement.
vbInformation	64	Icône d'information.
Groupe 3 : Boutons par défaut		
vbDefaultButton1	0	Premier bouton (par défaut).
vbDefaultButton2	256	Deuxième bouton.
vbDefaultButton3	512	Troisième bouton.
vbDefaultButton4	768	Quatrième bouton.
Groupe 4 : Modalité du message		
vbApplicationModal (par défaut)	0	La boîte de dialogue est modale. L'utilisateur doit répondre au message avant d'accéder à une autre fenêtre de l'application en cours.
vbSystemModal (Le message doit être extrêmement important !)	4096	La boîte de dialogue est modale pour le système. L'utilisateur doit répondre au message avant de pouvoir accéder à une autre application. Dans ce cas, *il n'est pas possible de passer à une autre application par la barre des tâches.*
vbMsgBoxHelpButton	16384	Ajout d'un bouton Aide.
VbMsgBoxSetForeground	65536	La boîte de dialogue apparaît toujours au premier plan.
vbMsgBoxRight	524288	Le texte est aligné à droite.
vbMsgBoxRtlReading	1048576	Le texte apparaît de droite à gauche.

L'argument *titre* est une chaîne de caractères qui est affichée dans la barre de titre du message. En l'absence de cet argument, le nom du projet apparaît dans la barre de titre.

L'argument *fichier d'aide* spécifie le nom du fichier d'aide de la boîte de dialogue et l'argument *contexte*, qui doit accompagner l'argument *fichier d'aide*, est le numéro de contexte dans le fichier d'aide. L'utilisateur peut presser le bouton Aide ou la touche F1 pour afficher la rubrique d'aide qui correspond à l'argument *contexte*.

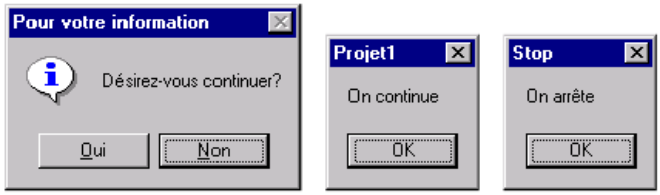
L'argument *réponse* spécifie le bouton qui a été pressé. Il s'agit d'une variable entière qui contient une valeur de 1 à 7. Chacune des valeurs correspond à une constante Visual Basic.

vbOK	1	OK
vbCancel	2	Annuler
vbAbort	3	Abandonner
vbRetry	4	Réessayer
vbIgnore	5	Ignorer
vbYes	6	Oui
vbNo	7	Non

Si la boîte de dialogue possède un bouton Annuler, la touche Échap est équivalente au bouton.

Exemple 1

```
Dim intRep As Integer

'réponse au message : emploi de parenthèses
intRep = MsgBox("Désirez-vous continuer?", _
                vbYesNo + vbInformation + vbDefaultButton2, _
                  "Pour votre information")
If intRep = vbYes Then   'ou = 6
  MsgBox "On continue"
Else
  MsgBox "On arrête",,"Stop"
End If
```



Le bouton Non (deuxième bouton) est le bouton par défaut.

Exemple 2

```
MsgBox "Le numéro est invalide" & vbNewLine & "ainsi que l'âge"
```

Exemple 3

```
MsgBox "Le programme est interrompu", vbCritical, "Erreur"
```

Exercice 1, Exercice 4, Exercice 7

La fonction *InputBox*

La fonction **InputBox** affiche une boîte de dialogue qui comprend un message, un bouton OK, un bouton Annuler et une zone de texte pour saisir une réponse.

Exemple

```
strChaine = InputBox("Message", "titre", "Tapez ici")
```

Syntaxe : **InputBox(***message*[, *titre*] [, *valeur*] [, *x*] [, *y*] [, *fichier d'aide*, *contexte*]**)**

Les arguments *message*, *titre*, *fichier d'aide* et *contexte* sont identiques à ceux de la fonction MsgBox (réf. La fonction *MsgBox*).

L'argument *valeur* est une chaîne de caractères qui est affichée dans la zone de texte. Par défaut, la zone est vide.

Les arguments *x* et *y* permettent de modifier la position d'affichage de la boîte de dialogue. L'argument *x* est la distance en twips avec l'extrémité gauche de l'écran et

l'argument *y* est la distance en twips avec l'extrémité supérieure de l'écran. Par défaut, la boîte de dialogue est centrée horizontalement dans l'écran et positionnée verticalement au tiers de l'écran.

Lorsque l'utilisateur clique sur le bouton OK ou presse la touche Entrée, la fonction InputBox retourne une chaîne de caractères représentant le texte contenu dans la zone de texte. Si l'utilisateur clique sur le bouton Annuler, la fonction retourne une chaîne de longueur nulle ("").

Exemple 1

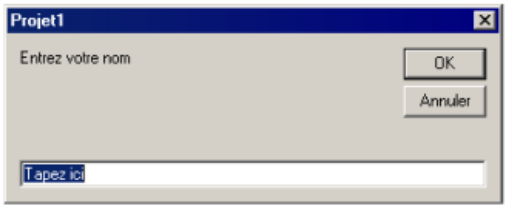
```
Dim strReponse As String
strReponse = InputBox("Entrez votre nom", "Saisie du nom", "nom")
```

Exemple 2

```
If InputBox("Entrez votre nom") = "" Then
  MsgBox "Le nom est obligatoire", , "Erreur de saisie"
End If
```

Exemple 3

```
Dim strNom As String
strNom = InputBox("Entrez votre nom", , "Tapez ici")
If strNom = "" Then
  MsgBox "Vous avez cliqué sur Annuler ou vous avez " & _
         "effacé la zone de texte et pressé OK."
ElseIf strNom = "Tapez ici" Then
  MsgBox "Vous n'avez rien tapé."
Else
  MsgBox "Bonjour " & strNom & " !"
End If
```



Exercice 8

5.13 LES STRUCTURES DE DÉCISION

Une **structure de décision** permet de vérifier une condition et d'effectuer des instructions selon le résultat de la condition. Visual Basic comporte deux structures de décision : l'instruction **If** et l'instruction **Select Case**.

L'instruction *If*

L'instruction **If** permet d'exécuter des instructions selon une condition qui peut être vraie ou fausse.

Syntaxe 1 : **If** *condition* **Then**

[*instructions si condition* ***vraie***]

[**Else**

instructions si condition ***fausse***]

End If

Syntaxe 2 : If ***condition-1*** Then

[*instructions si condition-1* ***vraie***]

[**ElseIf** *condition-2* **Then**

[*instructions si condition-2* ***vraie***]]...

[**Else**

[*instructions si toutes les conditions précédentes sont* ***fausses***]]

End If

Une **condition** peut être formée de variables, de constantes, de propriétés, de fonctions, d'opérateurs mathématiques, d'opérateurs relationnels et d'opérateurs logiques. Les variables, constantes, propriétés et opérateurs mathématiques ont déjà

été traités, alors seulement les opérateurs relationnels et les opérateurs logiques sont détaillés dans les deux prochaines sections.

Les opérateurs relationnels

Un **opérateur de relation** permet de comparer deux éléments. Ces deux éléments doivent être compatibles. Par exemple, comparer une variable String avec une variable Integer provoque une erreur d'exécution 13, type incompatible.

=	Égal à
>	Plus grand que
>=	Plus grand ou égal à
<	Plus petit que
<=	Plus petit ou égal à
<>	Différent de

Les opérateurs relationnels n'ont pas d'ordre de priorité. Ils sont évalués de gauche à droite, *après* les opérateurs mathématiques.

Exemple 7 – 5 > 9 + 3
2 > 9 + 3
2 > 12 (condition fausse)

Les opérateurs logiques

Une condition peut être **simple** (un seul opérateur relationnel) ou **composée** par l'utilisation d'un ou plusieurs **opérateurs logiques**.

Opérateur	Description	Priorité
Not	Inversion de la condition. Une condition vraie devient fausse et une condition fausse devient vraie.	1
And	Toutes les conditions simples doivent être vraies pour que la condition composée soit vraie.	2
Or (ou inclusif)	Au moins une des conditions simples doit être vraie pour que la condition composée soit vraie.	3
Xor (ou exclusif)	Si une seule des conditions simples est vraie, la condition composée est vraie.	4
Eqv (équivalence)	Évaluation de 2 conditions. Si les 2 sont vraies ou les 2 sont fausses, la condition composée est vraie.	5
Imp (implication)	Évaluation de 2 conditions. Si la première condition est vraie et que la deuxième est fausse, la condition composée est fausse. Dans tous les autres cas, la condition composée est vraie.	6

Table de vérité des opérateurs logiques

Condition1	Condition2	Not Condition1	And	Or	Xor	Eqv	Imp
Vraie	Vraie	Fausse	Vraie	Vraie	Fausse	Vraie	Vraie
Vraie	Fausse	Fausse	Fausse	Vraie	Vraie	Fausse	Fausse
Fausse	Vraie	Vraie	Fausse	Vraie	Vraie	Fausse	Vraie
Fausse	Fausse	Vraie	Fausse	Fausse	Fausse	Vraie	Vraie

Normes de programmation

Afin de s'assurer de la lisibilité du code de programmation, voici quelques règles à respecter :

- Les expressions **If** *condition* **Then** et **ElseIf** *condition* **Then** se placent seules sur une ligne ;
- Les mots clés **If**, **Else** et **End If** se placent seuls sur une ligne ;
- Les mots clés **If**, **Else**, **ElseIf** et **End If** sont alignés verticalement ;
- Les instructions conditionnelles sont mises en retrait.

Voici quelques exemples d'instructions If qui respectent la syntaxe Visual Basic, mais qui *ne rencontrent pas* les normes de programmation :

Exemple 1

```
If A = 2 Then C = C + 1
```

Le mot clé End If n'est pas nécessaire si les instructions sont sur la même ligne que la condition.

Exemple 2

```
If A > 10 Then A = A + 1 : B = B + 2
```

Exemple 3

```
If A = 1 Then strMsg = "Un" Else strMsg = "Plusieurs"
```

Exemple 4

```
If A = 1 Then
   strMsg = "Un"
Else : strMsg = "Plusieurs"
End If
```

Exemples de conditions simples

```
If txtRep = "Oui" Then
  lblFin = "Fin"
Else
  lblFin = ""
End If
```

```
If txtRep = "Oui" Then
  strFin = "Fin"
End If
```

```
If Csng(txtMontant.Text) < 0 Then      (conversion numérique)
  lblType.Caption = "Retrait"
End If
```

Exemples de conditions composées

```
If Cint(txtNombre) >= 1500 And Cint(txtNombre) < 3000 Then…

If Cint(txtNombre) < 1500 Or Cint(txtNombre) >= 3000 Then…
```

Un vendeur obtient une prime de 500 $ (sngPrime) s'il demeure en Ontario (cboProv) et qu'il vend + de 10,000 $ (sngVentes) :

```
If cboProv = "Ontario" And sngVentes > 10000 then
  sngPrime = 500
End If
```

Une prime de 1 000 $ est accordée aux vendeurs de l'Ontario et du Québec :

```
If cboProv = "Ontario" Or cboProv = "Quebec" then
  sngPrime = 1000
End If
```

Un vendeur obtient une prime de 200 $ s'il ne demeure pas en Ontario ou au Québec :

```
If cboProv <> "Ontario" And cboProv <> "Quebec" then
  sngPrime = 200
End If
```

L'emploi de l'opérateur Or dans le dernier exemple est une erreur courante de programmation. Avec l'opérateur Or, aussitôt que l'une des conditions est vraie, le résultat est vrai. Dans le cas présent, tous les vendeurs de toutes les provinces bénéficieraient d'une prime de 200 $.

Exemples de structures imbriquées

Une structure est imbriquée lorsqu'elle contient plusieurs conditions à vérifier.

Exemple 1

```
If Val(txtAge) < 18 Then
  lblMessage.Caption = "Vous n'êtes pas majeur"
Else
  If chkMembre.Value = vbChecked Then
    lblMessage.Caption = "Vous êtes un membre majeur"
  Else
    lblMessage.Caption = "Vous êtes majeur et non membre"
  End If
End If
```

Chaque instruction If requiert les mots clés End If pour en déterminer la fin ; cependant, ElseIf en un seul mot ne requiert pas de End If.

```
If Val(txtAge) < 18 Then
  lblMessage.Caption = "Vous n'êtes pas majeur"
ElseIf chkMembre.Value = vbChecked Then
  lblMessage.Caption = "Vous êtes un membre majeur"
Else
  lblMessage.Caption = "Vous êtes majeur et non membre"
End If
```

Exemple 2

```
Dim intNote As Integer
intNote = Cint(txtNote.Text)
If intNote >= 85 Then
  lblCote.Caption = "A"
ElseIf intNote >= 70 Then
  lblCote.Caption = "B"
ElseIf intNote >= 60 Then
  lblCote.Caption = "C"
Else
  lblCote.Caption = "Échec"
End If
```

Lorsqu'une *condition simple* est évaluée plusieurs fois et que des instructions différentes doivent être exécutées dans chacun des cas, il est préférable d'utiliser la structure **Select Case** au lieu de la structure **If/ElseIf**. Celle-ci est beaucoup plus simple et plus lisible.

Exercice 1, Exercice 2, Exercice 4

L'instruction *Select Case*

Dans certains cas, l'instruction **Select Case** remplace avantageusement l'instruction If/ElseIf par sa formulation plus simple. Il doit s'agir d'une condition simple qui possède au moins 2 actions possibles.

Syntaxe :

Select Case *expression_test*

[**Case** *expression_liste_1*

[*instructions si expression_liste_1 correspond à expression_test*]]

[**Case** *expression_liste_2*

[*instructions si expression_liste_2 correspond à expression_test*]]

[**Case** *expression_liste_n*

[*instructions si expression_liste_n correspond à expression_test*]]

[**Case Else**

[*instructions si aucune expression_liste ne correspond à expression_test*]]

End Select

L'argument *expression_test* peut être une expression numérique ou une chaîne de caractères. Il peut comporter des variables, des constantes, des fonctions, des opérateurs, des propriétés et les valeurs True/False.

Aussitôt qu'une expression de la liste correspond à *expression_test*, les instructions placées à la suite de l'expression correspondante sont exécutées et l'instruction suivante est celle qui suit End Select.

L'argument *expression_liste* est une liste qui peut contenir l'un des mots clés suivants :

- **To** qui indique une plage de valeurs ; la plus petite valeur doit être placée en avant du mot To et la plus grande après le mot To ;
- **Is** qui s'utilise avec un opérateur relationnel ; si le mot Is est omis, il est ajouté automatiquement par l'Éditeur de Visual Basic.

L'argument *expression_liste* peut contenir plusieurs expressions qui doivent être séparées par des virgules.

Le type de données d'une expression doit être *compatible* avec le type de données de *expression_test*. (Exemple : si expression_test est numérique, les expressions citées doivent être numériques.)

Les instructions Select Case peuvent être *imbriquées*. À chaque instruction Select Case doit correspondre une instruction End Select.

Exemple 1

```
Dim intNote As Integer
intNote = CInt(txtNote.Text)
Select Case intNote
  Case Is >= 85
    lblCote.Caption = "A"
  Case 70 To 84
    lblCote.Caption = "B"
  Case 60 To 69
    lblCote.Caption = "C"
  Case Else
    lblCote.Caption = "E"
End Select
```

Exemple 2

```
Select Case Ucase(txtMention.Text)
  Case "A", "B"
    lblMessage.Caption = "Bravo!"
  Case "C", "D"
    lblMessage.Caption = "Pas si pire..."
  Case "E"
    lblMessage.Caption = "C'est un échec."
  Case Else
    lblMessage.Caption = "La mention est invalide."
End Select
```

Dans cet exemple, la fonction **Ucase** convertit la mention en majuscule pour des besoins de comparaison. L'utilisateur peut ainsi entrer des minuscules ou majuscules, selon son choix.

Exemple 3

```
Select Case True
  Case imgVert.Visible
    imgVert.Visible = False
    imgJaune.Visible = True
  Case imgJaune.Visible
    imgJaune.Visible = False
    imgRouge.Visible = True
  Case Else
    imgRouge.Visible = False
    imgVert.Visible = True
End Select
```

Cet exemple a été codé avec la structure de décision If/ElseIf dans l'exercice 2 "Lumières de circulation".

Autres exemples

```
Case 1 To 4, 7 To 9, 11, 13, Is > intNombreMax

Case "A", "D" To "R", strLettre
```

Exercice 6

5.14 LES STRUCTURES RÉPÉTITIVES

Les structures **répétitives** permettent d'exécuter un bloc d'instructions un certain nombre de fois. Visual Basic comporte quatre structures répétitives : la boucle **For/Next**, la boucle **Do/Loop**, la boucle **While/Wend** et la boucle **For/Each**.

La boucle *For/Next*

La boucle **For Next** répète un bloc d' instructions un certain nombre de fois connu à l'avance.

Syntaxe :

For *compteur* = *valeur de départ* **To** *valeur de fin* [**Step** *valeur d'incrémentation*]
[instructions à répéter]
[**Exit For**]
[instructions à répéter]
Next [*compteur*]

L'instruction débute par le mot **For** et se termine par le mot **Next** ; les instructions qui doivent être répétées sont insérées entre les mots For et Next.

L'instruction **Exit For** permet de sortir d'une boucle ; par la suite, l'instruction située immédiatement après l'instruction Next est exécutée.

L'argument *compteur* est une variable numérique entière qui permet de compter le nombre de répétitions.

L'argument *valeur de départ* est la valeur initiale de la variable *compteur* au début de l'exécution de la boucle.

L'argument *valeur de fin* est la valeur finale de la variable *compteur* lorsque l'exécution de la boucle est terminée.

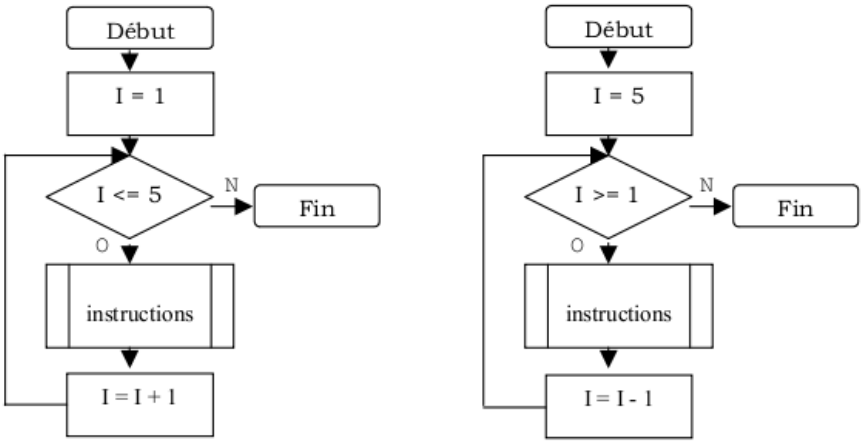
L'argument *valeur d'incrémentation* spécifie la valeur d'incrémentation de *compteur* après chaque exécution de la boucle ; si aucune valeur n'est indiquée, la valeur par défaut est 1. La *valeur d'incrémentation* contrôle l'exécution de la boucle de la façon suivante :

- Si la *valeur d'incrémentation* est >= 0, les instructions sont répétées jusqu'à ce *compteur* <= *valeur de fin*.
- Si la *valeur d'incrémentation* est < 0, les instructions sont répétées jusqu'à ce *compteur* >= *valeur de fin*.

Logique d'une boucle For

```
For I = 1 To 5
  instructions
Next I
```

```
For I = 5 To 1 Step -1
  instructions
Next I
```



Logiquement, la valeur de la variable *compteur* ne devrait pas être modifiée à l'intérieur de la boucle puisque celle-ci est incrémentée automatiquement. Une telle modification risque de provoquer des résultats imprévisibles.

Les boucles For Next peuvent *s'imbriquer* l'une dans l'autre ; chaque mot Next correspond alors à l'instruction For qui le précède. Dans une imbrication, si l'instruction Exit For est utilisée, le contrôle sera transféré à l'instruction For, située à un niveau au-dessus de la boucle dans laquelle l'instruction Exit For apparaît.

Exemple 1 : Initialisation à 0 des éléments d'un tableau (3 répétitions)

```
Dim sngTableau(1 To 3) AS Single
Dim I As Integer
For I = 1 To 3 Step 1
  sngTableau(I) = 0
Next I
```

```
For I = 1 To 3
  sngTableau(I) = 0
Next
```

Exemple 2 : Déplacement d'une image de gauche à droite

```
Dim intTwip As Integer
For intTwip = 0 To 6500 Step 50
  imgAvion.Left = intTwip
Next
```

Exemple 3 : Déplacement d'une image de droite à gauche

```
Dim intTwip As Integer
For intTwip = 6500 To 0 Step -50
  imgAvion.Left = intTwip
Next
```

Exemple 4 : Imbrication de 2 boucles For

```
Dim I As Integer, J As Integer
For I = 1 To 7
  For J = 1 To 5
    bloc d'instructions à répéter
  Next J
Next I
```

Le bloc d'instructions est exécuté **7 * 5** fois.

Exemple 5 : Imbrication de 3 boucles For

```
Dim I As Integer, J As Integer, K As Integer
For I = 1 To 4
  For J = 5 To 7
    For K = 5 To 1 Step -2
      bloc d'instructions à répéter
    Next K
  Next J
Next I
```

Le bloc d'instructions est exécuté **4 * 3 * 3** fois.

La boucle *Do/Loop* (*Do/While*, *Do/Until*)

La boucle **Do Loop** répète un bloc d'instructions *tant qu'une condition est vraie* (**While**) ou *jusqu'à ce qu'une condition devienne vraie* (**Until**).

Syntaxe :

Boucle *pré-test*	***Boucle*** *post-test*
Do [**While** | **Until** *condition*]	**Do**
[*bloc d'instructions*]	[*bloc d'instructions*]
[**Exit Do**]	[**Exit Do**]
[*bloc d'instructions*]	[*bloc d'instructions*]
Loop	**Loop** [**While** | **Until** *condition*]

L'argument *condition* est une expression conditionnelle qui peut être *vraie* (True) ou *fausse* (False).

Dans une boucle de **pré-test**, la condition est évaluée *avant* d'exécuter le bloc d'instructions tandis que dans une boucle de **post-test**, le bloc d'instructions est exécuté une première fois et la condition est évaluée *après* l'exécution ; ainsi le bloc d'instructions est toujours *exécuté au moins une fois*.

L'instruction **Exit Do** permet de sortir de la boucle. À ce moment, l'exécution se poursuit à l'instruction qui suit le mot **Loop**.

Les boucles Do Loop peuvent *s'imbriquer* l'une dans l'autre ; chaque Loop correspond alors à l'instruction Do qui le précède. Dans une imbrication, si l'instruction Exit Do est utilisée, le contrôle est transféré à l'instruction Do située à un niveau au-dessus de la boucle dans laquelle l'instruction Exit Do apparaît.

Exemple 1 : Affichage des nombres 1 à 10 dans un contrôle ListBox

```
Dim intCompt As Integer
intCompt = 1
Do While intCompt <= 10
  lstNombre.AddItem intCompt
  intCompt = intcompt + 1
Loop
```

```
Dim intCompt As Integer
intCompt = 1
Do Until intCompt > 10
  lstNombre.AddItem intCompt
  intCompt = intcompt + 1
Loop
```

```
Dim intCompt As Integer
intCompt = 1
Do
  lstNombre.AddItem intCompt
  intCompt = intcompt + 1
Loop While intCompt <= 10
```

```
Dim intCompt As Integer
intCompt = 1
Do
  lstNombre.AddItem intCompt
  intCompt = intcompt + 1
Loop Until intCompt > 10
```

Cet exemple illustre bien la différence entre une boucle *pré-test* et une boucle *post-test* ainsi que l'utilisation des mots *While* ou *Until*. Cependant, ces quatre structures pourraient avantageusement être remplacées par l'instruction **For Next**.

Exemple 2 : Saisie d'un mot de passe valide avec **While**

```
Const cstrBonMotDePasse As String = "secret"
Do While LCase(InputBox("Entrez le mot de passe :")) <> _
                                        cstrBonMotDePasse
  MsgBox "Mot de passe invalide" & vbNewLine & _
              "Entrez un autre mot de passe"
Loop
```

Le mot de passe est demandé à l'utilisateur dans une boîte de dialogue (InputBox), *tant que* celui-ci n'entrera pas le mot "secret" en minuscule ou en majuscule.

Exemple 3 : Saisie d'un mot de passe valide avec **Until**

```
Const cstrBonMotDePasse As String = "secret"
Do Until LCase(InputBox("Entrez le mot de passe :")) = _
                                         cstrBonMotDePasse
  MsgBox "Mot de passe invalide" & vbNewLine & _
              "Entrez un autre mot de passe"
Loop
```

Le mot de passe est demandé à l'utilisateur *jusqu'à ce que* celui-ci entre le mot "secret".

La boucle *While/Wend*

La boucle **While Wend** exécute un bloc d'instructions *tant que la valeur d'une condition est vraie*.

Syntaxe :
While *condition*
[*bloc d'instructions*]
Wend

Les boucles While Wend peuvent s'imbriquer. Chaque Wend correspond alors à l'instruction While qui le précède. Aucune instruction Exit n'est disponible pour sortir d'une boucle While Wend.

Exemple : Affichage des nombres 1 à 10 dans un contrôle ListBox

```
Dim intCompt As Integer
intCompt = 1
While intCompt <= 10
  lstNombre.AddItem intCompt
  intCompt = intcompt + 1
Wend
```

```
Dim intCompt As Integer
intCompt = 1
Do While intCompt <= 10
  lstNombre.AddItem intCompt
  intCompt = intcompt + 1
Loop
```

Cet exemple illustre la ressemblance entre l'instruction While Wend et l'instruction Do While. Cette dernière est plus structurée et plus souple.

La boucle *For Each*

La boucle **For Each Next** répète un bloc d'instructions pour chaque élément d'un tableau ou d'une collection.

Les tableaux sont décrits dans la section 5.18 Les tableaux en mémoire et les collections sont décrites dans la section 4.1 L'objet *Form*.

Syntaxe :
For Each *élément* **In** *groupe*
[*bloc d'instructions*]
[**Exit For**]
[*bloc d'instructions*]
Next [*élément*]

L'argument *élément* est une variable de *type Variant* qui représente chacun des éléments d'une collection ou d'un tableau. Dans le cas d'une collection, la variable peut être de type objet.

L'argument *groupe* est le nom d'une collection ou d'un tableau.

Exemple 1 : Initialisation à 0 des éléments d'un tableau à une dimension

```
Dim mlngStat(9) As Long              'tableau de portée module

Dim vntElement As Variant
For Each vntElement In mlngStat
  vntElement = 0
Next
```

Exemple 2 : Initialisation à 0 des éléments d'un tableau à deux dimensions

```
Dim mlngStat(2, 3) As Long           'tableau de portée module

Dim vntElement As Variant
For Each vntElement In mlngStat
  vntElement = 0
Next
```

Exemple 3 : Effacer toutes les zones de texte de la feuille courante

```
Dim Element As Control
For Each Element In Me.Controls
  If TypeOf Element Is TextBox Then
    Element.Text = ""
  End If
Next
```

L'exercice 15 met en pratique les différentes structures répétitives.

5.15 LA STRUCTURE WITH

L'instruction **With** permet de regrouper plusieurs instructions qui affectent un même objet, évitant ainsi au programmeur de répéter le nom de l'objet à chaque instruction.

Syntaxe : **With** *objet*
[*instructions*]
End With

Exemple

```
With txtZoneTexte
  .BackColor = vbBlack
  .ForeColor = vbWhite
  .Text = "Blanc sur noir"
  .Alignment = vbCenter
End With
```

Exercice 8

5.16 L'INSTRUCTION BEEP

L'instruction **Beep** émet un signal sonore par le haut-parleur de l'ordinateur. Son intonation et sa durée dépendent de l'ordinateur.

Syntaxe : **Beep**

Exemple

```
Private Sub cmdCloche_Click()
  Beep
End Sub
```

Un son est émis lorsque l'utilisateur clique sur le bouton Cloche.

Exercice 4

5.17 LA GESTION DES ERREURS

La **gestion des erreurs** d'exécution est un élément de programmation très important. Il consiste à intercepter une erreur avant qu'un message, souvent incompréhensible pour l'utilisateur, soit affiché et que l'exécution d'une application soit interrompue, laissant souvent un utilisateur perplexe devant son écran. L'instruction **On Error** permet d'activer un gestionnaire d'erreurs, l'instruction **Resume** offre la possibilité de reprendre l'exécution à la suite d'une erreur et l'objet **Err** fournit des détails sur les erreurs d'exécution.

Activer un gestionnaire d'erreurs avec l'instruction *On Error*

Un **gestionnaire d'erreurs** activé par l'instruction **On Error** permet d'intercepter et de traiter les erreurs au cours de l'exécution. En l'absence d'une instruction On Error, *toute erreur d'exécution est fatale et l'exécution du programme est immédiatement interrompue.*

Syntaxe : **On Error GoTo** *ligne*
On Error Resume Next
On Error GoTo 0

On Error GoTo *ligne*

L'argument *ligne* peut être une étiquette de ligne ou un numéro de ligne. Lorsqu'une erreur d'exécution survient, le contrôle est transmis au gestionnaire d'erreurs qui débute à l'endroit où est placé l'argument *ligne*. La ligne spécifiée par l'argument *ligne* doit être placée dans la même procédure que l'instruction On Error.

Exemple : Créer un gestionnaire d'erreurs

```
Private Sub cmdCalculer_Click()
On Error GoTo GestionErreur
   '
   ' Section de la procédure où une erreur d'exécution peut survenir,
   '              entre On Error et GestionErreur:
   '
   ' Il peut s'agir d'une donnée non numérique ou d'un dépassement
   ' de capacité dans une variable. Une erreur peut aussi être provoquée
   ' par la méthode Raise de l'objet Err (réf. L'objet Err et les principaux codes
d'erreur).
   '
   Exit Sub
GestionErreur:
   '
   ' Gestionnaire d'erreurs dans lequel les erreurs sont traitées par des
   ' instructions.
End Sub
```

À la rencontre de l'instruction On Error, le gestionnaire d'erreurs est activé. Toute erreur qui se produit provoque un branchement immédiat à la ligne identifiée par l'étiquette GestionErreur: où l'erreur est traitée. L'instruction **Exit Sub** placée juste avant l'étiquette GestionErreur:, provoque une sortie de la procédure dans le cas où aucune erreur n'est survenue.

On Error Resume Next

Cette instruction intercepte toute erreur d'exécution dans la procédure en cours à partir de l'endroit où elle est placée. Lorsqu'une erreur survient, Resume Next spécifie de reprendre l'exécution à partir de l'instruction qui suit celle en erreur. L'instruction en erreur est tout simplement escamotée et dans un tel cas, il faut s'attendre à des résultats plus ou moins imprévisibles. Si cette instruction est placée dans une procédure où un gestionnaire d'erreurs est activé, elle désactive ce dernier. Pour le réactiver, il faudrait recoder la ligne "On Error GoTo *ligne*".

Exemple

```
Private Sub cmdCalculer_Click()
   '   Les erreurs dans cette zone ne sont pas interceptées.
   '
On Error Resume Next
   '
   ' Toute erreur qui survient provoque un branchement
   ' à l'instruction suivante.
   '
```

On Error GoTo 0

Cette instruction désactive dans une procédure en cours tout gestionnaire d'erreurs qui a été activé par On Error GoTo *ligne*.

Exemple

```
Private Sub cmdCalculer_Click()

On Error GoTo Erreur
  '
  ' Toute erreur qui survient provoque un branchement
  ' à l'étiquette Erreur:
  '
On Error GoTo 0
  ' Si une erreur survient, celle-ci est interceptée et
  ' l'exécution continue sans tenir compte de l'erreur,
  ' jusqu'à la rencontre d'une autre instruction "On Error GoTo Erreur".
  .
  .
Erreur:
  .
  .
End Sub
```

Il faut être prudent avec l'utilisation des instructions "On Error Resume Next" et "On Error GoTo 0". Ces instructions empêchent les erreurs de provoquer un arrêt de l'exécution. Toutefois, les erreurs sont toujours présentes dans le programme et il est fort possible qu'elles engendrent d'autres erreurs difficilement repérables.

Exercice 7, Exercice 23

Reprise du traitement après une erreur (instruction *Resume*)

À l'intérieur d'un gestionnaire d'erreurs, l'instruction **Resume** permet de reprendre l'exécution après avoir traité l'erreur. Elle précise à quel endroit l'exécution doit reprendre.

Syntaxe : **Resume** [**0**]
Resume Next
Resume *ligne*

Resume [**0**]

L'exécution reprend *à partir de l'instruction qui a causé l'erreur*. Il s'agit donc d'exécuter une deuxième fois une même instruction. Par exemple, une instruction cause une erreur parce qu'elle recherche un fichier stocké sur le lecteur A:. Après avoir informé l'utilisateur qu'il doit insérer une disquette dans le lecteur, l'instruction Resume permet alors de reprendre le traitement. Il faut cependant limiter ce genre d'intervention afin de ne pas créer une boucle sans fin.

Resume Next

L'exécution reprend à partir de celle qui suit l'instruction en erreur.

Resume *ligne*

L'exécution reprend *à la ligne spécifiée*.

Exemple

```
Private Sub cmdCalculer_Click()
On Error GoTo TraiterErreur
  .
  .
Reprise:
  .
  .
  Exit Sub
TraiterErreur:
  .
  Resume
  .
  Resume Next
  .
  Resume Reprise
  .
End Sub
```

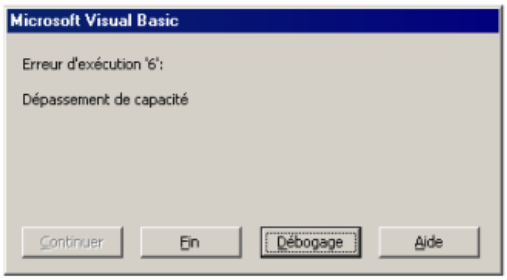
Peu importe l'instruction Resume utilisée dans le gestionnaire d'erreurs, à la rencontre de l'une d'elles, un branchement est effectué à une instruction comprise entre les instructions "On Error GoTo TraiterErreur" et Exit Sub.

Exercice 19, Exercice 20

L'objet *Err* et les principaux codes d'erreur

L'objet **Err** est un objet global qui permet d'obtenir des informations sur une erreur d'exécution, au moment où elle se produit.

La propriété **Number** retourne le numéro de l'erreur qui est survenue et la propriété **Description** retourne sa description. Dans la fenêtre qui suit, Err.Number = 6 et Err.Description = "Dépassement de capacité".



Les propriétés de l'objet Err sont initialisées après une instruction Exit Sub, Exit Function ou Resume Next. La méthode **Clear** peut être utilisée pour réinitialiser l'objet Err.

Les principaux codes d'erreur

Err.Number	Err.Description	Cause
6	Dépassement de capacité	Exemple : une valeur qui dépasse 32767 dans une variable Integer.
9	Indice en dehors de la plage	Exemple : Un tableau est déclaré avec 7 éléments et l'indice = 8.
11	Division par zéro	Le diviseur = 0 dans une opération de division.
13	Type incompatible	Exemple : comparaison entre une variable String et une variable Integer.

Vous trouverez une liste complète des erreurs d'exécution avec leurs causes dans l'Index de l'aide MSDN à la rubrique Erreurs au moment de l'exécution, Codes, bouton Afficher, Erreurs récupérables.

La méthode **Raise** de l'objet Err permet de générer une erreur d'exécution. Dans certains cas, il peut être très utile de provoquer une erreur d'exécution et par le fait même, un arrêt de l'exécution. La méthode Raise permet de numéroter nos propres erreurs et d'afficher des messages beaucoup plus significatifs que ceux qui sont générés par Visual Basic.

Syntaxe : **Err.Raise** *numéro*[, *source*][, *description*][, *fichier d'aide*][, *contexte*]

L'argument *numéro* est un entier de type Long qui identifie l'erreur. Ce numéro doit être compris entre 513 et 65 535 parce que Visual Basic utilise les numéros 0 à 512 pour ses propres erreurs.

L'argument *source* est une chaîne de caractères qui identifie l'application ou l'objet d'où provient l'erreur. Si cet argument est omis, le nom du projet sera utilisé.

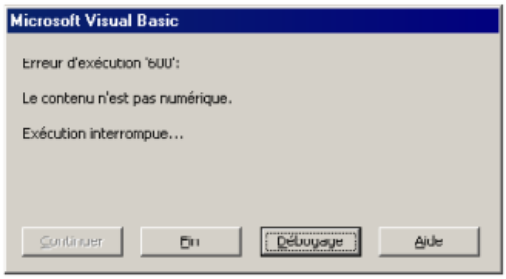
L'argument *description* est une chaîne de caractères qui précise l'erreur. En l'absence de cette chaîne, si le numéro d'erreur est celui d'une erreur Visual Basic, la description de l'erreur Visual Basic est utilisée. Dans le cas contraire, la description suivante est utilisée : "Erreur définie par l'application ou par l'objet."

L'argument *fichier d'aide* spécifie le chemin d'accès du fichier d'aide sur cette erreur. En l'absence de cet argument, les paramètres du fichier d'aide de Visual Basic seront utilisés. L'argument *contexte* est le numéro de rubrique dans le *fichier d'aide*.

Avant de générer une erreur par la méthode Raise, il est judicieux d'utiliser la méthode Err.Clear pour initialiser les propriétés de l'objet Err.

Exemple

```
If IsNumeric(txtnombre) = False Then
    Err.Raise 600, , "Le contenu n'est pas numérique." & _
                     vbNewLine & vbNewLine & _
                     "Exécution interrompue... "
End If
```



Exercice 25, Exercice 27

5.18 LES TABLEAUX EN MÉMOIRE

Appelés aussi **vecteurs** (tableaux à 1 dimension) ou **matrices** (tableaux à 2 dimensions), les **tableaux en mémoire** (terme anglais : *arrays*) ou **tableaux de variables** permettent de regrouper plusieurs variables sous un nom unique suivi d'un ou plusieurs **indices** ou **index** (terme anglais : *subscripts*) entre parenthèses, permettant un accès individuel à chacune des variables. Tout comme les groupes de contrôles, en utilisant un nom unique pour plusieurs éléments, il ainsi possible d'éviter des répétitions dans le code de programmation.

Exemple : Tableau des jours de la semaine : `strJourSem`

Dimanche	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi

Le tableau `strJourSem` compte **7 éléments** de type chaîne de caractères. Chaque élément du tableau est accessible par le nom du tableau suivi de son **indice**.

L'**indice** est une valeur numérique entière qui spécifie la *position* de l'élément dans le tableau. Il peut être représenté par une constante, une variable ou une expression numérique et par défaut, sa valeur est comprise entre 0 et $n - 1$, n étant le nombre d'éléments dans le tableau.

`strJourSem(2)`	constante	"Mardi"
`strJourSem(I)`	variable	"Jeudi" (I = 4)
`strJourSem(I + 1)`	expression numérique	"Vendredi"

Pour référencer un élément dans un tableau, *l'indice est obligatoire*. Par exemple, Visual Basic ne reconnaît pas l'expression `strJourSem`. Étant donné qu'un indice doit toujours suivre le nom d'un tableau, il est *permis d'utiliser des noms courts* comme par exemple, le nom de variable I. L'identificateur **I** (majuscule) est souvent utilisé pour représenter un **i**ndice ou un **i**ndex.

Le **nombre d'indices** cités à la suite du nom d'un tableau spécifie la *dimension* de celui-ci. Par exemple, l'élément `strJourSem(I)` fait partie d'un tableau à une dimension.

Exemple : Tableau à 2 dimensions *:* `intElement`

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)

Le tableau est constitué de 3 lignes et 4 colonnes. Il compte 12 éléments référencés avec 2 indices : `intElement(I, J)`.

Pour un tableau à 3 dimensions, il faut ajouter la profondeur en plus des lignes et colonnes. Par exemple, l'élément `lngElement(3, 4, 5)` est placé dans la quatrième ligne, cinquième colonne et au troisième plan d'un tableau 3D.

En Visual Basic, un tableau peut comporter jusqu'à 60 dimensions. On y retrouve des **tableaux de longueur fixe** et des **tableaux dynamiques**.

Utilisation d'un tableau de longueur fixe

Un **tableau de longueur fixe** comporte un nombre fixe d'éléments. Il est déclaré comme une variable simple, en précisant sa *portée* (globale, module, locale) et son *type de données*. À la suite du nom d'un tableau, la valeur maximale de ses indices doit être indiquée entre parenthèses. Le nombre d'indices cités spécifie la dimension du tableau et la valeur des indices indique le nombre d'éléments du tableau. Lors de la déclaration d'un tableau, les *éléments sont initialisés* comme les variables.

Exemple

```
Public gintTableau(10) As Integer
```

Tableau à 1 dimension de portée globale qui contient **11 éléments** de type entier : `gintTableau(0)` à `gintTableau(10)`.

Par défaut, la limite inférieure des tableaux est 0. L'instruction `Option Base 1` placée dans la section Déclarations d'un module, fixe la limite inférieure des tableaux à 1.

Exemple

```
Option Explicit
Option Base 1
Dim mintTableau(7) As Integer
```

Le tableau `mintTableau` compte 7 éléments dont les indices varient de 1 à 7.

Autres exemples

```
Dim sngNombre(60 To 80) As Single
```

Tableau à 1 dimension de portée locale dont les indices varient de 60 à 80.

```
Dim mlngPopulation(5, 4) As Long
```

Tableau à 2 dimensions de portée module qui peut contenir (6 * 5) éléments dont les indices varient de (0, 0) à (5, 4).

```
Option Base 1
Public gstrNom(7, 4, 3 To 10) As String
```

Tableau à 3 dimensions de portée globale qui peut contenir (7 * 4 * 8) éléments dont les indices varient de (1, 1, 3) à (7, 4, 10).

```
Dim msngValeur(-3 to 4) As Single
```

Tableau à une dimension de portée module dont les indices varient de –3 à 4.

Chargement des valeurs dans un tableau fixe :

```
Dim mstrJour(1 To 7) As String

Private Sub Form_Load()
   mstrJour(1) = "Dimanche"
   mstrJour(2) = "Lundi"
   mstrJour(3) = "Mardi"
   mstrJour(4) = "Mercredi"
   mstrJour(5) = "Jeudi"
   mstrJour(6) = "Vendredi"
   mstrJour(7) = "Samedi"
End Sub
```

Exercice 10, partie 2, Exercice 11, Exercice 14, Exercice 25, Exercice 26 (2 dimensions)

Les boucles For/Next et For/Each sont très utiles dans le traitement des tableaux en mémoire. Consultez les sections La boucle For/Next page 268 et La boucle For Each, page 272, pour des exemples d'utilisation.

Utilisation d'un tableau dynamique (instruction *ReDim*)

Un **tableau dynamique** est un tableau dont le nombre d'éléments et la dimension ne peuvent être définis lors de la déclaration. Le tableau est déclaré sans aucun indice par l'instruction **Dim** et par la suite, celui-ci est redimensionné dynamiquement par l'instruction exécutable **ReDim**.

Syntaxe : **ReDim** [**Preserve**] *nomtableau*(*indices*) [**As** *type*]...

Exemple

```
Dim mintTableau() As Integer

Private Sub _______
  .
  ReDim mintTableau(intNombre - 1)
  .
End Sub
```

À la rencontre de l'instruction Redim, le nombre d'éléments du tableau à une dimension `mintTableau` est fixé à (`intNombre - 1`).

L'instruction **Redim** peut être utilisée à plusieurs reprises dans un même programme pour redimensionner un tableau. À chaque fois qu'un tableau est redimensionné, les éléments de celui-ci sont réinitialisés. Pour conserver les valeurs, utilisez le mot clé **Preserve** dans l'instruction Redim : `ReDim Preserve mintTableau(intNombre - 1)`

Les fonctions *Array*, *LBound* et *UBound*

La fonction Array

La fonction **Array** retourne une variable de type Variant qui contient un tableau.

Syntaxe : **Array(***liste des éléments***)**

L'argument *liste des éléments* est une énumération de valeurs séparées par des virgules. Chaque valeur est affectée à un élément du tableau. Si aucune valeur n'est citée, un tableau de longueur nulle est créé.

Exemple : Charger les valeurs dans un tableau en une seule instruction

```
Option Explicit
Option Base 1
Dim mvntJourSem As Variant

Private Sub Form_Load()
  mvntJourSem = Array("Dimanche", "Lundi", "Mardi", _
                      "Mercredi", "Jeudi", "Vendredi", _
                      "Samedi")
  Debug.Print mvntJourSem(6)
End Sub
```

Les noms de jour sont chargés dans la variable `mvntJourSem` de type Variant qui devient un tableau. La méthode Print de l'objet Debug affiche le mot Vendredi dans la fenêtre Exécution.

Les fonctions LBound et UBound

La fonction **LBound** retourne une valeur de type Long qui représente le plus petit indice pour une dimension d'un tableau tandis que la fonction **UBound** retourne l'indice le plus élevé.

Syntaxe : **LBound(***nomtableau*[, *dimension*]**)**

UBound(*nomtableau*[, *dimension*]**)**

L'argument *dimension* est un nombre entier qui spécifie pour quelle dimension du tableau la fonction doit renvoyée une limite d'indice. Le nombre 1 spécifie la première dimension (par défaut), le nombre 2 spécifie la deuxième, etc.

Exemple

```
Private Sub Form_Load()
  Dim intValeur(4, 7) As Integer
  Debug.Print LBound(intValeur, 1)
```

```
    Debug.Print LBound(intValeur, 2)
    Debug.Print UBound(intValeur, 1)
    Debug.Print UBound(intValeur, 2)
End Sub
```

Les valeurs 0, 0, 4 et 7 sont affichées dans la fenêtre Exécution. Le tableau `intValeur` compte 5 * 8 éléments dont les indices varient de (0, 0) à (4, 7).

Exercice 15

5.19 LES TYPES DE DONNÉES PERSONNALISÉS (INSTRUCTION TYPE)

L'instruction **Type** est utilisée pour créer des **types de données personnalisés** appelés aussi **types de données utilisateur**. Ces types de données sont identifiés par le préfixe **udt** (User Data Type).

Un type de données personnalisé permet de créer une **structure de données** qui regroupe des variables avec des types de données différents. Une structure peut aussi contenir des tableaux. Chaque variable ou tableau est un élément du type de données personnalisé.

Les types personnalisés sont très utiles et même obligatoires, dans le cas des fichiers à accès direct, pour enregistrer des données liées sur un fichier. Par exemple, l'enregistrement d'un employé est formé de plusieurs données : nom, âge, taux horaire, adresse etc. Toutes ces informations sont liées entre elles.

L'instruction Type doit être placée dans la section Déclarations d'un module. Les types personnalisés peuvent être déclarés en tant que Private ou Public (module standard).

Syntaxe : [**Private** | **Public**] **Type** *nomtype*
nomélément [**(**[*indices*]**)**] **As** *type***...**
End Type

L'argument *nomtype* est le nom du type de données utilisateur. Il doit être conforme aux règles de syntaxe des noms de variables.

L'argument *nomélément* représente le nom de chaque élément du type utilisateur. Il doit respecter les règles de syntaxe des noms affectés aux variables.

Utilisez des noms qui débutent par une majuscule et aucun préfixe.

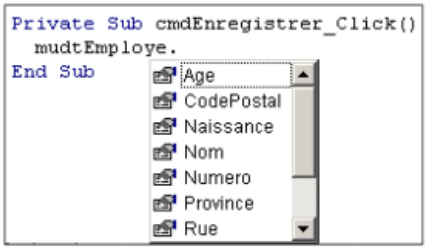
Exemple : Structure d'enregistrement d'un employé

```
Option Explicit
Private Type Employe
  Numero As Integer
  Nom As String * 30
  Rue As String * 20
  Ville As String * 20
  Province As String * 2
  CodePostal As String * 6
  Sexe As Boolean
  Age As Integer
  Naissance As Date
  SalaireBrut As Single
End Type

Dim mudtEmploye As Employe     'portée module

Private Sub cmdEnregistrer_Click()
  .
  mudtEmploye.Province = "Qc"
  .
End Sub
```

L'instruction Type crée le type de données `Employe` formé de plusieurs éléments. Pour utiliser ce type de données, il faut déclarer une variable `mudtEmploye` et spécifier le type de données `Employe`.

Pour accéder à un élément de la variable, la syntaxe est semblable à celle des propriétés d'objets. Exemple : `mudtEmploye.Province = "Qc"`

Lors de la saisie du code, à la suite d'un point, l'Éditeur de Visual Basic affiche un liste des éléments de la variable :



Exercice 27

5.20 LES FONCTIONS INTÉGRÉES

Visual Basic contient une multitude de fonctions intégrées. Les prochaines sections décrivent la plupart d'entre elles. Afin de s'y retrouver plus facilement, les diverses fonctions sont classées selon le genre d'opérations qu'elles accomplissent.

Les fonctions de conversion de types de données

Les **fonctions de conversion de type de données** convertissent une *expression* en un type de données spécifique.

Syntaxe : **Fonction(***expression***)**

Tableau des fonctions de conversion de type de données

CBool	Conversion au type Boolean.
CByte	Conversion au type Byte.
CCur	Conversion au type Currency.
CDate	Conversion au type Date.
CDbl	Conversion au type Double.
CInt	Conversion au type Integer.
CLng	Conversion au type Long.
CSng	Conversion au type Single.
CStr	Conversion au type String.
CVar	Conversion au type Variant.
Str	Conversion d'un nombre en chaîne de caractères.
Val	Conversion d'une chaîne de caractères en nombre.

L'argument *expression* est une expression de chaîne de caractères ou une expression numérique. La valeur de l'*expression* doit être compatible avec la plage de valeurs du type de données spécifié par le nom de la fonction. Par exemple, la conversion suivante est impossible :

```
intVar = CInt(50567)
```

La fonction **Str** convertit un nombre en chaîne de caractères. Cette fonction ne reconnaît que le point comme séparateur décimal. Dans le cas de décimales, utilisez la fonction **CStr**.

La fonction **Val** convertit une chaîne de caractères en valeur numérique. Cependant, celle-ci ne reconnaît que le point comme séparateur décimal. Lorsque la chaîne à convertir est une chaîne de longueur nulle ("") ou non numérique, la valeur numérique 0 est retournée.

Les fonctions **CInt** et **CLng** arrondissent les décimales.

Avant de convertir une expression avec la fonction **CDate**, utilisez la fonction **IsDate** pour vérifier si l'expression correspond à une date valide ou à une heure valide, dont le format est défini dans les Paramètres régionaux de Windows. Lors de la conversion d'un nombre par la fonction CDate, la partie entière du nombre est convertie en date et la partie décimale est convertie en heure. Par exemple, la valeur 0,5 représente la moitié de la journée, c'est-à-dire midi, la journée débutant à minuit.

Exercice 7, Exercice 9

Les fonctions de conversion

Asc(*chaîne*)	Retourne un nombre entier qui correspond au code ANSI du premier caractère de la chaîne spécifiée. Exemple : `Asc("A")` retourne 65 Consultez l'Annexe B pour les différents codes ANSI. Exercice 8
Chr(*code de caractère*)	Retourne une chaîne qui contient le caractère correspondant au code spécifié. Exemple : `Chr(65)` retourne "A"
Hex(*nombre*)	Retourne une chaîne qui représente la valeur hexadécimale du nombre spécifié. Exemple : `Hex(18)` retourne 12
Oct(*nombre*)	Retourne une chaîne qui représente la valeur octale du nombre spécifié. Exemple : `Oct(18)` retourne 22

Les fonctions de chaînes de caractères

Lcase(*chaîne*)	Retourne une chaîne en lettres minuscules.
Ucase(*chaîne*)	Retourne une chaîne en lettres majuscules.
LTrim(*chaîne*)	Retourne une chaîne dont les espaces à gauche sont supprimées.
RTrim(*chaîne*)	Retourne une chaîne dont les espaces à droite sont supprimées.
Trim(*chaîne*)	Retourne une chaîne dont les espaces à gauche et à droite sont supprimées.
Left(*chaîne, nombre*)	Retourne une chaîne contenant le nombre de caractères d'une chaîne spécifiée à partir de la gauche.
Right(*chaîne, nombre*)	Retourne une chaîne contenant le nombre de caractères d'une chaîne spécifiée à partir de la droite.
Mid(*chaîne, position* [, *longueur*])	Retourne une chaîne d'une longueur précise à partir d'une position donnée ; si la longueur est omise, tous les caractères à partir de la position sont retournés.
Len(*chaîne*)	Retourne une valeur de type Long qui correspond à la longueur ou au nombre de caractères d'une chaîne.

InStr([*position*,] *chaîne1*, *chaîne2*)	Recherche d'une *chaîne2* dans une *chaîne1*. Retourne une valeur de type Long représentant la position de *chaîne2* dans *chaîne1* à partir d'une *position* de départ dans *chaîne1* ; par défaut, la position de départ = 1.
InStrRev([*position*,] *chaîne1*, *chaîne2*)	Cette fonction est semblable à la précédente sauf que la recherche est effectuée à partir de la fin de la chaîne.
Space(*nombre*)	Retourne une chaîne contenant le nombre d'espaces spécifié.
String(*nombre*, *caractère*)	Retourne une chaîne qui contient un certain nombre de fois un caractère spécifié.
StrReverse(*chaîne*)	Retourne une chaîne dont les caractères sont inversés.
StrConv(*chaîne*, *type de conversion*)	Retourne une chaîne convertie selon un type de conversion spécifié par une constante intégrée : vbUpperCase majuscule vbLowerCase minuscule vbProperCase première lettre de tous les mots en majuscule et les autres en minuscules
Replace(*chaîne1*, *chaîne2*, *chaîne3* [, *début*][, *nombre*])	Retourne une chaîne dans laquelle les caractères *chaîne2* ont été remplacés par des caractères *chaîne3*. L'argument *début* indique à quelle position dans *chaîne1* le remplacement doit être effectué (par défaut = 1). L'argument *nombre* spécifie le nombre de remplacements à effectuer (par défaut, toutes les occurrences sont remplacées).

Exemple `txtNom.Text = ␣␣␣␣Poirier` (le nom est précédé de 4 espaces)

`txtNom = Ltrim(txtNom)`	`Poirier`	suppression des espaces de gauche
`txtNom = Ucase(txtNom)`	`POIRIER`	conversion en majuscules
`txtNom = Lcase(txtNom)`	`poirier`	conversion en minuscules
`intLongueur = Len(txtNom)`	`7`	longueur de la chaîne
`strRes = Left(txtNom, 2)`	`po`	extraction de 2 caractères à partir de la gauche
`strRes = Right(txtNom, 4)`	`rier`	extraction de 4 caractères à partir de la droite
`strRes = Mid(txtNom, 3, 1)`	`i`	extraction de 1 caractère à partir du troisième
`txtPrenom = "Marie-France"` `If Instr(txtPrenom, "-") > 0…`		recherche d'un tiret dans la chaîne ; la valeur retournée est 6 qui est la position du tiret dans la chaîne
`strChaine = String(5, "*")`	`*****`	chaîne de 5 *
`txtNom = StrReverse(txtNom)`	`reiriop`	inversion des caractères
`txtNom = StrConv("jean dupuis RENOIR", vbProperCase)`	`Jean Dupuis Renoir`	premier caractère de chaque mot converti en majuscule et les autres en minuscules

`strChaine = "aaa␣␣aaa␣␣"`
 	
`strChaine = Replace(strChaine, "aaa", "zzz")`
 	
`strChaine = Replace(strChaine, "aa", "zz")`
	
	`zzz␣␣zzz␣␣`
	
	`zza␣␣zza␣␣`
		remplacement d'une chaîne de caractères par une autre dans la variable `strChaine`

`"Allo" & Space(3) & "toi!"`	`Allo␣␣␣toi!`	3 espaces

Les instructions **LSet** et **RSet** permettent d'aligner à gauche (LSet) ou à droite (RSet) une chaîne, à l'intérieur d'une variable de type String.

Syntaxe : **RSet** *variable* = *chaîne*

LSet *variable* = *chaîne*

Exemple

```
Dim strVariable As String * 7

LSet strVariable = "Allo!"

RSet strVariable = "Allo!"
```

Après l'instruction `LSet, strVariable = Allo!␣␣`.
Après l'instruction `RSet, strVariable = ␣␣Allo!`.

Les fonctions mathématiques

Abs(*nombre*)	Retourne la valeur absolue d'un nombre.
Atn(*nombre*)	Retourne la valeur de l'arc tangente d'un nombre.
Cos(*nombre*)	Retourne la valeur du cosinus d'un nombre.
Exp(*nombre*)	Retourne la valeur exponentielle d'un nombre.
Fix(*nombre*)	Retourne la partie entière d'un nombre. Si le nombre est négatif, le premier entier négatif supérieur ou égal à *nombre* est retourné. Exemple : `Fix(-5.2)` retourne –5.
Int(*nombre*)	Retourne la partie entière d'un nombre. Si le nombre est négatif, le premier entier négatif inférieur ou égal à *nombre* est retourné. Exemple : `Int(-5.2)` retourne –6.
Log(*nombre*)	Retourne le logarithme d'un nombre.
Round(*nombre* [, *décimales*])	Retourne un nombre arrondi avec un nombre spécifique de *décimales*. Exemple : `Round(5.4)` retourne 5 `Round(5.456, 2)` retourne 5,46 `Round(5.567, 1)` retourne 5,6

Sgn(*nombre*)	Retourne une valeur 0, 1 ou –1 qui identifie le signe d'un nombre : 0, si le nombre = 0 1, si le nombre > 0 -1, si le nombre < 0
Sin(*nombre*)	Retourne la valeur du sinus d'un angle.
Sqr(*nombre*)	Retourne la valeur de la racine carrée d'un nombre.
Tan(*nombre*)	Retourne la valeur de la tangente d'un angle.

Les fonctions financières

DDB, **SLN**, **SYD**	Calcul d'un amortissement.
NPV, **PV**	Calcul de la valeur nette d'un investissement.
IRR, **MIRR**	Calcul d'un taux de rendement interne.
IPmt, Pmt, PPmt	Calcul d'un paiement pour rembourser un emprunt. Syntaxe : **Pmt**(*rate*, *nper*, *valeur*) *rate* est le taux pour un paiement ; *nper* est le nombre total de paiements ; *valeur* est le montant d'emprunt. Exemple : Pour rembourser 3 000 $ en 3 ans (36 mensualités) à un taux annuel de 7 % : `Pmt(.07 / 12, 36, 3000)` retourne -92,63 $ `-Pmt(.07 /12 , 36, 3000)` retourne 92,63 $
FV	Calcul de la somme payée pour rembourser un emprunt. Exemple : `-FV(0.07 / 12, 36, 92.63)` retourne 3 698,73 $
Nper	Calcul du nombre de paiements pour rembourser un emprunt. Exemple : `-NPer(0.07 / 12, 92.63, 3698.73)` retourne 36
Rate	Calcul du taux d'intérêt pour rembourser un emprunt. Exemple : `(Rate(36, -92.63, 3000) * 12)` retourne 0,07

Exercice 9 Financement automobile

Les fonctions *IsNumeric*, *IsEmpty*, *IsNull* et *IsDate*

IsNumeric(*expression*)	Retourne une valeur de type booléenne (True/False) indiquant si le contenu de l'*expression* est numérique. Exercice 7
IsEmpty(*variable*)	Retourne une valeur de type booléenne (True/False) spécifiant si une variable de type Variant contient la valeur vide (= Empty). Les variables de type Variant sont initialisées avec la valeur vide.

IsNull(*variable*)	Renvoie une valeur de type booléenne (True/False) indiquant si le contenu ne contient aucune donnée valide, ce qui correspond à la valeur Null. Les champs d'une base de données Access dont aucune valeur n'a été affectée ont une valeur = Null.
IsDate(*expression*)	Retourne une valeur de type booléenne (True/False) spécifiant si une *expression* correspond à une *date* ou à une *heure* valide. Par exemple, un mois 13 ou un jour 33 ne sont pas valides. Les dates et les heures devraient être vérifiées par cette fonction *avant* d'être utilisées dans des comparaisons ou des calculs. Lorsque *expression* est la propriété Text d'un contrôle MaskEdBox, sa propriété PromptInclude doit être fixée à True pour utiliser la fonction IsDate. Exercice 27

Exemple

`If IsNumeric(txtNumero.Text)…`
`If IsNumeric(txtNumero.Text) = True…`
`If Not IsNumeric(txtNumero.Text)…`
	Validation numérique d'une zone de texte.

`If IsEmpty(vntNombre)…`
`If IsEmpty(vntNombre) = True…`
`If vntNombre = Empty …`
	Une variable de type Variant, a-t-elle encore sa valeur d'initialisation ?

`If IsNull(CodePostal)…`
 	
`If CodePostal = Null…`
	Si le champ CodePostal de la base de données ne contient rien…
	La comparaison avec Null *n'est pas équivalente* à la ligne de code précédente. Utilisez la première forme pour vérifier si un champ contient une donnée valide.

`If IsDate(txtDateRemise.Text)…`
`If IsDate(txtHeureRemise.Text)…`
 	
`If CDate(txtDateRemise.Text) < Date…`
`If CDate(txtHeureRemise.Text) < Time…`
	Validation d'une date ou d'une heure.
	
	Une fois qu'une date ou une heure est validée, elle peut être comparée, en autant qu'elle est convertie au format Date.

`mskDate.PromptInclude = True` `If IsDate(mskDate.Text)…`	Validation d'une date saisie avec un contrôle MaskEdBox.

Les fonctions *VarType* et *TypeName*

La fonction VarType

La fonction **VarType** permet d'identifier le type de données d'une variable. Elle retourne une valeur de type Integer qui correspond à une constante Visual Basic.

Syntaxe : **VarType(***variable***)**

Exemple `VarType(intVariable)` retourne la valeur 2.

Tableau des valeurs de retour

vbEmpty	0	Empty (valeur vide)
vbNull	1	Null (données non valides)
vbInteger	2	Entier
vbLong	3	Entier long
vbSingle	4	Nombre à virgule flottante en simple précision
vbDouble	5	Nombre à virgule flottante en double précision
vbCurrency	6	Valeur monétaire
vbDate	7	Valeur de date
vbString	8	Chaîne de caractères
vbObject	9	Objet
vbError	10	Valeur erronée
vbBoolean	11	Valeur booléenne
vbVariant	12	Variant (pour les tableaux Variant)
vbDataObject	13	Objet d'accès aux données
vbDecimal	14	Valeur décimale
vbByte	17	Valeur binaire
vbUserDefinedType	36	Type défini par l'utilisateur
vbArray	8192	Tableau

La fonction TypeName

La fonction **TypeName** permet d'identifier le type de données d'une variable. Elle retourne une chaîne de caractères qui correspond au type de données identifié.

Syntaxe : **TypeName(***variable***)**

Exemple `TypeName(intVariable)` retourne la chaîne "Integer".

Les fonctions de date et heure

Date	Retourne une valeur de type Date représentant la date actuelle du système. La fonction **Date** permet aussi de modifier la date du système. Exemple : `If dtmDate > Date` compare avec la date du jour `Date = #31/12/2003#` modifie la date du système

<table>
<tr><td>DateAdd(intervalle, nombre, date)</td><td>Retourne une valeur de type Date contenant une date à laquelle un intervalle de temps spécifié a été ajouté.
Exemple : <code>dtmDateEcheance = DateAdd("d", 30, Date)</code>
La date d'échéance = la date du jour + 30 jours
Valeurs de l'argument intervalle :
<table>
<tr><td>yyyy</td><td>année</td><td>q</td><td>trimestre</td></tr>
<tr><td>m</td><td>mois</td><td>y</td><td>jour dans l'année</td></tr>
<tr><td>d</td><td>jour</td><td>w</td><td>jour dans la semaine</td></tr>
<tr><td>ww</td><td>semaine</td><td>h</td><td>heure</td></tr>
<tr><td>n</td><td>minute</td><td>s</td><td>seconde</td></tr>
</table></td></tr>
<tr><td>DateDiff(intervalle, date1, date2)</td><td>Retourne une valeur de type Long indiquant l'intervalle de temps entre deux dates.
Exemple : <code>dtmDate1 = #4/14/2002# '14 avril
dtmDate2 = #12/1/2002# '1 décembre
lngSem = DateDiff("ww", dtmDate1, dtmDate2)
lngSem = 33 semaines</code></td></tr>
<tr><td>DatePart(intervalle, date)</td><td>Retourne une valeur de type Integer contenant un élément d'une date.
Exemple : <code>intJour = DatePart("y", "2/25/2002")
intJour = 56</code> (le 25 février est le 56^{ième} jour de l'année)</td></tr>
<tr><td>DateSerial(année, mois, jour)</td><td>Convertit au type Date, 3 nombres de type Integer qui correspondent à une année, à un mois et à un jour.
Exemple : <code>dtmDate = DateSerial(2002, 2, 16)
dtmDate = 2002-02-16</code></td></tr>
<tr><td>DateValue(date)</td><td>Convertit au type Date une chaîne de caractères qui représente une date.
Exemple : <code>dtmDate = DateValue("18 décembre 2002")
dtmDate = DateValue("18-déc.-2002")
dtmDate = 2002-12-18</code></td></tr>
<tr><td>Day(date)</td><td>Retourne une valeur de type Integer représentant un jour dans le mois (1 à 31).
Exemple : <code>dtmDate = #2/28/2002#
intJour = Day(dtmDate)
intJour = 28</code></td></tr>
<tr><td>Hour(heure)</td><td>Retourne une valeur de type Integer représentant un nombre d'heures (0 à 23).
Exemple : <code>dtmHeure = #3:39:58 PM#
intHeure = Hour(dtmHeure)
intHeure = 15</code></td></tr>
</table>

Minute(*heure*)	Retourne une valeur de type Integer représentant le nombre de minutes dans une heure (0 à 59).
	Exemple : `dtmHeure = #3:39:58 PM#`
	`intMinute = Minute(dtmHeure)`
	`intMinute = 39`

Month(*date*)	Retourne une valeur de type Integer représentant le mois dans une date (1 à 12).
	Exemple : `dtmDate = #2/28/2002#`
	`intMois = Month(dtmDate)`
	`intMois = 2`

MonthName(*mois*)	Retourne une chaîne spécifiant un mois (janvier à décembre). Exemple : `strMois = MonthName(11)` `strMois = "novembre"`

Now	Retourne une valeur de type Date spécifiant la date et l'heure actuelles du système d'exploitation.
	Exemple : `dtmDate = Now`
	`Now = 2002-06-29 12:26:34`
	La date et l'heure du système sont enregistrées sous forme numérique.
	Exemple : `sngNombre = Now`
	`37436,52`
	La date correspond à la partie entière et l'heure à la partie fractionnaire. Il y a 37 436 jours écoulés depuis le 1 janvier 1900. Une fraction égale à 0 représente minuit et 0,50 représente midi. Ainsi 0,52 correspond à 12:26.

Second(*heure*)	Retourne une valeur de type Integer représentant le nombre de secondes dans une minute (0 à 59).
	Exemple : `dtmHeure = #3:39:58 PM#`
	`intSeconde = Second(dtmHeure)`
	`intSeconde = 58`

Time	Retourne une valeur de type Date représentant l'heure actuelle du système. La fonction **Time** permet aussi de modifier l'heure du système. Exemple : `If dtmHeure > Time` compare avec la date du jour `Time = #12:30#` modifie l'heure
Timer	Retourne une valeur de type Single représentant le nombre de secondes écoulées depuis minuit. Exemple : `sngSecondes = Timer` `sngSecondes = 43567,49`

TimeSerial(*heure, minute, seconde*)	Convertit au type Date, 3 nombres de type Integer qui correspondent à une heure, minute et seconde. Exemple : `dtmDate = TimeSerial(13,12,14)` `dtmDate = 13:12:14`
TimeValue(*heure*)	Convertit au type Date, une chaîne de caractères qui représente une heure. Exemple : `dtmHeure = TimeValue("5:2 PM")` `dtmHeure = "17:02:00"`
Weekday(*date*)	Retourne une valeur de type Integer spécifiant le numéro du jour dans une date (1 à 7, dimanche à samedi). Exemple : `intNoJour = Weekday(Date)` `intNoJour = 3`
WeekdayName (*numéro du jour*)	Retourne une chaîne de caractères qui correspond à un numéro de jour (dimanche à samedi). Exemple : `strJour = WeekdayName(intNoJour)` `strJour = "mardi"`

Year(*date*)	Retourne une valeur de type Integer représentant l'année dans une date.
	Exemple : `dtmDate = #2/28/2002#`
	`intAnnee = Year(dtmDate)`
	`intAnnee = 2002`

Note :

Les fonctions de date et d'heure sont dépendantes des **Paramètres régionaux** du système. Il ne faut donc pas s'étonner de certaines différences.

Les fonctions de formatage

Une opération de formatage consiste à *afficher* ou à *imprimer des résultats* selon certains formats.

Les fonctions de formatage prédéfinies

FormatCurrency(*expression* [, *nombre de décimales*])	Formatage selon le format monétaire défini dans les Paramètres régionaux de Windows. Exemple : 45 678,99 $
FormatPercent(*expression* [, *nombre de décimales*])	Formatage sous forme de pourcentage. L'expression est multipliée par 100 et un caractère % est ajouté à la fin. Exemple : ,07 devient 7%

FormatNumber(*expression* [, *nombre de décimales*])	Formatage sous forme de nombre avec un nombre fixe de décimales. Exemple : `FormatNumber(sngNombre,3)`

FormatDateTime(*date|heure* [, *nom du format*])

Formatage d'une date ou d'une heure selon un format prédéfini.

Valeurs de *nom du format* :

vbGeneralDate	0	Format général de date et d'heure selon le contenu du paramètre *date|heure*.
		`FormatDateTime(Date, vbGeneralDate) 2002-06-29`
		`FormatDateTime(Now, vbGeneralDate) 2002-06-29 6:45:34`
		`FormatDateTime(Time, vbGeneralDate) 6:45:34`

vbLongDate	1	Format complet de date. `FormatDateTime(Date, vbLongDate)  29 juin 2002`
vbShortDate	2	Format abrégé de date. `FormatDateTime(Date, vbShortDate)  2002-06-29`
vbLongTime	3	Format étendu d'heure. `FormatDateTime(Time, vbLongTime)`
vbShortTime	4	Format abrégé d'heure. `FormatDateTime(Time,vbShortTime)`

La fonction Format

La fonction **Format** formate une expression selon un format spécifié. Il peut s'agir d'un **format nommé prédéfini** ou d'un **format défini par l'utilisateur**.

Syntaxe : **Format(***expression, format***)**

Pour formater des **nombres**, utilisez des formats numériques nommés prédéfinis ou créez des formats numériques définis par l'utilisateur.

Pour formater des **dates et des heures**, utilisez des formats de date et d'heure nommés prédéfinis ou créez des formats de date et d'heure définis par l'utilisateur.

Pour formater des **chaînes de caractères**, créez des formats de chaîne définis par l'utilisateur.

Formater des nombres avec des formats nommés prédéfinis

"currency"	Affichage avec un séparateur de milliers, deux décimales et le symbole monétaire. Exemple : `lblRes = Format(5000, "currency")` `lblRes = 5 000,00 $`
"fixed"	Affichage avec au moins un entier et deux décimales. Exemple : `lblRes = Format(1250.7, "fixed")` `lblRes = 1250,70`
"standard"	Affichage avec un séparateur de milliers, au moins un entier et deux décimales. Exemple : `lblRes = Format(1250.7, "standard")` `lblRes = 1 250,70`
"percent"	Affichage d'un nombre multiplié par 100 avec le signe %, avec 2 décimales. Exemple : `lblRes = Format(0.058, "percent")` `lblRes = 5,80%`
"general number"	Affichage du nombre sans séparateur de milliers. Exemple : `lblRes = Format(1250.778, "general number")` `lblRes = 1250,778`
"scientific"	Affichage avec la notation scientifique standard. Exemple : `lblRes = Format(1250, "general number")` `lblRes = 1,25E+03`

"Yes/No"	Affichage de **Non** (si numérique et valeur = 0) ou de **Oui**.
	Exemple : `blnVar = True`
	`lblRes = Format(blnVar, "Yes/No")`
	`lblRes = Oui`

"True/False"	Affichage de **Faux** (si numérique et valeur = 0) ou de **Vrai**.
	Exemple : `sngVar = 0`
	`lblRes = Format(sngVar, "True/False")`
	`lblRes = Faux`

Formater des nombres avec des formats définis par l'utilisateur

Le tableau suivant présente les caractères disponibles pour créer des formats numériques définis par l'utilisateur :

0	Espace réservé à un chiffre ou à un zéro non significatif. `Format(5.4, "00.00")` 05,40 `Format(555.456, "00.00")` 555,46
#	Espace réservé à un chiffre. Les zéros non significatifs ne sont pas affichés. `Format(5.4, "###.##")` 5,4 `Format(555.456, "####.#")` 555,5 `Format(-2000, "####.#")` -2000, `Format(2000-, "####.#")` erreur de compilation
. (point)	Espace réservé au séparateur décimal. `Format(0.4, "0.0")` 0,4 `Format(0.456, "##.##")` ,46
%	Espace réservé au signe pourcentage %. `Format(0.4, "0.0%")` 40,0% `Format(0.04, "##%")` 4%
, **␣ (espace)**	Espace réservé au séparateur de milliers. `Format(555.678, "0,000.00")` 0 555,68 `Format(555.678, "# ###.####")` 555,678
- + $ ( )	Le caractère est affiché à la position spécifiée. `Format(500, "$###+")` $500+ `Format(5000, "($###-)")` ($5000-) `Format(-5000, "($###-)")` -($5000-)
****	Affiche des caractères à la suite du format. `Format(8334.9, "#,##0.00\ CR")` 8 334,90 CR

Un format numérique défini par l'utilisateur peut être constitué de **un à quatre paramètres** séparés par un **point-virgule**.

Premier paramètre : Format pour **toutes les valeurs**.

Deuxième paramètre : Format pour **les valeurs négatives**.

Troisième paramètre : Format pour **les valeurs = 0**.

Quatrième paramètre : Format pour **les valeurs = Null**.

Exemple

```
lblRes = Format(500, "$#,##0.00;($#,##0.00)")       $500,00
lblRes = Format(-50000, "$#,##0.00;($#,##0.00)")    ($50 000.00)
lblRes = Format(-500, "$#,##0.00;$#,##0.00 -")      $500,00 -
```

Si 2 points-virgules sont consécutifs, le paramètre manquant est affiché selon le format d'une valeur positive.

Exemple

```
lblRes = Format(500.35, "$#,##0;;\Zéro")            $500
lblRes = Format(-500, "$#,##0;;\Zéro")              -$500
lblRes = Format(0, "$#,##0;;\Zéro")                 Zéro
```

Utiliser les formats de date/heure nommés prédéfinis

General Date	Format général de date et d'heure selon le contenu du paramètre qui précède le nom du format.
	`Format(Date, "General Date")` `2002-06-29`
	`Format(Now, "General Date")` `2002-06-29 6:45:34`
	`Format(Time, "General Date")` `6:45:34`

Long Date	Format complet de date. `Format(Date, "Long Date")` `25 février 2002`
Medium Date	Format semi-complet de date. `Format(Date, "Medium Date")` `25-févr-02`
Short Date	Format abrégé de date. `Format(Date, "Short Date")` `2002-02-25`
Long Time	Format étendu d'heure. `Format(Time, "Long Time")` `14:54:07`
Medium Time	Format semi-étendu d'heure. `Format(Time, "Medium Time")` `02:54 PM`
Short Time	Format abrégé d'heure. `Format(Time, "Short Time")` `14:54`

Utiliser les formats de date/heure définis par l'utilisateur

:	Séparateur d'heure.
/	Séparateur de date.
c	La date sous la forme **ddddd** et l'heure sous la forme **ttttt**.
d	Le jour sans 0 non significatif (1 à 31).
dd	Le jour avec un 0 non significatif (01 à 31).
ddd	Le jour sous la forme abrégée (dim à sam).
dddd	Le jour sous la forme complète (dimanche à samedi).
ddddd	La date sous la forme **jj/mm/aa**.
dddddd	La date sous la forme **jj mmmm aaaa**.
w	Le jour de la semaine (1 à 7, dimanche à samedi)
ww	Le numéro de la semaine dans l'année (1 à 53).
m	Le mois sans 0 non significatif (1 à 12).
mm	Le mois avec 0 non significatif (01 à 12).
mmm	Le mois sous la forme abrégée (jan à déc).
mmmm	Le mois sous la forme complète (janvier à décembre).
y	Le jour de l'année (1 à 366).
yy	L'année sous la forme `AA` (01 à 99).
yyyy	L'année sous la forme `AAAA` (100 à 9999).
q	Le numéro du trimestre (1 à 4).
ttttt	L'heure sous la forme complète hh:mm:ss.
h	L'heure sans 0 non significatif (0 à 23).
hh	L'heure avec 0 non significatif (00 à 23).

n	Les minutes sans 0 non significatif (0 à 59). Dans certains cas, **m** peut être utilisé.
nn	Les minutes avec zéro non significatif (00 à 59). Dans certains cas, **mm** peut être utilisé.
s	Les secondes sans 0 non significatif (0 à 59).
ss	Les secondes avec 0 non significatif (00 à 59).
AM/PM	Affichage avec le format 12 heures ; **AM** en majuscules pour une heure avant midi ; **PM** en majuscules pour une heure entre midi et 11h59.
am/pm	Affichage avec le format 12 heures ; **am** en minuscules pour une heure avant midi ; **pm** en minuscules pour une heure entre midi et 11h59.
A/P	Affichage avec le format 12 heures ; **A** en majuscule pour une heure avant midi ; **P** en majuscule pour une heure entre midi et 11h59.
a/p	Affichage avec le format 12 heures ; **a** en minuscule pour une heure avant midi ; **p** en minuscule pour une heure entre midi et 11h59.
AMPM	Affichage en minuscules ou majuscules selon les paramètres régionaux. Le format par défaut est **AM/PM**.

Exemple

```
Format(Date, "dddd, d mmm yyyy")     samedi, 29 juin 2002

Format(Date, "c")                    2002-06-29
Format(Date, "ddddd")                2002-06-29

Format(Date, "dddddd")               29 juin 2002

Format(Date, "ddd dd mmmm yy")       sam. 29 juin 02

Format(Time, "ttttt")                16:28:34

Format(Time, "hh:nn:ss am/pm")       04:28:47 am
Format(Time, "hh:mm:ss am/pm")       04:28:47 am
```

Utiliser les formats de chaînes de caractères définis par l'utilisateur

<	Conversion en minuscules. `Format("ExPrESSion", "<")` `expression`
>	Conversion en majuscules. `Format("ExPrESSion", ">")` `EXPRESSION`

@	Espace réservé à un caractère ou à un caractère d'espacement en l'absence d'un caractère à la position occupée par **@**.
	`Format("mot", "@@@@@@>")` `␣␣MOT`
	`Format("mot", "@@@@@>!")` `MOT␣␣`

!	Remplacement de gauche à droite des caractères par des espaces. Par défaut, le remplacement s'effectue de droite à gauche.
	`Format("mot", "@@@@@@@@")` `␣␣␣␣␣mot`
	`Format("mot", "!@@@@@@@")` `mot␣␣␣␣␣`

&	Espace réservé à un caractère. Aucun caractère d'espacement n'est ajouté. `Format("mot", "&&&&&&")` `mot`

Les fonctions de sélection (*IIf*, *Switch* et *Choose*)

La fonction IIf

La fonction **IIf** retourne l'un ou l'autre de deux arguments selon qu'une condition est vraie ou fausse.

Syntaxe : **IIf(***condition,argument1, argument2***)**

L'*argument1* est retourné si la condition est *vraie* et l'*argument2* est retourné si la condition est *fausse*.

Exemple

```
lblType.Caption = IIf(sngmontant >= 0, "Dépôt", "Retrait")
```

Si le montant est >= 0, le mot Dépôt est affiché dans l'étiquette. Dans le cas contraire, le mot Retrait est affiché.

La fonction Switch

La fonction **Switch** retourne une valeur parmi une liste, selon une condition.

Syntaxe : **Switch(***expression1***,** *valeur1* [, *expression2, valeur2*]…**)**

Si la condition qui correspond à une valeur est vraie, celle-ci est retournée. Les expressions et les valeurs doivent apparaître par paires.

Exemple

```
strVille = "Rome"
lblLangue.Caption = Switch(strVille = "Londres", "Anglais", _
                           strVille = "Rome", "Italien", _
                           strVille = "Paris", "Français", _
                           strVille = "Hong Kong", "Mandarin", _
                           strVille = "Mexico", "Espagnol")
```

Dans l'exemple précédent, le mot "Italien" est affiché dans l'étiquette lblLangue. Si strVille = "Montréal", la valeur retournée est Null et l'erreur d'exécution 94, "Utilisation non autorisée de Null" est générée parce que la valeur Null n'est pas autorisée pour la propriété Caption d'une étiquette.

La fonction Choose

La fonction **Choose** retourne une valeur énumérée dans une liste dont la position correspond à une valeur d'index. Si *index* = 1, la première valeur est retournée, si *index* = 2, la deuxième valeur est retournée, etc.

Syntaxe : **Choose(***index*,*valeur-1*[, *valeur-2*]…**)**

L'argument *index* est un nombre compris entre 1 et le nombre de valeurs énumérées.

Les arguments *valeur-n* constituent une liste de valeurs séparées par des virgules.

Exemple

```
lblNomCouleur.Caption = Choose(Val(txtNoCouleur.Text), _
                    "Jaune", "Rouge", "Bleu", "Vert", "Blanc")
```

Si la zone de texte contient la valeur 2, le mot "Rouge" est affiché dans l'étiquette lblNomCouleur.

Autres fonctions

La fonction LoadPicture

La fonction **LoadPicture** permet de charger un élément graphique, comme une image, dans la propriété **Picture** d'une **feuille**, d'un contrôle **PictureBox** ou d'un contrôle **Image**.

Exemple

```
imgGravure.Picture = LoadPicture("C:\Windows\lstboot.bmp")
imgGravure.Picture = LoadPicture(App.Path & "\Images\mailll.ico")
imgGravure.Picture = LoadPicture()
```

Les deux premières lignes de code chargent des images dont le nom de fichier est spécifié avec le chemin d'accès. L'expression App.Path représente le chemin d'accès du projet. En l'absence d'un nom de fichier, la fonction LoadPicture *supprime l'image* du contrôle ou de la feuille.

Exercice 16, partie 4

La fonction Rnd (instruction **Randomize)**

La fonction **Rnd** permet de générer aléatoirement un nombre. Elle retourne une valeur de type Single comprise entre 0 et 1.

Syntaxe : **Rnd**[(*nombre*)]

Pour générer un premier nombre aléatoire, la fonction **Rnd** utilise une *valeur initiale* et par la suite les autres nombres sont générés à partir du nombre précédent. Ainsi, à chaque appel de la fonction **Rnd**, la même série de nombres aléatoires est générée. Pour obtenir la valeur initiale d'une série de nombres, utilisez l'instruction **Randomize** avant de faire appel une première fois à la fonction **Rnd**.

Syntaxe : **Randomize** [*nombre*]

L'instruction **Randomize** initialise le générateur de nombres aléatoires à partir d'un *nombre* qui représente une *nouvelle valeur initiale*. En l'absence de l'argument *nombre*, l'heure actuelle du système est utilisée.

La valeur de l'argument *nombre* définit la façon dont la fonction Rnd génère une série de nombres :

nombre	Valeur retournée par la fonction Rnd
nombre < 0	La valeur de *nombre* est utilisée comme valeur initiale.
nombre > 0	Nombre aléatoire suivant dans la série (par défaut).
nombre = 0	Dernier nombre aléatoire généré.

Exemple

```
Private Sub Form_Load()
  Randomize
End Sub
Private Sub cmdGenererNombre_Click()
  lblNombre.Caption = Rnd(-1)
End Sub
```

À chaque clic sur le bouton "Générer nombre", le *même nombre* est toujours affiché à cause de la valeur constante –1 qui est fixe.

```
Private Sub cmdGenererNombre_Click()
  lblNombre.Caption = Rnd
End Sub
```

Un *nouveau nombre* est généré à chaque clic sur le bouton "Générer nombre".

Pour générer des nombres aléatoires *entiers* compris entre une *limite inférieure* et une *limite supérieure*, utilisez la formule ci-dessous :

Int((*limite sup.***- *limite inf.* + 1) * Rnd +** *limite inf.***)**

Exemple

```
intNombre = Int((10 - 1 + 1) * Rnd + 1)       'entre 1 et 10
intNombre = Int((50 - 10 + 1) * Rnd + 10)     'entre 10 et 50
```

Les formules peuvent être simplifiées ainsi :

```
intNombre = Int(10 * Rnd + 1)                 'entre 1 et 10
intNombre = Int(41 * Rnd + 10)                'entre 10 et 50
```

Exercice 16, parties 1 et 3

La fonction QBColor

La fonction **QBColor** retourne une valeur de type Long représentant un code de couleur associé à un *nombre entier* compris entre 0 et 15. Cette fonction provient de l'ancêtre de Visual Basic : QuickBasic. Pour un plus grand choix de couleurs, utilisez la fonction RGB décrite dans la section suivante.

Syntaxe : **QBColor(***nombre***)**

Tableau des valeurs de l'argument *nombre* :

0	Noir	4	Rouge	8	Gris	12	Rouge clair
1	Bleu	5	Magenta	9	Bleu clair	13	Magenta clair
2	Vert	6	Jaune	10	Vert clair	14	Jaune clair
3	Cyan	7	Blanc	11	Cyan clair	15	Blanc pur

Exemple

```
QBColor(15)          retourne le code décimal du blanc (16777215).
Hex(QBColor(15))     retourne le code hexadécimal du blanc (FFFFFF).

txtZoneTexte.BackColor = QbColor(14)
```

L'arrière-plan de la zone de texte devient jaune.

Exercice 16, partie 2

La fonction RGB

La fonction **RGB** (**R**ed, **G**reen, **B**lue) retourne une valeur de type Long qui représente une couleur.

Syntaxe : **RGB(***red, green, blue***)**

Les arguments *red*, *green*, et *blue* sont des nombres entiers compris entre 0 et 255 qui définissent respectivement la composante rouge, verte et bleue d'une couleur.

Tableau des valeurs de quelques couleurs standard :

Couleur	Rouge	Vert	Bleu
Noir	0	0	0
Bleu	0	0	255
Vert	0	255	0
Cyan	0	255	255
Rouge	255	0	0
Magenta	255	0	255
Jaune	255	255	0
Blanc	255	255	255

Exemple

```
lblNombre.Caption = RGB(255, 255, 0)          retourne 65535
lblNombre.Caption = Hex(RGB(255, 255, 0))     retourne FFFF
lblNombre.BackColor = RGB(255, 255, 0)        arrière-plan jaune
```

Le code de couleur du jaune a la valeur décimale 65535 et la valeur hexadécimale FFFF.

Exercice 16, partie 1

La fonction Shell

La fonction **Shell** permet d'exécuter un programme exécutable.

Syntaxe : **Shell** *nom du fichier exécutable*

L'argument *nom du fichier exécutable* est une chaîne contenant le nom du fichier .exe avec son chemin d'accès.

Exemple

```
Shell "C:\Windows\Calc.exe"
```

La fonction Shell démarre la calculatrice de Windows.

La fonction DoEvents

La fonction **DoEvents** arrête temporairement l'exécution et donne le contrôle au système d'exploitation pour qu'il exécute les tâches se trouvant dans sa file d'attente. Cette fonction est utile avant d'exécuter des tâches qui demandent un maximum de ressources système ou des ressources continues.

Syntaxe : **DoEvents**

Exemple

```
DoEvents
bloc d'instructions qui produit une animation
.
.
```

Avant de produire une animation, la fonction DoEvents est appelée pour que le système exécute toutes les tâches en attente. Lorsque l'animation débute, celle-ci est exécutée sans intermittence puisque le système est libre de toutes autres tâches.

Exercice 16, partie 4

6 Découvrir des contrôles utiles par la pratique

6.1 LES CONTRÔLES DU SYSTÈME DE FICHIERS

Les contrôles du système de fichiers **DriveListBox** (liste des lecteurs), **DirListBox** (liste des dossiers) et **FileListBox** (liste des fichiers) sont inclus dans la boîte à outils. Ils permettent de retrouver des fichiers dans le système en sélectionnant le lecteur et le dossier du fichier recherché. L'exercice 18 vise à mettre en relation ces 3 contrôles, lors de la sélection d'un fichier par l'utilisateur.

Les contrôles du système de fichiers sont détaillés dans la section Les contrôles *DriveListBox*, *DirListBox* et *FileListBox*, page 183.

Exercice 18 Créer le projet *Sélection d'un fichier*

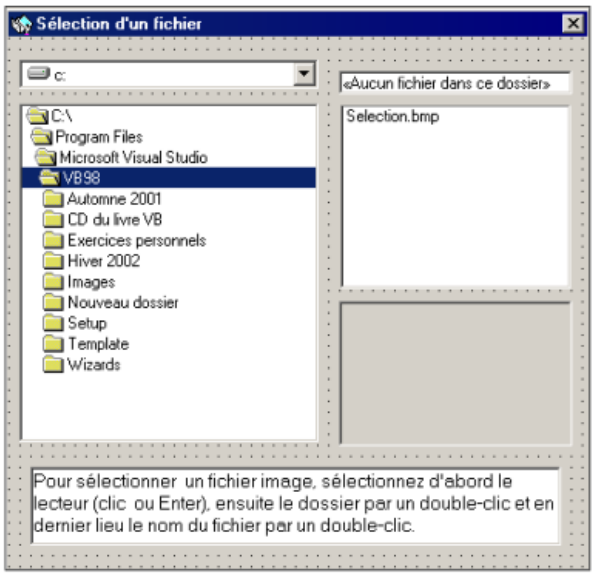
Étape 1 Préparation, observation et enregistrement

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre6 /Exercice18 /Debut dans un nouveau dossier .../Visual Basic /Exercice 18 Fichiers.
2. Exécutez le fichier .../Visual Basic /Exercice 18 Fichiers /**Fichiers.exe**. Cette application permet d'afficher un fichier Image dont l'extension est .bmp, .ico, .cur ou .wmf dans un contrôle PictureBox. Parcourez les dossiers de votre système afin de sélectionner des fichiers correspondants à ces types.

Le dossier **Program Files \Microsoft Visual Studio \Common \Graphics** regroupe l'ensemble des fichiers Image fournis avec Visual Basic.

3. Dans Visual Basic, ouvrez un nouveau projet. Nommez le projet **Fichiers** et la forme **frmFichiers**. Enregistrez la forme dans le **Fichiers.frm** du dossier .../Visual Basic /Exercice 18 Fichiers et le projet dans le fichier **Fichiers.vbp** du même dossier.

Étape 2 Conception de l'interface graphique du projet



Ajoutez les contrôles suivants sur la feuille :

- 1 étiquette explicative au bas de la feuille ;
- 1 contrôle **DriveListBox** dans le haut de la feuille (**drvLecteur**) ;
- 1 contrôle **DirListBox** à gauche de la feuille (**dirDossier**) ;
- 1 contrôle **FileListBox** à droite de la feuille (**filFichier**) ;
- 1 étiquette **lblNonTrouve** au-dessus du contrôle FileListBox qui ne doit pas être visible au démarrage ;
- 1 contrôle **PictureBox** sous le contrôle FileListBox (**picImage**).

Étape 3 Coder les procédures

1. Ajoutez la procédure suivante :

```
'Lorsque l'utilisateur sélectionne un lecteur,
'les dossiers doivent être synchronisés
'avec le nouveau lecteur.
Private Sub drvLecteur_Change()
On Error GoTo GestionErreur
```

```
  dirDossier.Path = drvLecteur.Drive
  Exit Sub
GestionErreur:
  drvLecteur.Drive = dirDossier.Path
  MsgBox "Le lecteur choisi n'est pas prêt.", _
                        vbCritical, "Erreur"
End Sub
```

Lorsque l'utilisateur sélectionne un lecteur en cliquant ou en pressant la touche Enter, l'événement **Change** du contrôle DriveListBox est déclenché. En affectant la valeur de la propriété **Drive** du contrôle DriveListBox à la propriété **Path** du contrôle DirListBox, les 2 contrôles sont ainsi synchronisés.

La propriété **Drive** du contrôle DriveListBox retourne le nom du lecteur sélectionné (exemple : C:\).

La propriété **Path** du contrôle DirListBox retourne le chemin d'accès du dossier sélectionné (exemple : C:\Windows\Fichiers).

L'instruction **On Error** permet d'intercepter toute erreur qui pourrait se produire. Par exemple, il peut s'agir d'un CD-ROM ou d'un lecteur de disquettes vide. En cas d'erreur, le dossier sélectionné précédemment redevient le lecteur sélectionné.

2. Ajoutez la procédure suivante :

```
'Lorsque l'utilisateur sélectionne un dossier,
'la liste des fichiers est synchronisée avec
'le nouveau dossier.
Private Sub dirDossier_Change()
  filFichier.Path = dirDossier.Path
  If filFichier.ListCount = 0 Then
    lblNonTrouve.Visible = True
  Else
    lblNonTrouve.Visible = False
  End If
End Sub
```

Lorsque l'utilisateur sélectionne un dossier par un double-clic, l'événement **Change** du contrôle DirListBox est déclenché. En affectant la même valeur à la propriété **Path** des contrôles DirListBox et FileListBox, les 2 contrôles sont ainsi synchronisés.

Si le dossier sélectionné ne comporte aucun fichier (ListCount = 0), une étiquette contenant l'information «Aucun fichier dans ce dossier» est affichée.

3. Ajoutez la procédure suivante :

```
Private Sub filFichier_dblClick()
  Dim strChemin As String
  If Len(filFichier.Path) = 3 Then
    'Fichier à la racine du lecteur, 3 caractères (ex : C:\),
    'il faut seulement joindre le nom du fichier
    strChemin = filFichier.Path & filFichier.FileName
  Else
    'Fichier non à la racine du lecteur,
    'il faut joindre au chemin d'accès un \ et le nom du fichier
    strChemin = filFichier.Path & "\" & filFichier.FileName
  End If
On Error GoTo Erreur
  picImage.Picture = LoadPicture(strChemin)
  Exit Sub
Erreur:
  MsgBox "Impossible d'afficher ce fichier." _
              , vbInformation, "Erreur"
End Sub
```

> Lorsque l'utilisateur sélectionne un fichier par un double-clic, la fonction **LoadPicture** charge le fichier image dans le contrôle PictureBox. Si le fichier sélectionné n'est pas un fichier image, un message d'erreur est affiché.
>
> Afin de spécifier un chemin d'accès valide dans la fonction LoadPicture, il faut vérifier si le fichier est stocké à la racine d'un lecteur ou non. Si la longueur du chemin d'accès du fichier est 3, il s'agit nécessairement d'un dossier situé à la racine d'un lecteur. Dans ce dernier cas, il ne faut pas ajouter le caractère \ dans le chemin d'accès pour éviter un chemin d'accès erroné comme celui-ci : A:\\Image.bmp.

4. Exécutez le projet. Parcourez les différents dossiers de votre système afin de sélectionner des fichiers de type Image, c'est-à-dire des fichiers dont l'extension est .bmp, .ico, .cur, et .wmf. Pour une grande sélection d'images, parcourez les dossiers des images incluses avec Visual Basic. Sélectionnez aussi des fichiers qui ne sont pas de type Image et observez le résultat.

5. Afin d'afficher *seulement certains types de fichiers*, modifiez la propriété **Pattern** du contrôle **FileListBox** :

Pattern : *.ico;*.bmp;*.cur;*.wmf

6. Exécutez de nouveau le projet et observez le filtrage dans les fichiers qui sont affichés.

7. Enregistrez tous les fichiers du projet.

6.2 LES BOÎTES DE DIALOGUE STANDARD DE WINDOWS

Dans un projet Visual Basic, il est possible d'utiliser les boîtes de dialogue standard de Windows par l'intermédiaire du contrôle **CommonDialog**. Les boîtes de dialogue suivantes sont disponibles :

Ouvrir pour sélectionner un fichier et l'ouvrir ;

Enregistrer sous pour enregistrer un fichier ;

Impression pour imprimer ;

Police pour sélectionner une police de caractères ;

Couleurs pour sélectionner une couleur.

Ces boîtes de dialogue sont utilisées par la plupart des applications Windows. Par exemple, la boîte de dialogue Ouvrir est utilisée par Microsoft Word et par Microsoft Excell. Les utilisateurs étant généralement familiers avec ces boîtes de dialogue, il est judicieux de les utiliser dans une nouvelle application.

Introduction au contrôle CommonDialog

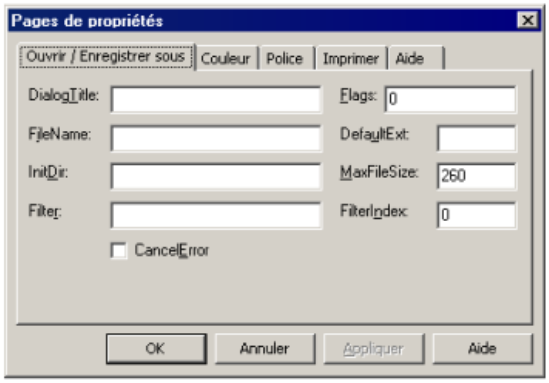
Le contrôle **CommonDialog** ou **Boîtes de dialogue standard** dispose de 5 méthodes **Show** qui permettent d'ouvrir une boîte de dialogue standard de Windows dans une application Visual Basic :

Méthode	Boîte de dialogue standard
ShowOpen	Ouvrir
ShowSave	Enregistrer sous
ShowPrinter	Impression
ShowFont	Police
ShowColor	Couleur

Le préfixe **dlg** identifie le contrôle **CommonDialog** et celui-ci est accessible par le menu **Projet /Composants/ Microsoft Common Dialog Control 6.0**.

Lorsque le contrôle CommonDialog est ajouté à une feuille, la taille de celui-ci n'est pas modifiable et il n'est pas visible à l'exécution.

Les pages de propriétés qui suivent sont accessibles par la propriété (Personnalisé) du contrôle. Il est possible d'y fixer, en mode création, des propriétés pour chacune des boîtes de dialogue.



La boîte de dialogue Couleur

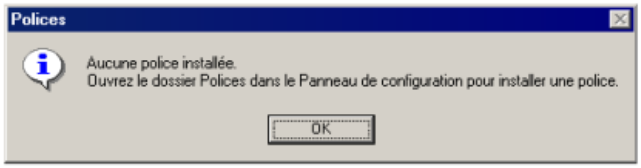
La propriété **Color** définit la couleur sélectionnée dans la boîte de dialogue.

La boîte de dialogue Police

La propriété **Flags** permet de définir les options de la boîte de dialogue Police au moyen de constantes jointes par le caractère +. Quelques constantes :

cdlCFScreenFonts	Affiche les polices écran reconnues par le système.
cdlCFPrinterFonts	Affiche les polices reconnues par l'imprimante.
cdlCFBoth	Affiche les polices de l'écran et de l'imprimante.
cdlCFTTOnly	Affiche seulement les polices TrueType.
cdlCFWYSIWYG	Affiche les polices communes à l'écran et à l'imprimante.
cdlCFEffects	Affiche les options Barré, Souligné et Couleur.
À partir de l'index de l'Aide MSDN, consultez la rubrique CommonDialog, contrôle (boîte de dialogue Police), Propriétés, Flags, pour une liste complète des constantes.	

Attention ! Il faut attribuer au moins l'une des valeurs cdlCFScreenFonts, cdlCFPrinterFonts ou cdlCFBoth à la propriété **Flags** avant d'invoquer la méthode **ShowFont**. Dans le cas contraire, le message suivant est affiché :



Exemple

```
dlgControle.Flags = cdlCFBoth + cdlCFEffects
```

Toutes les polices d'écran et d'imprimante sont disponibles dans la liste Police ainsi que les options Barré, Souligné et Couleur.

Autres propriétés de la boîte de dialogue Police

Color	Couleur de la police sélectionnée. La propriété Flags doit avoir la valeur cdlCFEffects.
FontBold	Style Gras.
FontItalic	Style Italique.
FontStrikethru	Effet Barré. La propriété Flags doit avoir la valeur cdlCFEffects.
FontUnderline	Effet Souligné. La propriété Flags doit avoir la valeur cdlCFEffects.
FontName	Nom de la police sélectionnée.
FontSize	Taille de la police sélectionnée.

La boîte de dialogue Impression

La propriété **Flags** permet de définir des options de la boîte de dialogue Impression au moyen de constantes. Quelques constantes :

cdlPDDisablePrintToFile	La case à cocher Imp. dans fichier n'est pas affichée.
cdlPDNoSelection	Le bouton d'option Sélection est désactivé.
cdlPDPrintSetup	La boîte de dialogue "Configuration de l'impression" est affichée au lieu de la boîte de dialogue Impression.
À partir de l'index de l'Aide MSDN, consultez la rubrique CommonDialog, contrôle (boîte de dialogue Impression), Propriétés, Flags, pour une liste complète des constantes.	

Les boîtes de dialogue Ouvrir et Enregistrer sous

Les boîtes de dialogue Ouvrir et "Enregistrer sous" sont presque identiques. Toutes deux permettent à l'utilisateur de choisir un lecteur, un dossier, un nom de fichier et un type de fichier.

La propriété **Flags** permet de définir des options des boîtes de dialogue Ouvrir et "Enregistrer sous" au moyen de constantes. Quelques constantes :

cdlOFNOverwritePrompt	Spécifie d'afficher un message à l'utilisateur, lui demandant si le fichier doit être remplacé dans le cas où il existe déjà.
cdlOFNPathMustExist	Indique d'afficher un message pour informer l'utilisateur que le chemin d'accès qu'il a spécifié n'est pas valide.
cdlOFNHideReadOnly	Masque la case à cocher "Lecture seule".
À partir de l'index de l'Aide MSDN, consultez la rubrique CommonDialog, contrôle (boîtes de dialogue Ouvrir et Enregistrer sous), Propriétés, Flags, pour une liste complète des constantes.	

La propriété **Filter** des boîtes de dialogue Ouvrir et "Enregistrer sous" permet de préciser un ou plusieurs *types de fichiers* à ouvrir ou à enregistrer en spécifiant leurs extensions. Ces types de fichiers apparaissent dans la liste **Type** des boîtes de dialogue Ouvrir et "Enregistrer sous".

Exemple

Dans la figure qui précède, des fichiers de type .txt et *.* (tous les types) peuvent être ouverts. Ces 2 types ont été déclarés par la ligne de code qui suit :

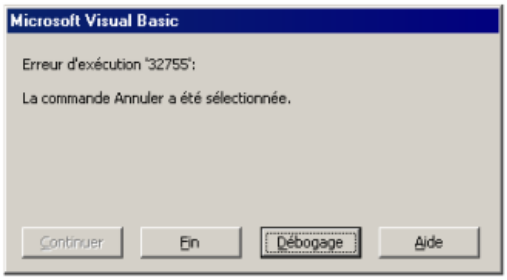
```
dlgDial.Filter = "Fichiers texte(*.txt)|*.txt" & _
                 "|Tous les fichiers(*.*)|*.*"
```

La propriété **Filter** est une chaîne de caractères constituée de la description et du type de tous les types de fichiers. Le caractère | sépare la description d'un type de son extension. Les types de fichiers (description + extension) sont aussi séparés par le caractère |.

La propriété **FileName** retourne le nom du fichier qui a été sélectionné dans la boîte de dialogue Ouvrir et Enregistrer sous. Cette propriété peut aussi être utilisée pour définir un *nom de fichier par défaut* qui est affiché lors de l'ouverture des boîtes de dialogue.

La gestion du bouton Annuler d'une boîte de dialogue standard

Toutes les boîtes de dialogue standard de Windows comportent un bouton Annuler qui permet à l'utilisateur de fermer la boîte de dialogue et ainsi d'annuler toute sélection effectuée dans la boîte de dialogue. Il est très fréquent, lorsque l'utilisateur presse le bouton Annuler, que l'erreur d'exécution suivante soit générée et que par conséquent, l'exécution soit interrompue.



Pour gérer l'erreur précédente, fixez la propriété **CancelError** du contrôle CommonDialog à **True** et interceptez l'erreur par l'instruction **On Error**. Ainsi, à chaque fois que l'utilisateur presse le bouton Annuler, une erreur est automatiquement générée et celle-ci porte un numéro égal à la constante **cdlCancel**.

Exemple

```
Private Sub cmdSelectionCouleur_Click()

On Error GoTo BoutonAnnuler
  dlgDialogue.ShowColor
  .
  .
  Exit Sub
BoutonAnnuler:
  If Err.Number = cdlCancel Then
    Debug.Print "Le bouton Annuler a été pressé"
  Else
    Debug.Print "Une vraie erreur est survenue"
  End If
End Sub
```

Avant d'invoquer la méthode **ShowColor**, un gestionnaire d'erreurs est activé par l'instruction **On Error**. Aussitôt qu'une erreur survient ou que le bouton Annuler est pressé, un branchement est effectué à l'étiquette BoutonAnnuler: et l'erreur peut ainsi être traitée.

Exercice 19 Créer le projet *Boîtes de dialogue standard*

Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre6 /Exercice19 /Debut dans un nouveau dossier .../Visual Basic /Exercice 19 Boites standards.
2. Exécutez le fichier

   .../Visual Basic /Exercice 19 Boites standards /**BoiteStandard.exe**.
3. Testez chacun des boutons et observez les différentes boîtes de dialogue standard de Windows qui sont affichées. Pour ne pas imprimer le contenu de la zone de texte, sélectionnez le bouton Annuler dans la boîte de dialogue Impression.
4. Dans Visual Basic, ouvrez le projet **BoiteStandard.vbp** stocké dans le dossier .../Visual Basic /Exercice 19 Boites standards.
5. Ajoutez un contrôle **CommonDialog** dans la boîte à outils par Projet /Composants /Microsoft Common Dialog Control 6.0.
6. Ajoutez un contrôle CommonDialog sur la feuille et nommez-le **dlgDialogue**.
7. Afin de gérer le bouton Annuler de chacune des boîtes de dialogue, fixez la valeur de la propriété **CancelError** du contrôle dlgDialogue à **True**.

Étape 2 La méthode ShowColor

1. Codez le bouton cmdQuitter qui procède au déchargement de la feuille.
2. Codez la procédure Click du bouton cmdCouleur :

```
Private Sub cmdCouleur_Click()
  On Error GoTo BoutonAnnuler
  With dlgDialogue
    .Color = txtZoneTexte.ForeColor 'couleur sélectionnée
    .ShowColor
    txtZoneTexte.ForeColor = .Color 'nouvelle couleur appliquée
  End With
  Exit Sub
BoutonAnnuler:
  If Err.Number <> cdlCancel Then
    MsgBox "La couleur ne peut être modifiée.", , "Erreur"
  End If
End Sub
```

La ligne de code `.Color = txtZoneTexte.ForeColor` sélectionne la couleur actuelle du texte afin que celle-ci soit la couleur par défaut lors de l'ouverture de la boîte de dialogue Couleur.

La méthode **ShowColor** du contrôle CommonDialog ouvre la boîte de dialogue Couleur de Windows.

Lorsque la boîte de dialogue Couleur est fermée, la nouvelle couleur sélectionnée dans la boîte de dialogue est appliquée au texte de la zone de texte.

Si l'utilisateur presse le bouton Annuler dans la boîte de dialogue, le numéro de l'erreur est vérifié par le gestionnaire d'erreurs activé par l'instruction On Error. Si le numéro d'erreur ne correspond pas à la constante `cdlCancel`, une erreur est survenue et un message d'erreur est affiché à l'utilisateur. Notez que ce message n'est pas obligatoire.

3. Testez le bouton cmdCouleur. Observez la couleur sélectionnée par défaut dans la boîte de dialogue. Testez l'application d'une nouvelle couleur et le bouton Annuler.

Dans une zone de texte, la couleur est appliquée à tout le texte. Pour affecter une couleur à un texte sélectionné, il faut utiliser un contrôle RichTextBox. Ce contrôle est traité dans la section 6.3.

Étape 3 La méthode ShowFont

1. Codez la procédure Click du bouton cmdPolice :

```
Private Sub cmdPolice_Click()
  With dlgDialogue
    .Flags = cdlCFBoth + cdlCFEffects
    .FontName = txtZoneTexte.FontName             'nom de police
    .FontBold = txtZoneTexte.FontBold             'style gras
    .FontItalic = txtZoneTexte.FontItalic         'style italique
    .FontSize = txtZoneTexte.FontSize             'taille
    .FontStrikethru = txtZoneTexte.FontStrikethru 'barré
    .FontUnderline = txtZoneTexte.FontUnderline   'souligné
    .Color = txtZoneTexte.ForeColor               'couleur

    .ShowFont

    txtZoneTexte.FontName = .FontName
    txtZoneTexte.FontBold = .FontBold
    txtZoneTexte.FontItalic = .FontItalic
    txtZoneTexte.FontSize = .FontSize
    txtZoneTexte.FontStrikethru = .FontStrikethru
```

```
    txtZoneTexte.FontUnderline = .FontUnderline
    txtZoneTexte.ForeColor = .Color
  End With
End Sub
```

La propriété **Flags** spécifie d'afficher dans la boîte de dialogue les polices de l'écran et les polices de l'imprimante ainsi que les choix Barré, Souligné et Couleur.

Les lignes de code qui précèdent la méthode **ShowFont** affectent plusieurs valeurs de propriétés de la zone de texte txtZoneTexte au contrôle dlgDialogue. Lors de l'affichage de la boîte de dialogue Police, ces valeurs sont celles qui sont sélectionnées par défaut.

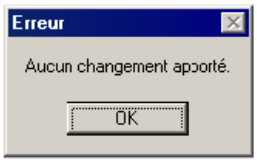
La méthode **ShowFont** du contrôle CommonDialog ouvre la boîte de dialogue Police de Windows.

Les instructions qui suivent la méthode **ShowFont**, affectent les valeurs des propriétés sélectionnées dans la boîte Police à la zone de texte txtZoneTexte.

2. Testez le bouton cmdPolice. Sélectionnez une police, un style, une taille, une couleur et cochez les cases Barré et Souligné. Pressez le bouton OK et observez le résultat dans la zone de texte.

3. Testez le bouton cmdPolice une seconde fois et observez les propriétés sélectionnées par défaut lors de l'affichage de la boîte de dialogue Police. Cette fois, pressez le bouton Annuler. L'erreur d'exécution 32755 devrait survenir...

4. En vous fiant sur le code de programmation de la procédure `cmdCouleur_Click`, gérez le bouton Annuler pour intercepter l'erreur. Si une erreur autre que le bouton Annuler survient, affichez le message suivant :



5. Testez le bouton Annuler de la boîte de dialogue Police.

Étape 4 La méthode ShowPrinter

1. Codez la procédure Click du bouton cmdImprimer :

```
Private Sub cmdImprimer_Click()
On Error GoTo BoutonAnnuler
  With dlgDialogue
    .Flags = cdlPDNoSelection + cdlPDHidePrintToFile
    .ShowPrinter
  End With

  Printer.Print txtZoneTexte
  Printer.EndDoc
  MsgBox "Le document est en cours d'impression", , "Impression"
  Exit Sub
BoutonAnnuler:
  MsgBox "Impression annulée.", , ""
End Sub
```

La propriété **Flags** spécifie de ne pas afficher l'option Sélection et de masquer la case à cocher "Imp. dans fichier" dans la boîte de dialogue Impression. Notez que la propriété Flags n'est pas obligatoire, elle est utilisée ici, à titre d'exemple.

La méthode **ShowPrinter** du contrôle CommonDialog ouvre la boîte de dialogue Impression de Windows.

Si l'utilisateur presse le bouton Annuler ou si une erreur survient, le message "Impression annulée". est affiché.

La méthode **Print** de l'objet **Printer** permet d'imprimer le contenu de la zone de texte. Cependant, elle ne reconnaît pas le formatage du texte (couleur, barré, souligné, etc.). Dans l'exercice 21, une méthode d'impression qui reconnaît le formatage sera utilisée.

La méthode **EndDoc** envoie le document à l'imprimante.

Les méthodes Print et EndDoc de l'objet Printer sont décrites dans la section Les principales méthodes de l'objet Printer, page 369.

2. Testez le bouton cmdImprimer. Dans un premier temps, imprimez la zone de texte et dans un deuxième temps, pressez le bouton Annuler dans la boîte de dialogue et observez le résultat.

Étape 5 La méthode ShowSave

1. Codez la procédure Click du bouton cmdEnrSous :

```
Private Sub cmdEnrSous_Click()
  With dlgDialogue
    .Flags = cdlOFNOverwritePrompt + cdlOFNPathMustExist
    .Filter = "Fichiers texte (*.txt)|*.txt"
    .ShowSave
  End With
  Open dlgDialogue.FileName For Output As #1
  Print #1, txtZoneTexte
  Close #1
End Sub
```

La propriété **Flags** spécifie d'afficher une confirmation de remplacement, si le fichier existe déjà, et un message si le chemin d'accès spécifié n'est pas valide. Notez que la propriété Flags n'est pas obligatoire, elle est utilisée ici, à titre d'exemple.

La propriété **Filter** spécifie que seulement le type .txt doit apparaître dans la liste Type de la boîte de dialogue Enregistrer sous.

La méthode **ShowSave** du contrôle CommonDialog ouvre la boîte de dialogue "Enregistrer sous" de Windows.

L'instruction **Open** ouvre le fichier qui doit être enregistré, l'instruction **Print** enregistre le fichier et l'instruction **Close** ferme le fichier.

Les instructions Open, Print et Close sont détaillées dans la section Les fichiers et fonctions d'accès aux fichiers, page 355.

2. Testez le bouton cmdEnrSous comme suit :
   - Déroulez la liste Type et observez son contenu ;
   - Dans la zone "Nom de fichier", entrez le nom **Zone de texte** (sans extension) ;
   - Sélectionnez le dossier .../Visual Basic /Exercice 19 Boites standards ;
   - Pressez le bouton Enregistrer.
3. À partir de l'Explorateur de Windows, double-cliquez sur le fichier **Zone de texte.txt**. L'application Bloc-notes devrait démarrer. Observez le contenu du fichier et assurez-vous que la zone de texte a été enregistrée intégralement.

4. Testez le bouton cmdEnrSous une seconde fois et pressez le bouton Annuler. Une erreur 32755 devrait être générée. Apportez les changements nécessaires pour afficher le message qui suit lorsque le bouton Annuler est pressé ou lorsque toute autre erreur survient. Testez le bouton Annuler.

Étape 6 La méthode ShowOpen

1. Codez la procédure du bouton cmdOuvrir :

```
Private Sub cmdOuvrir_Click()
On Error GoTo BoutonAnnuler
  With dlgDialogue
    .Filter = "Fichiers texte (*.txt)|*.txt|" & _
              "Tous les fichiers(*.*)|*.*"
    .FileName = ""
    .ShowOpen
  End With
  Open dlgDialogue.FileName For Input As #1
  txtZoneTexte = Input(LOF(1), #1)
  Close #1
  Exit Sub
BoutonAnnuler:
  txtZoneTexte.Text = "«Aucun fichier n'a été ouvert»"
End Sub
```

La propriété **Filter** spécifie que les types de fichiers *.txt et *.* apparaissent dans la liste Type de la boîte de dialogue Ouvrir.

La propriété **FileName** spécifie de ne pas afficher de nom de fichier par défaut dans la zone "Nom de fichier" de la boîte de dialogue Ouvrir.

La méthode **ShowOpen** du contrôle CommonDialog ouvre la boîte de dialogue Ouvrir de Windows.

La fonction **LOF** (Length of File, longueur du fichier) dans l'instruction **Input**, indique de lire tout le contenu du fichier et de le placer dans la zone de texte txtZoneTexte.

La fonction LOF et l'instruction Input sont décrites dans la section Les instructions et fonctions d'accès aux fichiers, page 355.

2. Testez le bouton cmdOuvrir. Dans un premier temps, pressez le bouton Annuler et observez le message affiché dans la zone de texte. Dans un deuxième temps, testez comme suit :
   - Déroulez la liste Type et observez son contenu ;
   - Sélectionnez le fichier **Zone de texte.txt** qui a été enregistré au numéro 2 de l'étape 5 ;
   - Pressez le bouton Ouvrir ;
   - Observez le contenu de la zone de texte.
3. Enregistrez tous les fichiers du projet.

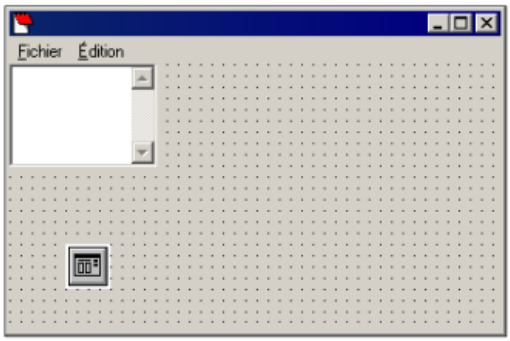
Exercice 20 Programmer un éditeur de texte

L'exercice 20 vise à créer un petit **éditeur de texte** ressemblant un peu au Bloc-Notes de Windows.

Étape 1 Préparation et enregistrement

Dans Visual Basic, ouvrez un nouveau projet. Nommez le projet **Editeur** et la feuille **frmEditeur**. Enregistrez la feuille dans le fichier **Editeur.frm** d'un *nouveau dossier* .../Visual Basic /Exercice 20 Editeur de texte et le projet dans le fichier **Editeur.vbp** du même dossier.

Étape 2 Conception de l'interface graphique du projet



1. Fixez les propriétés de **frmEditeur** :
   - **Height** = 5145 ;
   - **Width** = 7875 ;
   - la feuille est centrée dans l'écran au démarrage.
2. Ajoutez une zone de texte **txtEdition** et effacez son contenu. La propriété **MultiLine** doit avoir la valeur **True** et la propriété **ScrollBars** doit être **2-vertical**.
3. Ajoutez un contrôle **CommonDialog**, nommez-le **dlgEdition** et affectez la valeur **True** à la propriété **CancelError**.
4. Ajoutez la procédure suivante :

```
Private Sub Form_Resize()
  txtEdition.Height = frmEditeur.Height
  txtEdition.Width = frmEditeur.Width - 100
End Sub
```

> La taille de la zone de texte est ajustée avec celle de la forme à chaque fois que la fenêtre est redimensionnée. La largeur de la zone est réduite de 100 par rapport à la forme afin que la barre de défilement à droite soit mieux dégagée.

5. Affichez le titre dans la barre de titre de la feuille :
   - Déclarez la constante de portée module qui suit :

```
Const cmstrTitre As String = _
          "Nouveau document - Éditeur de texte"
```

   - Traduisez le pseudo-code :

```
Private Sub Form_Load()
  Affichez le contenu de la constante cmstrTitre dans la barre de titre de
  la feuille.
  Effacez le contenu du Presse-papiers.
End Sub
```

6. Créez le menu :

Caption	Name	ShortCut
&Fichier	mnuFichier	
…&Nouveau	mnuFichierNouveau	Ctrl+N
…&Ouvrir…	mnuFichierOuvrir	Ctrl+O
…-	Sep1	

...&Enregistrer	mnuFichierEnr	Ctrl+S
...En&registrer sous...	mnuFichierEnrSous	
...-	Sep2	
...&Imprimer...	mnuFichierImp	Ctrl+P
...-	Sep3	
...&Quitter	mnuFichierQuitter	Ctrl+Q
&Edition	mnuEdition	
...&Couper	mnuEditionCouper	Ctrl+X
...Co&pier	mnuEditionCopier	Ctrl+C
...C&oller	mnuEditionColler	Ctrl+V
...-	Sep	
...&Rechercher...	mnuEditionRech	Ctrl+R
...Sui&vant (non accessible)	mnuEditionRechSuiv	F3

7. Codez le choix Quitter au menu qui procède au déchargement de la feuille.
8. Codez le choix Édition au menu. Activez les choix Couper et Copier seulement s'il y a un *texte sélectionné* dans la zone de texte. Activez le choix Coller seulement si le presse-papiers contient du texte. Codez aussi les choix Copier, Couper et Coller du menu Édition.

Ces opérations ont déjà été codées dans l'exercice 12.

9. Codez le choix Rechercher au menu :
   - Déclarez une variable de portée module pour stocker la chaîne recherchée dans le texte :

     ```
     Dim mstrRechercher As String
     ```

   - Codez la procédure suivante :

     ```
     Private Sub mnuEditionRech_Click()
       Dim intPosTrouve As Integer
       mstrRechercher = InputBox("Chaine recherchée :", _
                                 "Recherche")
       If mstrRechercher = "" Then
         Exit Sub
       End If
       intPosTrouve = InStr(1, txtEdition.Text, mstrRechercher, 1)
       If intPosTrouve = 0 Then
         MsgBox "Recherche infructueuse!", , "Recherche"
       Else
         txtEdition.SelStart = intPosTrouve - 1
         txtEdition.SelLength = Len(mstrRechercher)
         mnuEditionRechSuiv.Enabled = True
       End If
     End Sub
     ```

La fonction **InputBox** retourne une chaîne de longueur nulle ("") lorsque l'utilisateur utilise le bouton Annuler ou lorsque celui-ci n'a rien inscrit.

La fonction **Instr** recherche la présence de la chaîne recherchée dans la zone de texte et retourne la *position* de la chaîne trouvée (réf. Les fonctions de chaînes de caractères).

En supposant que la chaîne trouvée est à la position 17, le point d'insertion est placé à la position 16 (`intPosTrouve - 1`) et la chaîne est sélectionnée dans le texte. Elle apparaît ainsi en surbrillance.

Le choix au menu Suivant est activé si la recherche a été fructueuse.

- Testez et effectuez une recherche fructueuse et une recherche infructueuse. Testez aussi le bouton Annuler.

10. Codez le choix Suivant au menu :

```
Private Sub mnuEditionRechSuiv_Click()
  Dim intPosTrouve As Integer
  Dim intPosDebut As Integer
  intPosDebut = txtEdition.SelStart + 2
  intPosTrouve = InStr(intPosDebut, txtEdition.Text, _
                                    mstrRechercher, 1)
  If intPosTrouve = 0 Then
    MsgBox "La recherche est terminée.", , "Suivant"
    mnuEditionRechSuiv.Enabled = False
  Else
    txtEdition.SelStart = intPosTrouve - 1
    txtEdition.SelLength = Len(mstrRechercher)
  End If
End Sub
```

Suite à une première recherche, le point d'insertion est placé à la position 16 et la chaîne trouvée débute à la position 17. La seconde recherche doit débuter à la position 18 (`txtEdition.SelStart + 2`).

Lorsque la fonction InStr retourne la valeur 0, aucune chaîne n'a été trouvée et la recherche est terminée. Le choix au menu Suivant redevient alors inaccessible.

11. Testez des recherches avec plusieurs occurrences d'une chaîne de recherche. Observez l'accessibilité du choix Suivant au menu.

12. Codez le choix Fichier au menu :

    - Déclarez 2 variables booléennes de portée module :

```
Dim mblnModifie As Boolean
Dim mblnAnnuler As Boolean
```

mblnModifie	Spécifie si oui (True) ou non (False) le texte a été modifié depuis le dernier enregistrement.
mblnAnnuler	Spécifie si oui (True) ou non (False) l'utilisateur a pressé le bouton Annuler dans la boîte de dialogue Enregistrer sous.

    - Dans la procédure événementielle txtEdition_Change, fixez la valeur True à la variable mblnModifie.

> À chaque fois qu'une modification est effectuée dans la zone de texte, l'événement Change est déclenché et la valeur True de la variable mblnModifie spécifie que le texte a été modifié.

13. Complétez la procédure Click du choix "Enregistrer sous" du menu Fichier. Traduisez le pseudo-code en Visual Basic et entrez les lignes de code Visual Basic.

```
Private Sub mnuFichierEnrSous_Click()

Activez un gestionnaire d'erreurs. (réf. Exercice 19)
  Utilisez la méthode ShowSave du contrôle CommonDialog pour enregistrer la
  zone de texte dans un fichier.
  Spécifiez le type de fichier "Fichiers texte (*.txt)|*.txt".
  Précisez qu'un message de remplacement doit être affiché lorsqu'un
  fichier existe déjà. (réf. Exercice 19)

  frmEditeur.Caption = "Fichier en cours : " & dlgEdition.FileName
  mblnModifie = False
  mblnAnnuler = False
  Exit Sub

Nom de l'étiquette du gestionnaire d'erreurs
  Si l'utilisateur a pressé le bouton "Annuler", affectez la valeur True à
  la variable mblnAnnuler.
End Sub
```

Après avoir enregistré la zone de texte, la propriété **FileName** du contrôle dlgEdition retourne le nom du fichier en cours avec son chemin d'accès. Ce dernier est affiché dans la barre de titre de la feuille.

La variable mblnModifie prend la valeur False puisque le texte a été enregistré et n'a évidemment subi aucune modification depuis l'enregistrement.

La modification de la variable mblnAnnuler est expliquée un peu plus loin dans l'exercice.

14. Testez le choix Enregistrer sous. Entrez du texte dans l'éditeur et enregistrez-le. Après avoir enregistré, observez le contenu de la barre de titre. Testez le bouton Annuler dans la boîte de dialogue Enregistrer sous.

15. Codez le choix Nouveau du menu Fichier :

- Déclarez les constantes et la variable de portée module qui suivent :

```
Const cmintBoutons As Integer = vbYesNoCancel + vbExclamation _
                              + vbDefaultButton3
Const cmstrMessage As String = "Voulez-vous enregistrer" & _
                               " le document actuel ?"
Dim mintReponse As Integer
```

- Ajoutez la procédure suivante :

```
Private Sub mnuFichierNouveau_Click()
  If mblnModifie = True Then   'modification depuis dernier enr?
    mintReponse = MsgBox(cmstrMessage, cmintBoutons, "Éditeur")
    Select Case mintReponse
      Case vbYes
        Call mnuFichierEnr_Click
        If mblnAnnuler = True Then
          Exit Sub
        End If
      Case vbCancel
        Exit Sub
    End Select
  End If
  txtEdition.Text = ""
  mblnModifie = False
  frmEditeur.Caption = cmstrTitre
  dlgEdition.FileName = ""
End Sub
```

Si le choix Nouveau est sélectionné et que le document en cours n'a pas été enregistré, le message "Voulez-vous enregistrer le document actuel?" est affiché. Le message comporte 3 boutons de réponse : Oui, Non et Annuler (bouton par défaut).

Si l'utilisateur désire enregistrer le document en cours (vbYes), la procédure du choix Enregistrer est appelée. Par la suite, la variable mblnAnnuler est vérifiée au cas où l'utilisateur aurait sélectionné le bouton Annuler dans la boîte de dialogue Enregistrer sous. Dans ce cas, l'instruction Exit Sub permet de sortir de la procédure afin que les instructions qui suivent ne soient pas exécutées.

Si l'utilisateur choisit le bouton Annuler en réponse au message (vbCancel), l'instruction Exit Sub est aussi utilisée pour sortir de la procédure.

Si l'utilisateur choisit le bouton Non ou à la suite d'un enregistrement, les 4 dernières instructions de la procédure *initialisent le nouveau document*. Par exemple, la zone de texte est effacée et la barre de titre spécifie qu'il s'agit d'un nouveau document.

16. Avant de testez, *mettez en commentaires les 4 lignes de code* du cas vbYes. Testez le choix Nouveau avec un document en cours modifié et non enregistré. À la suite du message, sélectionnez les boutons Annuler et Non.

17. Codez le choix Enregistrer du menu Fichier :

```
Private Sub mnuFichierEnr_Click()
  If frmEditeur.Caption = cmstrTitre Then
    Call mnuFichierEnrSous_Click
  Else
    Open dlgEdition.FileName For Output As #1
    Print #1, txtEdition.Text
    Close #1
    mblnModifie = False
  End If
End Sub
```

Si la barre de titre contient "Nouveau document...", la procédure du choix "Enregistrer sous" est appelée parce que le texte *n'a pas encore été sauvegardé* et que l'utilisateur doit au moins fournir un nom de fichier dans la boîte de dialogue Enregistrer sous.

Si le fichier a déjà été enregistré une première fois, il est enregistré de nouveau par les instructions Open, Print et Close.

18. Testez le choix Enregistrer avec un texte qui a déjà été enregistré et avec un autre texte qui n'a jamais été enregistré.

19. Ne plus commentez les 4 lignes de code de la procédure du choix Nouveau et testez le cas où le texte en cours a été modifié mais non enregistré. Répondez Oui au message d'enregistrement et assurez-vous que le texte en cours est bien enregistré avant de produire un nouveau document.

20. Codez le choix Imprimer au menu Fichier en vous référant au code de programmation du bouton Imprimer de l'exercice 19.

21. Ajoutez le code de la procédure Click du choix Ouvrir du menu Fichier :

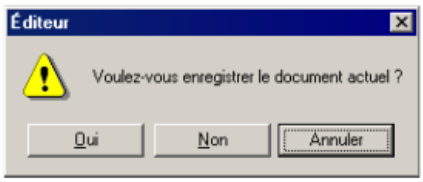
```
Private Sub mnuFichierOuvrir_Click()
  On Error GoTo GestionErreur
  If mblnModifie = True Then
    mintReponse = MsgBox(cmstrMessage, cmintBoutons, "Editeur")
    Select Case mintReponse
      Case vbYes
        Call mnuFichierEnr_Click
        If mblnAnnuler = True Then
          Exit Sub
        End If
      Case vbCancel
        Exit Sub
    End Select
  End If
  dlgEdition.Filter = "Fichiers texte(*.txt)|*.txt|" & _
                      "Tous les fichiers(*.*)|*.*"
  dlgEdition.FileName = ""
  dlgEdition.ShowOpen
  Open dlgEdition.FileName For Input As #1
  txtEdition.Text = Input(LOF(1), 1)
  Close #1
  mblnModifie = False
  Me.Caption = "Fichier en cours : " & dlgEdition.FileName
  Exit Sub
GestionErreur:
  If Err.Number <> cdlCancel Then
    MsgBox "Aucun fichier n'a été ouvert", , "Erreur"
  End If
End Sub
```

Lorsque l'utilisateur désire ouvrir un fichier, il faut d'abord enregistré tout texte en cours modifié qui n'est pas enregistré. Par la suite, le fichier est ouvert à partir de la boîte de dialogue standard Ouvrir et chargé dans la zone de texte txtEdition. La barre de titre affiche le nom et le chemin d'accès du fichier ouvert.

22. Testez le choix Ouvrir lorsqu'un document est en cours et qu'il n'est pas enregistré. Testez aussi le cas où il n'y a pas de document en cours.

23. Codez la procédure de fin de programme Form_Unload :

- Lorsque l'utilisateur ferme la feuille par le bouton Fermer de la barre de titre, vérifiez d'abord si le document en cours a été modifié depuis le dernier enregistrement. Si c'est le cas, affichez le message suivant :



- Codez les réponses au message comme suit :

```
Case vbYes
  Call mnuFichierEnr_Click
  If mblnAnnuler = True Then
    Cancel = True
  End If
Case vbCancel
  Cancel = True
```

Si l'utilisateur désire enregistrer et que par la suite, il choisit le bouton Annuler dans la boîte de dialogue Enregistrer sous, la fermeture de la feuille est annulée (Cancel = True).

Si l'utilisateur choisit le bouton Annuler à la suite du message, la fermeture de la feuille est aussi annulée.

Dans les autres cas, le fichier est enregistré au besoin et la feuille est déchargée de la mémoire.

24. Testez la fermeture de la feuille par le choix Quitter au menu Fichier et par le bouton Fermer de la barre de titre. Testez le cas où un document en cours n'est pas enregistré.

25. Avant de terminer, réorganisez le code de programmation et placez les procédures dans un ordre logique :

    Form_Load

    Form_Resize

    mnuFichierNouveau_Click

    mnuFichierOuvrir_Click

    mnuFichierEnr_Click

    mnuFichierEnrSous_Click

    mnuFichierImp_Click

    mnuEdition_Click

    mnuEditionCouper_Click

    mnuEditionCopier_Click

    mnuEditionColler_Click

    mnuEditionRech_Click

    mnuEditionRechSuiv_Click

    txtEdition_Change

    Form_Unload

    mnuFichierQuitter_Click

6.3 LE CONTRÔLE RICHTEXTBOX

Le contrôle **RichTextBox** ou **Éditeur RTF** permet à l'utilisateur d'entrer ou de modifier du texte comme dans une zone de texte. Le contrôle RichTextBox supporte toutes les propriétés, méthodes et événements du contrôle TextBox. De plus, il permet de fixer des attributs de formatage comme la police, la couleur ou le style gras à un *texte sélectionné*, contrairement au contrôle TextBox qui ne permet que le formatage de *tout le texte*. Le contrôle TextBox permet de créer un éditeur comme le Bloc-Notes de Windows tandis que le contrôle RichTextBox offre les possibilités pour créer un éditeur de texte semblable au WordPad de Windows.

Le préfixe **rtf** (**R**ich **T**ext **F**ormat) identifie le contrôle **RichTextBox** et celui-ci est accessible par le menu **Projet /Composants/ Microsoft Rich Textbox Control 6.0**.

Le format **Rich Text Format** est un standard pour l'enregistrement de textes avec certains attributs de formatage. Le contrôle RichTextBox peut traiter des fichiers de format RTF (.rtf) et des fichiers texte standard (.txt).

Le contrôle **RichTextBox** génère un événement **SelChange** lorsque la sélection d'un texte a été modifiée ou que le point d'insertion a été déplacé dans le texte.

Les principales propriétés du contrôle RichTextBox

Les propriétés permettent par exemple de mettre du texte en gras ou en italique, de modifier la couleur, et de créer des *exposants* et des *indices*. Elles permettent aussi la *mise en retrait* de paragraphes et l'insertion de *puces*.

Consultez l'aide MSDN pour plus de détails sur la mise en retrait, l'insertion de puces, les exposants et les indices.

AutoVerbMenu	Active oui ou non le menu contextuel du contrôle RichTextBox. Ce menu comprend les choix Annuler, Couper, Copier, Coller et Supprimer. Par défaut, le menu contextuel n'est pas accessible à l'utilisateur.
FileName	Identifie par une chaîne de caractères le nom et le chemin d'accès du fichier où est stocké le texte contenu dans le contrôle RichTextBox.
ScrollBars	Spécifie si le contrôle comporte des barres de défilement horizontale, verticale ou les deux. Par défaut, aucune barre n'est affichée. Pour s'assurer qu'un long texte puisse être visualisé, modifiez cette propriété en mode création.
SelBold, **SelItalic** **SelStrikethru** **SelUnderline**	Spécifient le style de la police de caractères d'un texte sélectionné (**Gras**, *Italique*, ~~Barré~~ et Souligné).*
SelColor	Spécifie la couleur d'un texte sélectionné.*
SelFontName	Spécifie le nom de la police de caractères d'un texte sélectionné.*
SelFontSize	Spécifie la taille de la police de caractères d'un texte sélectionné.*
SelLength	Spécifie le nombre de caractères sélectionnés dans le texte ou qui doivent être sélectionnés dynamiquement.
SelStart	Spécifie la position de départ d'un texte sélectionné ou la position du point d'insertion dans le texte si aucun texte n'est sélectionné.
SelText	Chaîne de caractères qui représente le *texte actuellement sélectionné*. Si aucun texte n'est sélectionné, sa valeur est égale à une chaîne de longueur nulle ("").
Text	Chaîne de caractères qui représente *tout le texte* du contrôle (idem TextBox).
*Lorsqu'un texte n'est pas sélectionné dans le contrôle RichTextBox, la propriété sera affectée à tout nouveau texte tapé après le point d'insertion en cours.	

Les principales méthodes du contrôle RichTextBox

Find	Recherche une chaîne de caractères spécifiée dans le contrôle RichTextBox. Cette méthode offre des options comme par exemple : Respect de la casse ou Recherche sur un mot entier. Pour plus de détails, consultez l'aide MSDN.
LoadFile *nom* [*type*]	Ouvre un fichier RTF ou un fichier texte standard et charge son contenu dans le contrôle RichTextBox, remplaçant ainsi son contenu antérieur. L'argument *nom* est une chaîne qui spécifie le chemin d'accès et le nom du fichier. L'argument *type* peut prendre la valeur 0 ou 1. Le type 0 spécifie un fichier RTF (par défaut) et le type 1 spécifie un fichier texte standard. Les constantes rtfRTF et rtfText peuvent aussi être utilisées.
SaveFile *nom* [*type*]	Enregistre dans un fichier le contenu du contrôle RichTextBox. Les arguments *nom* et *type* sont semblables à ceux de la méthode **LoadFile**.
SelPrint(Printer.*hDC***)**	Permet d'imprimer tout le texte du contrôle ou une partie du texte si un texte est sélectionné. Le texte est envoyé à l'objet Printer dont la propriété hDC identifie le périphérique imprimante.

Exercice 21 Modifier le projet *Boîtes de dialogue standard*

Étape 1 Préparation

1. À partir de l'Explorateur de Windows, copiez tous les fichiers de votre dossier .../Visual Basic /Exercice 19 Boites standards dans un nouveau dossier .../Visual Basic /Exercice 21 RichTextBox.
2. Dans Visual Basic, ouvrez le projet **BoiteStandard.vbp** stocké dans le dossier .../Visual Basic /Exercice 21 RichTextBox.

Étape 2 Remplacer un contrôle TextBox par un contrôle RichTextBox

1. Remplacez le contrôle TextBox par un contrôle RichTextBox :
   - Enlevez la zone de texte txtZoneTexte ;
   - Ajoutez un contrôle RichTextBox à la boîte à outils ;
   - Ajoutez un contrôle RichTextBox à la feuille, de la même taille que l'ancien contrôle TextBox ;
   - Nommez le nouveau contrôle **rtfZoneTexte** et fixez la valeur 3-rtfBoth à sa propriété **ScrollBars**.
2. Dans la fenêtre de code, modifiez tous les noms txtZoneTexte pour rtfZoneTexte.

Étape 3 Formater des sélections de texte

1. Exécutez le projet et sélectionnez le bouton Couleur. Corrigez les erreurs de compilation en remplaçant la propriété ForeColor par SelColor.
2. Exécutez le projet, entrez du texte et sélectionnez une portion du texte. Choisissez le bouton Couleur et la couleur rouge. Observez le résultat.
3. Dans le contrôle RichTextBox, sélectionnez un texte *en noir et en rouge* et pressez le bouton Couleur. Une erreur devrait être générée.

> L'opération précédente génère une erreur parce que le texte contient au moins 2 définitions de couleur, et il devient impossible d'afficher une couleur par défaut dans la boîte de dialogue Couleur.
>
> Lorsqu'un texte sélectionné contient plusieurs couleurs (**SelColor**), plusieurs styles (**SelBold**, **SelUnderline**, **SelStrikethru**, **SelItalic**) ou plusieurs polices (**SelFontName**, **SelFontSize**), la valeur de la propriété est égale à **Null**. En vérifiant cette valeur avant de faire appel aux boîtes de dialogue Couleur et Police, il est ainsi possible d'intercepter l'erreur.

4. Modifiez les lignes de code en gras dans la procédure du bouton Couleur :

```
Private Sub cmdCouleur_Click()
  .
  With dlgDialogue
    If IsNull(rtfZoneTexte.SelColor) = False Then
      .Color = rtfZoneTexte.SelColor
    End If
    .ShowColor
    rtfZoneTexte.SelColor = .Color
  End With
  rtfZoneTexte.SetFocus     (Ajoutez cette ligne !)
  .
  .
```

5. Testez la dernière correction avec des sélections de texte à plusieurs couleurs.
6. Modifiez la procédure du bouton Police en vous basant sur les modifications apportées dans la procédure du bouton Couleur. Tenez compte des styles, couleurs, tailles et noms de polices multiples. Redonnez le focus au contrôle rtfZoneTexte lors de la fermeture de la boîte de dialogue Police. Testez minutieusement ces changements.

Étape 4 Utiliser les méthodes SaveFile et LoadFile

1. Modifiez la procédure du bouton Enregistrer sous :
   - Enlevez les instructions Open, Print et Close ;
   - Remplacez les lignes enlevées par la ligne de code qui suit ;

     ```
     rtfZoneTexte.SaveFile dlgDialogue.FileName
     ```

     > La méthode **SaveFile** enregistre le contenu du contrôle RichTextBox dans le fichier désigné dans la boîte de dialogue Enregistrer sous (`dlgDialogue.FileName`).

   - La propriété **Filter** du contrôle dlgDialogue doit contenir la chaîne qui suit.

     ```
     "Fichiers formatés (*.rtf)|*.rtf"
     ```

2. Modifiez la procédure du bouton Ouvrir :
   - Enlevez les instructions Open, Input et Close ;
   - Remplacez les lignes enlevées par la ligne de code qui suit :

     ```
     rtfZoneTexte.LoadFile dlgDialogue.FileName
     ```

     > La méthode **LoadFile** ouvre un fichier et charge son contenu dans le contrôle RichTextBox. Le fichier ouvert est celui spécifié dans la boîte de dialogue Ouvrir (`dlgDialogue.FileName`).

   - La zone **Type** de la boîte de dialogue Ouvrir doit afficher les types de fichiers comme suit :

3. Testez les boutons Ouvrir et Enregistrer sous.
4. Modifiez la procédure du bouton Imprimer :
   - Enlevez les lignes de code `Printer.Print rtfZoneTexte` et `Printer.EndDoc` ;
   - Remplacez les lignes enlevées par la ligne de code qui suit :

     ```
     rtfZoneTexte.SelPrint (Printer.hDC)
     ```

La méthode **SelPrint** dirige le contenu du contrôle RichTextBox vers l'objet **Printer** et reconnaît les textes formatés.

5. Testez le bouton Imprimer.

6.4 LES BARRES D'OUTILS ET LES BARRES D'ÉTAT

Visual Basic dispose de contrôles qui permettent de créer des **barres d'outils** et des **barres d'état**. Le contrôle **ToolBar** permet de créer des barres d'outils, conjointement avec le contrôle **ImageList** qui définit les images apparaissant sur les boutons de la barre d'outils. Le contrôle **StatusBar** permet de créer des barres d'état.

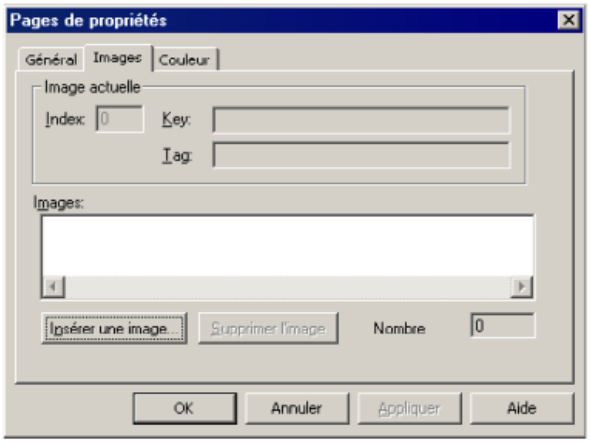
Le contrôle *ImageList*

Le contrôle **ImageList** ou **Liste d'images** contient une collection d'objets **ListImage** qui sont identifiés par un *index* ou une *clé*. La propriété **Index** est une valeur numérique qui spécifie la *position* d'une image dans la liste et la propriété **Key** est une chaîne de caractères qui représente le *nom* d'une image.

Le contrôle **ImageList** est un conteneur d'images qui est utilisé conjointement avec un autre contrôle, comme par exemple, le contrôle **ToolBar**. Le contrôle ImageList est lié à un second contrôle par la propriété **ImageList** de ce dernier.

Le préfixe **ils** identifie le contrôle **ImageList** et celui-ci est accessible par le menu **Projet /Composants/ Windows Common Control 6.0**.

La taille du contrôle ImageList n'est pas modifiable et le contrôle n'est pas visible en mode exécution. En mode création, pour ajouter les images dans la liste, utilisez la fenêtre Pages de propriétés, accessible par la propriété Personnalisé.



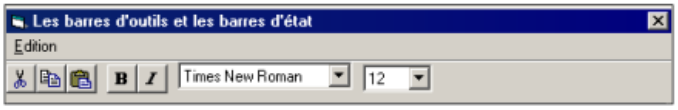
Pour ajouter une image dans la liste :

- Sélectionnez l'onglet **Images** ;
- Cliquez sur le bouton "Insérer une image" ;
- Parcourez les dossiers et sélectionnez un fichier image ;
- Cliquez sur le bouton Ouvrir ;
- Une première image est insérée et sa propriété **Index** = 1 ;
- Pour nommer une image, entrez un nom dans la zone Key.

Le contrôle *ToolBar*

Le contrôle **ToolBar** ou **Barre d'outils** contient une collection **Buttons** formée d'objets **Button**, chaque objet représentant un *bouton* de la barre d'outils. Un bouton peut comporter du texte ou/et une image. Les images sont déclarées dans un contrôle **ImageList** lié au contrôle ToolBar.

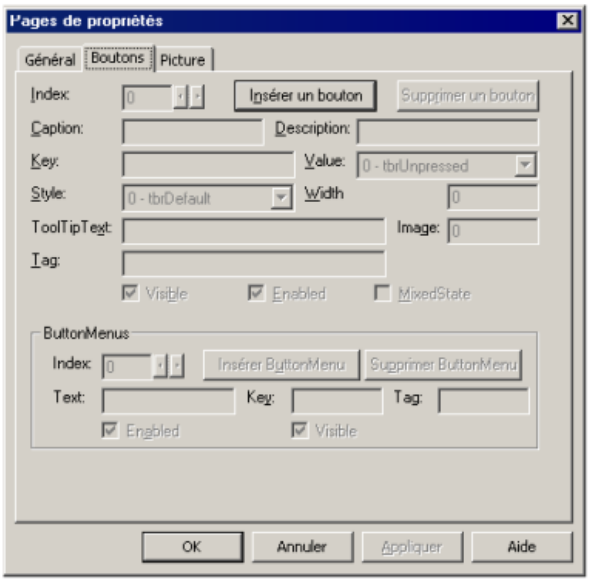
Exemple d'un contrôle ToolBar



Le préfixe **tlb** identifie le contrôle **ToolBar** et celui-ci est accessible par le menu **Projet /Composants/ Windows Common Control 6.0**.

Lorsqu'un contrôle ToolBar est ajouté à une feuille, il est automatiquement placé dans le haut de la feuille et ajusté à la largeur de la feuille.

En mode création, pour créer les différents boutons de la barre d'outils, utilisez la fenêtre Pages de propriétés, accessible par la propriété Personnalisé :



Pour ajouter un objet **Button** dans la barre d'outils :

- Sélectionnez l'onglet **Boutons** ;
- Cliquez sur le bouton "Insérer un bouton" ;
- La valeur de la propriété **Index** est fixée automatiquement ;
- Fixez les valeurs des propriétés.

Il est aussi possible de créer des objets **ButtonMenu** dans la section **ButtonsMenu**, dans le bas de l'onglet Boutons de la fenêtre Pages de propriétés. Un objet ButtonMenu permet de créer un menu déroulant.

Les principales propriétés générales (onglet Général)

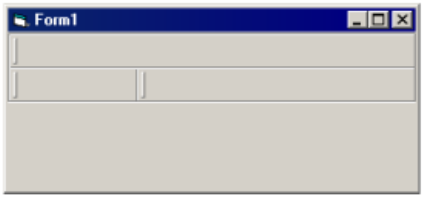
ImageList	Spécifie le nom du contrôle ImageList associé. Affectez le même contrôle à la propriété HotImageList.
HotImageList	Spécifie le nom du contrôle ImageList qui est associé pour les images autoactivées. Bizarrement, cette propriété doit être définie conjointement avec la propriété ImageList pour pouvoir associer une image à un bouton.
ShowTips	Spécifie si oui ou non les infos-bulles des boutons sont affichées (valeur True par défaut).
AllowCustomize	Affiche ou non la boîte de dialogue "Personnaliser la barre d'outils" lorsque l'utilisateur double-clique sur la barre d'outils. Par défaut, la valeur est True et la boîte de dialogue est accessible.

Les principales propriétés des objets Button (onglet Boutons)

Caption	Définit le texte affiché sur un bouton.
Index	Spécifie la position du bouton dans la barre d'outils. Les valeurs d'index débutent à 1.
Key	Permet de nommer un bouton. Cette propriété est très utile pour la lisibilité du code de programmation. Choisissez des noms courts et significatifs.
Image	Spécifie le numéro de l'image affichée sur le bouton. Il doit correspondre à la valeur de la propriété Index de l'image dans le contrôle ImageList.
Style	Définit le style du bouton. **0-tbrDefault** désigne un bouton ayant la forme d'un bouton de commande standard (par défaut). **1-tbrCheck** définit un bouton à deux états ; il peut être enfoncé ou relâché. **2-tbrButtonGroup** permet de créer un groupe de boutons dont un seul peut être enfoncé à la fois. Les boutons d'un groupe doivent être consécutifs. Un bouton de style 3-tbrSeparator doit précéder et suivre le groupe de boutons. **3-tbrSeparator** définit un espacement sur la barre d'outils d'une largeur fixe de 8 pixels. **4-tbrPlaceholder** permet l'insertion d'un contrôle sur la barre d'outils comme par exemple, un ComboBox. **5-tbrDropDown** est le style spécifique aux objets **ButtonMenu**.
ToolTipText	Définit une info-bulle pour un bouton.
Value	Définit l'état du bouton. **0-tbrPressed** spécifie que le bouton est relâché. **1-tbrUnpressed** spécifie que le bouton est enfoncé.
Enabled	Le bouton est accessible ou non.
Visible	Le bouton est visible ou non.

Pour programmer une barre d'outils, utilisez l'événement **ButtonClick** qui est déclenché lorsqu'un bouton est sélectionné.

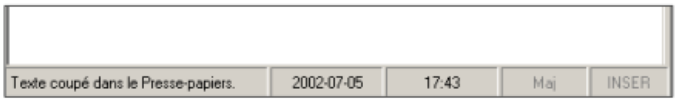
Pour créer des barres d'outils semblables à la figure suivante, utilisez plutôt le contrôle **CoolBar** accessible par Projet /Composants/ Windows Common Controls-3 6.0. Consultez l'aide MSDN à ce sujet.



Le contrôle *StatusBar*

Le contrôle **StatusBar** ou **Barre d'état** permet d'insérer une barre d'état à une application. Une barre d'état apparaît généralement au bas d'une fenêtre et peut contenir diverses informations comme la date et l'heure du système, l'état de certaines touches du clavier ainsi que des informations propres à l'application. Par exemple, dans le cas d'un traitement de texte, on y retrouvera le numéro de la page en cours, le nombre de pages dans le document, la colonne où se trouve le point d'insertion, etc.

Exemple d'un contrôle StatusBar au bas d'une fenêtre

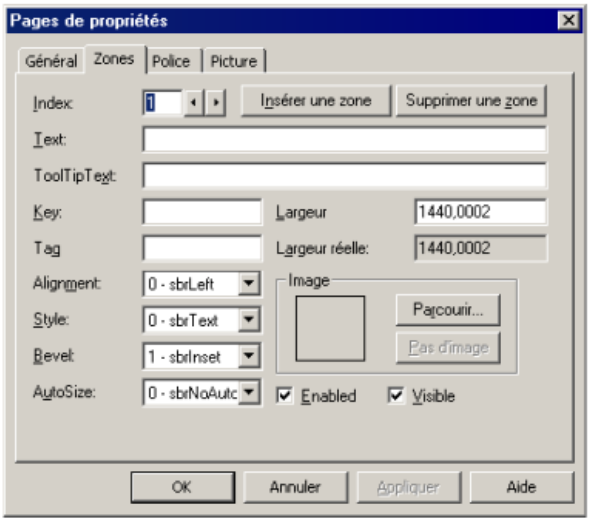


Le contrôle **StatusBar** peut comporter jusqu'à seize objets **Panel** formant une collection **Panels**. Un objet Panel représente une section ou une *zone* de la barre d'état.

Le préfixe **sta** identifie le contrôle **StatusBar** et celui-ci est accessible par le menu **Projet /Composants/ Windows Common Control 6.0**.

Lorsqu'un contrôle StatusBar est ajouté à une feuille, il est automatiquement placé dans le bas de la feuille et ajusté à la largeur de la feuille.

En mode création, pour créer les différentes zones de la barre d'état, utilisez la fenêtre Pages de propriétés, accessible par la propriété Personnalisé. La première zone est déjà créée, utilisez le bouton "Insérer une zone" pour ajouter une seconde zone.



Les principales propriétés des objets Panel (onglet Zones)

Index	Spécifie la position de la zone dans la barre d'état. Les valeurs d'index débutent à 1 et la valeur maximale est 16.
Text	Définit le texte affiché dans une zone ayant le style 0-sbrText.
ToolTipText	Définit une info-bulle pour la zone.
Key	Permet de nommer une zone. Cette propriété est utile dans le cas où la zone est référencée dans le code de programmation. Elle permet d'améliorer la lisibilité du code. Choisissez des noms courts et significatifs.
Picture (section Image de l'onglet Zones)	Permet d'afficher une image dans la zone. Spécifiez le chemin d'accès et le nom du fichier image.
Alignment	Le contenu de la zone est aligné à gauche (par défaut), au centre ou à droite.
Style	Définit le style de la zone. **0-sbrText** définit un style de zone qui peut contenir du texte et/ou une image. **1-sbrCaps** affiche les lettres **Maj** en gras si la touche FixMaj est activée. Elles sont estompées lorsque la touche n'est pas activée. **2-sbrNum** affiche les lettres **Num** estompées ou en gras selon l'état de la touche VerrNum. **3-sbrIns** affiche les lettres **INSER** estompées ou en gras selon l'état de la touche Inser. **4-sbrScrl** affiche les lettres **DÉFIL** estompées ou en gras selon l'état de la touche Défil. **5-sbrTime** affiche l'heure courante. **6-sbrDate** affiche la date courante.

Bevel	Définit l'apparence de la zone. **0-sbrNoBevel** affiche le texte sur la barre d'état. **1-sbrInset** incruste la zone dans la barre d'état. **2-sbrRaised** affiche la zone en relief sur la barre d'état.
Width	Définit la largeur d'une zone fixe.
AutoSize	Spécifie si les zones sont de largeurs fixes ou variables. **0-sbrNoAutoSize** spécifie que les zones occupent la largeur fixée par la propriété Width (par défaut). **1-sbrSpring** spécifie que la largeur des zones est ajustable lorsque la feuille est redimensionnée. **2-sbrContents** spécifie que la zone est ajustable en fonction de son contenu.
Enabled	La zone est accessible ou non à l'utilisateur.
Visible	La zone est visible ou non pour l'utilisateur.

Exercice 22 Compléter le projet *Barre d'outils et barre d'état*

Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre6 /Exercice22 /Debut dans un nouveau dossier .../Visual Basic /Exercice 22 Barres Outils Etat.

2. Dans Visual Basic, ouvrez le projet **Barres.vbp** stocké dans votre dossier .../Visual Basic /Exercice 22 Barres Outils Etat.

> Ce projet est un petit éditeur de texte comportant un menu offrant seulement les choix Couper/Copier/Coller. Le but de cet exercice est d'ajouter au projet une barre d'outils, une barre d'état et un menu contextuel et de démontrer comment synchroniser un même choix disponible dans la barre de menus, dans la barre d'outils et dans le menu contextuel.
>
> La zone de saisie du texte est un contrôle RichTextBox qui porte le nom **rtfTexte**.

3. Ajoutez le composant "Windows Common Control 6.0" à la boîte à outils. Observez les nouveaux contrôles dans la boîte à outils.

Étape 2 Créer une barre d'outils avec les contrôles ImageList *et* ToolBar

1. Ajoutez un contrôle **ImageList** à un endroit quelconque de la feuille. Nommez-le **ilsImages**.

2. Accédez à la fenêtre "Pages de propriétés" du contrôle ImageList et insérez les 5 images suivantes : Cut.bmp, Copy.bmp, Paste.bmp, Bld.bmp et Itl.bmp qui sont stockées dans le dossier Program Files \Microsoft Visual Studio \Common \Graphics \Bitmaps \tlbn_W95.

3. Ajoutez un contrôle **ToolBar** à la feuille. Nommez-le **tlbOutils**.
4. Accédez à la fenêtre "Pages de propriétés" du contrôle ToolBar :
   - Sélectionnez l'onglet Général et par la suite, sélectionnez **ilsImages** dans la zone de liste **ImageList** et dans la zone de liste **HotImageList**.
   - Sélectionnez l'onglet Boutons et insérez 6 boutons :

Index	Key	Image	ToolTipText	Style
1	Couper	1	Couper	0-tbrDefault
2	Copier	2	Copier	0-tbrDefault
3	Coller	3	Coller	0-tbrDefault
4				3-tbrSeparator
5	Gras	4	Style Gras	1-tbrCheck
6	Italique	5	Style Italique	1-tbrCheck

5. Ajoutez les procédures des choix Couper, Copier et Coller du menu :

```
Private Sub mnuEditionCouper_Click()
  Clipboard.Clear
  Clipboard.SetText rtfTexte.SelText
  rtfTexte.SelText = ""
End Sub

Private Sub mnuEditionCopier_Click()
  Clipboard.Clear
  Clipboard.SetText rtfTexte.SelText
End Sub

Private Sub mnuEditionColler_Click()
  rtfTexte.SelText = Clipboard.GetText
End Sub
```

6. Testez les choix Couper, Copier et Coller du menu.

7. Testez les choix Couper, Copier, et Coller en utilisant les raccourcis clavier Ctrl+X, Ctrl+C et Ctrl+V. Comme vous pouvez le constater, les raccourcis clavier ne fonctionnent pas bien parce qu'ils sont déjà préprogrammés dans un menu contextuel propre au contrôle RichTextBox. Ce menu est accessible lorsque la propriété AutoVerbMenu = True.

8. Pour corriger ce bogue, enlevez ou modifiez (à votre choix) les raccourcis clavier des 3 choix à partir du Créateur de menus. Testez de nouveau les raccourcis clavier Ctrl+X, Ctrl+C et Ctrl+V.

9. Codez la procédure des 5 boutons de la barre d'outils :

```
Private Sub tlbOutils_ButtonClick(ByVal _
                        Button As MSComctlLib.Button)
  Select Case Button.Key
    Case "Couper"
      Call mnuEditionCouper_Click
    Case "Copier"
      Call mnuEditionCopier_Click
    Case "Coller"
      Call mnuEditionColler_Click
    Case "Gras"
      rtfTexte.SelBold = Not rtfTexte.SelBold
    Case "Italique"
      rtfTexte.SelItalic = Not rtfTexte.SelItalic
  End Select
End Sub
```

> L'événement **ButtonClick** est déclencher lorsque l'utilisateur clique sur un bouton de la barre d'outils.
>
> La valeur de la propriété **Key** de l'objet **Button** est vérifiée. Dans le cas des choix Couper, Copier et Coller, les procédures des choix de menu correspondantes sont appelées.
>
> Pour les choix Gras et Italique, une bascule est utilisée parce qu'un bouton du style tbrCheck peut avoir seulement deux états : enfoncé et relâché.

10. Testez les 5 boutons de la barre d'outils. Observez l'enfoncement et le relâchement des boutons Gras et Italique.

Étape 3 Ajouter des listes déroulantes à une barre d'outils

1. Ajoutez 2 contrôles ComboBox *non modifiables* sur la barre d'outils. Nommez-le premier cboPolices et l'autre, cboTailles. Ces 2 zones de listes contiendront une liste de polices de caractères et une liste des différentes tailles de polices.

2. Fixez la propriété Sorted du contrôle cboPolices à True afin que les polices soient affichées en ordre alphabétique.

3. Ajoutez 4 nouveaux boutons à la barre d'outils :

Index	Key	Width	ToolTipText	Style
7				3-tbrSeparator
8	Polices	2000	Polices de caractères	4-tbrPlaceholder
9				3-tbrSeparator
10	Tailles	800	Tailles des polices	4-tbrPlaceholder

4. Ajoutez la procédure suivante :

```
Private Sub Form_Activate()
  Dim I As Integer
  With tlbOutils.Buttons("Polices")
    cboPolices.Move .Left, .Top, .Width
  End With
  With tlbOutils.Buttons("Tailles")
    cboTailles.Move .Left, .Top, .Width
  End With
```

Au démarrage, les 2 contrôles ComboBox sont positionnés et alignés sur le bouton correspondant de la barre d'outils par la méthode **Move**.

```
  'Chargement des polices écran du système
  cboPolices.AddItem " "
  For I = 0 To Screen.FontCount - 1
    cboPolices.AddItem Screen.Fonts(I)
  Next
  cboTailles.AddItem " "
  For I = 6 To 24 Step 2
    cboTailles.AddItem I
  Next
```

Les polices et les tailles sont chargées dans les contrôles ComboBox par la méthode AddItem. L'objet **Screen** représente l'écran et sa propriété Fonts retourne toutes les polices écran du système.

Dans les 2 zones de liste, un élément " " est inséré (un caractère d'espacement encadré de guillemets). Cette police ou cette taille bidon est affichée dans la zone de liste lorsque l'utilisateur sélectionne un texte comprenant plusieurs polices ou/et plusieurs tailles.

```
  rtfTexte.Font.Name = "Times New Roman"
  rtfTexte.Font.Size = 12
  cboPolices.Text = rtfTexte.Font.Name
  cboTailles.Text = rtfTexte.Font.Size
End Sub
```

La police et la taille par défaut sont respectivement "Times New Roman" et 12. Celles-ci sont affichées dans les zones de liste.

5. Ajoutez les deux procédures suivantes :

```
Private Sub cboPolices_Click()
  rtfTexte.SelFontName = cboPolices.Text
  rtfTexte.SetFocus
End Sub

Private Sub cboTailles_Click()
  If cboTailles.Text <> " " Then
    rtfTexte.SelFontSize = cboTailles.Text
  End If
  rtfTexte.SetFocus
End Sub
```

Lorsqu'une nouvelle police ou une nouvelle taille est choisie, la valeur de celle-ci est affectée au texte sélectionné ou à la position du point d'insertion. Dans le cas de la taille, la valeur est affectée seulement si celle-ci est numérique.

6. Ajoutez la procédure suivante :

```
Private Sub rtfTexte_SelChange()
With rtfTexte
  If IsNull(.SelFontName) = True Then
    cboPolices.Text = cboPolices.List(0)
  Else
    cboPolices.Text = .SelFontName
  End If
```

```
  If IsNull(.SelFontSize) = True Then
    cboTailles.Text = cboTailles.List(0)
    Else
      cboTailles.Text = .SelFontSize
    End If
  End With
End Sub
```

> L'événement **SelChange** du contrôle RichTextBox se produit lorsque la sélection du texte a été modifiée ou lorsque le point d'insertion est déplacé.
>
> Lorsqu'une sélection de texte contient plusieurs polices ou plusieurs tailles, la valeur Null est retournée. Dans le cas où il s'agit de polices ou de tailles multiples, la première police ou la première taille est affichée dans le contrôle ComboBox. Il s'agit d'une police ou d'une taille à blanc.

7. Testez le choix d'une police de caractères et d'une taille. Effectuez des sélections de texte avec une seule police et une seule taille ainsi que des sélections multiples.

***Étape 4 Créer une barre d'état avec le contrôle* StatusBar**

1. Ajoutez un contrôle **StatusBar** sur la feuille. Nommez-le **staBarreEtat**.
2. Accédez à la fenêtre "Pages de propriétés" du contrôle StatusBar et insérez les 5 zones suivantes :

Index	Style	Alignment	Largeur
1	0-sbrText		3000
2	6-sbrDate	1-sbrCenter	1300
3	5-sbrTime	1-sbrCenter	1300
4	1-sbrCaps	1-sbrCenter	1000
5	3-sbrIns	1-sbrCenter	1000

3. Ajoutez la ligne suivante à la *fin* de la procédure mnuEditionCouper_Click :

```
staBarreEtat.Panels(1).Text = "Texte coupé dans le Presse-papiers."
```

> Une chaîne de caractères est affectée à la propriété Text de la *première* zone de la barre d'état.

4. Affichez dans la première zone de la barre d'état les messages suivants :

   Lorsque le choix Copier est sélectionné : "Texte copié dans le Presse-papiers".

Lorsque le choix Coller est sélectionné : "Le texte du Presse-papiers a été collé".

5. Testez les choix Couper, Copier et Coller avec les boutons correspondants de la barre d'outils et observez l'affichage dans la première zone de la barre d'état.

> Notez que lorsque les raccourcis clavier Ctrl+X, Ctrl+C et Ctrl+V sont utilisés, les messages ne sont pas affichés dans la barre d'état. Ces messages sont affichés, à titre d'exemple, seulement pour démontrer l'affectation d'une valeur à la propriété Text d'un objet Panel.

Étape 5 Synchroniser la barre de menus, le menu contextuel et la barre d'outils

1. Créez un menu contextuel à partir du Créateur de menus. Ajoutez les choix qui suivent *après* le choix Coller :

Caption	Name	Visible
Contextuel	mnuCon	Décochez la case.
...Couper	mnuConCouper	
...Copier	mnuConCopier	
...Coller	mnuConColler	
...-	Barre	
...Gras	mnuConGras	
...Italique	mnuConItalique	

2. Codez les instructions nécessaires pour afficher le menu contextuel de la zone de texte rtfTexte. Le menu contextuel doit être affiché à la *droite* du pointeur de la souris (réf. Exercice 17).

3. Codez les procédures des choix Couper, Copier et Coller du menu contextuel. Faites appel aux procédures déjà codées. Testez ces 3 choix.

4. Ajoutez les procédures suivantes :

```
Private Sub mnuConGras_Click()
  rtfTexte.SelBold = Not rtfTexte.SelBold
End Sub

Private Sub mnuConItalique_Click()
  rtfTexte.SelItalic = Not rtfTexte.SelItalic
End Sub
```

5. Testez les choix Gras et Italique du menu contextuel.

Vous avez probablement remarqué que les boutons Gras et Italique de la barre d'outils ne sont pas synchronisés avec les choix Gras et Italique du menu contextuel. De plus, les choix Gras et Italique du menu contextuel devraient être cochés/décochés selon qu'ils sont activés ou non. Les prochaines opérations visent à corriger ces lacunes.

6. Ajoutez 2 nouvelles procédures personnalisées :

```
Private Sub SynchroGras()
  If rtfTexte.SelBold = True Then
    tlbOutils.Buttons("Gras").Value = tbrPressed
    mnuConGras.Checked = True
  Else
    tlbOutils.Buttons("Gras").Value = tbrUnpressed
    mnuConGras.Checked = False
  End If
End Sub

Public Sub SynchroItalique()
  If rtfTexte.SelItalic = True Then
    tlbOutils.Buttons("Italique").Value = tbrPressed
    mnuConItalique.Checked = True
  Else
    tlbOutils.Buttons("Italique").Value = tbrUnpressed
    mnuConItalique.Checked = False
  End If
End Sub
```

Les propriétés **SelBold** et **SelItalic** sont vérifiées, les boutons de la barre d'outils sont enfoncés ou relâchés et les choix du menu contextuel sont cochés ou décochés selon le cas.

7. Ajoutez les lignes en gras dans les procédures :

```
Private Sub tlbOutils_ButtonClick(ByVal Button _
                                  As MSComctlLib.Button)
  Select Case Button.Key
    Case "Couper"
      Call mnuEditionCouper_Click
    .
    .
    Case "Gras"
      rtfTexte.SelBold = Not rtfTexte.SelBold
      Call SynchroGras
    Case "Italique"
```

```
      rtfTexte.SelItalic = Not rtfTexte.SelItalic
      Call SynchroItalique
  End Select
End Sub

Private Sub mnuConGras_Click()
  rtfTexte.SelBold = Not rtfTexte.SelBold
  Call SynchroGras
End Sub

Private Sub mnuConItalique_Click()
  rtfTexte.SelItalic = Not rtfTexte.SelItalic
  Call SynchroItalique
End Sub

Private Sub rtfTexte_SelChange()
  With rtfTexte
     .
     .
  End With
  Call SynchroGras
  Call SynchroItalique
End Sub
```

Les choix Gras et Italique sont synchronisés quand un nouveau choix est sélectionné soit par un bouton de la barre d'outils ou soit par le menu contextuel. Lorsque la sélection change ou que le point d'insertion est déplacé, il faut aussi ajuster les boutons et les choix du menu contextuel.

8. Testez à fond les choix Gras et Italique de la barre d'outils et du menu contextuel et assurez-vous de leur synchronisme.

9. Exécutez le projet et observez les choix Couper, Copier et Coller. Ils sont toujours accessibles.

Les prochaines opérations visent à rendre accessibles les choix Couper et Copier seulement si une sélection est en cours. Quant au choix Coller, celui-ci est accessible en tout temps. Tous les choix de la barre de menus, de la barre d'outils et du menu contextuel doivent être synchronisés ensemble.

10. Ajoutez les 2 nouvelles procédures personnalisées suivantes :

```
Private Sub CouperCopierAccesNon()
  tlbOutils.Buttons("Couper").Enabled = False
  tlbOutils.Buttons("Copier").Enabled = False
  mnuEditionCouper.Enabled = False
  mnuEditionCopier.Enabled = False
  mnuConCouper.Enabled = False
  mnuConCopier.Enabled = False
End Sub

Private Sub CouperCopierAccesOui()
  tlbOutils.Buttons("Couper").Enabled = True
  tlbOutils.Buttons("Copier").Enabled = True
  mnuEditionCouper.Enabled = True
  mnuEditionCopier.Enabled = True
  mnuConCouper.Enabled = True
  mnuConCopier.Enabled = True
End Sub
```

11. Au démarrage, les choix Couper et Copier ne doivent pas être accessibles ainsi que lorsque l'utilisateur sélectionne le choix Coller. Ajoutez les instructions nécessaires.

12. Ajoutez les lignes en gras dans la procédure suivante :

```
Private Sub rtfTexte_SelChange()
  .
    If .SelText <> "" Then
      Call CouperCopierAccesOui
    Else
      Call CouperCopierAccesNon
    End If
  End With
  Call SynchroGras
  Call SynchroItalique
End Sub
```

> Si un texte est sélectionné (`.SelText <> ""`), les choix Couper et Copier sont accessibles sinon, ils ne sont pas accessibles.

13. Testez les choix Couper, Copier et Coller à partir de la barre de menus, de la barre d'outils et du menu contextuel. Assurez-vous que les choix sont accessibles aux bons moments et qu'ils sont synchronisés.

14. Replacez les procédures dans un ordre logique.

7 Les fichiers de données et les rapports

7.1 LES FICHIERS DE DONNÉES

Les **fichiers de données** son utilisés pour conserver des informations de façon structurée afin de pouvoir y accéder facilement par la suite.

Structure d'un fichier de données

Un fichier de données est formé **d'enregistrements** subdivisés en **champs**. Un **champ** contient une information ou une donnée concernant par exemple une personne, comme son nom, son âge ou son code permanent.

Exemple

L'ensemble des informations sur les étudiants d'une maison d'enseignement, constitue le fichier des étudiants. Celui-ci est subdivisé en enregistrements dont chacun regroupe les données d'un seul étudiant qui sont stockées dans des champs.

Fichier de données des étudiants

	Champ 1	Champ 2	Champ 3	Champ 4
	Code permanent	**Nom**	**Naissance**	**Sexe**
Enregistrement 1	LARJ80120944	Jean Larue	1980/12/09	M
Enregistrement 2	POIM84553019	Marie Poirier	1984/05/30	F
Enregistrement 3	JETR82612755	Rita Jetté	1982/11/27	F

Les fichiers de données et leurs types d'accès

Il existe trois types de fichiers de données en Visual Basic. Le type d'un fichier dépend du mode d'accès aux données. On retrouve :

- Les fichiers séquentiels ;
- Les fichiers directs ;
- Les fichiers binaires.

Accès à un fichier séquentiel

On accède aux données d'un **fichier séquentiel**, enregistrement par enregistrement. Ce type d'accès se compare à l'enregistrement d'une cassette audio. Les chansons sont enregistrées les unes à la suite des autres tout comme les enregistrements d'un fichier séquentiel. Pour accéder à une chanson précise, il faut dérouler les chansons précédentes. Dans le cas d'un fichier séquentiel, les enregistrements précédents doivent être lus avant d'accéder à un enregistrement précis.

Ce type de fichiers est utilisé pour enregistrer des données qui sont enregistrées et lues en bloc. Ils sont aussi utilisés pour créer des fichiers texte. Les fichiers séquentiels sont faciles à manipuler, mais se révèlent plus lents d'accès que les fichiers binaires. Aussi, leurs enregistrements ne peuvent pas être mis à jour comme ceux des fichiers directs.

Les enregistrements des fichiers séquentiels peuvent être lus par la fonction **Input()**, les instructions **Input** ou **Line Input** et ils sont créés par les instructions **Print** ou **Write**.

Les exercices 23, 24, 25 et 26 démontrent l'utilisation des fichiers séquentiels.

Accès à un fichier direct

Les enregistrements d'un **fichier direct** sont accédés soit séquentiellement ou soit directement en spécifiant la position d'un enregistrement spécifique. L'accès direct aux enregistrements, se compare à l'accès aux chansons enregistrées sur un CD. On accède directement à une chanson par la position de celle-ci. Les enregistrements doivent être structurés et sont tous de la même longueur. La structure des champs des enregistrements est définie par l'instruction **Type** qui permet de créer un type de données personnalisé (réf. 5.19).

Exemple de définition d'une structure

```
Private Type udtEtudiant
  CodePermanent As String * 12
  Nom As String * 30
  Naissance As Date
  Sexe As Boolean
End Type
```

Les enregistrements sont tous construits de la même façon et chacun d'eux occupe 52 octets.

Bien que ce type de fichiers offre plus de possibilités que les fichiers séquentiels, comme l'accès direct et la possibilité de mise à jour, ils sont difficiles à manipuler et souvent, ils occupent inutilement de l'espace. Ils sont avantageusement remplacer par les bases de données.

Les enregistrements d'un fichier direct sont créés par l'instruction **Put** et sont lus par l'instruction **Get**.

Un fichier à accès direct est utilisé dans l'exercice 27.

Accès à un fichier binaire

Les **fichiers binaires** permettent d'améliorer la vitesse d'exécution des opérations séquentielles, de lecture et d'écriture sur des fichiers.

Comme les fichiers directs, une structure d'enregistrement peut être définie par l'instruction Type. Il n'est cependant pas obligatoire que les enregistrements soient tous de la même longueur, permettant ainsi un gain d'espace. Par exemple, le nom "Guy Roy" occupera moins d'espace que le nom "Guylaine Roy Tremblay" sur le fichier.

Les enregistrements d'un fichier binaire sont créés par l'instruction **Put** et sont lus par la fonction **Input()** ou l'instruction **Get**.

Les instructions et fonctions d'accès aux fichiers

L'instruction Open et la fonction FreeFile

L'instruction **Open** procède à l'ouverture d'un fichier. Avant de pouvoir exécuter une opération de lecture ou d'écriture sur un fichier, celui-ci doit être ouvert. L'instruction Open associe une zone tampon au fichier et spécifie le type d'accès du fichier.

Syntaxe : **Open** *filename* **For** *mode* **As** [**#**]*filenumber* [**Len**=*reclength*]

L'argument *filename* est une chaîne de caractères contenant le chemin d'accès et le nom du fichier. Par défaut, le chemin d'accès est celui du projet.

Exemple : "C:\Temp\VB6\Donnees.Dat" ou "Statistiques.txt".

L'argument *mode* est un mot clé qui spécifie le mode d'ouverture du fichier :

Le mot clé **Input** spécifie que le *fichier séquentiel* est ouvert en mode *lecture* au début de celui-ci. Le fichier doit exister, sinon l'erreur d'exécution 53 "Fichier introuvable" est générée.

Le mot clé **Output** spécifie que le *fichier séquentiel* est ouvert en mode *écriture* au début de celui-ci. Si le fichier existe, il est remplacé sans avertissement.

Le mot clé **Append** spécifie que le *fichier séquentiel* est ouvert en mode *écriture* à la fin de celui-ci, les nouveaux enregistrements étant ajoutés à la suite des enregistrements existants. Si le fichier n'existe pas, un nouveau fichier est créé.

Le mot clé **Random** (mode par défaut) ouvre un *fichier existant à accès direct* au début de celui-ci, permettant *la lecture et l'écriture* des enregistrements. Si le fichier n'existe pas, un nouveau fichier à accès direct est créé.

Le mot clé **Binary** ouvre un *fichier binaire existant* au début de celui-ci, permettant *la lecture et l'écriture* des enregistrements. Si le fichier n'existe pas, un nouveau fichier binaire est créé.

L'argument #*filenumber* est un numéro compris entre 1 et 511 qui identifie un fichier dans un programme. Ce numéro d'identification est utilisé par la suite dans les instructions de lecture, d'écriture et de fermeture pour référencer le fichier. Il s'agit généralement d'une constante comme par exemple #1 pour le premier fichier ouvert et #2 pour le deuxième. Il est aussi possible d'utiliser la fonction **FreeFile** qui retourne le premier numéro de fichier libre.

Exemple

```
Open "Statistiques.txt" For Input As #1

Dim intNo As Integer
intNo = FreeFile
Open "Statistiques.txt" For Input As intNo
```

L'argument **Len** = *reclength* est obligatoire pour un fichier à accès direct. Il spécifie la longueur d'un enregistrement (Maximum : 32 767 octets). S'il est utilisé pour un fichier séquentiel, il spécifie la longueur de la zone tampon. Cet argument est ignoré dans le cas d'un fichier binaire.

Si un fichier est déjà ouvert ou si le type d'accès spécifié n'est pas autorisé, l'instruction Open génère une erreur d'exécution. Par exemple, si un fichier est ouvert en mode Input et qu'une instruction d'écriture a été codée, une erreur est générée.

Exemples d'instructions Open

`Open "Statistiques.txt" For Input As #1`	Ouverture en mode lecture d'un fichier séquentiel.
`Open "A:Donnees.dat" For Output As #2`	Ouverture en mode écriture d'un fichier séquentiel.
`Open "Fichier.txt" For Append As #3`	Ouverture d'un fichier séquentiel en mode ajout.
`Open "Etudiants.dat" For Random As #4 _ Len = 52`	Ouverture d'un fichier à accès direct.
`Open "Etu.dat" For Binary As intNo`	Ouverture d'un fichier binaire.
`Open "A:Donnees.dat" For Output As 2`	Le caractère # est optionnel.

L'instruction Close

L'instruction **Close** referme un fichier ouvert par l'instruction Open, terminant ainsi toute opération de lecture ou d'écriture sur le fichier.

Syntaxe : **Close** [#][*filenumberlist*]

L'argument *filenumberlist* représente un ou plusieurs numéros de fichiers séparés par des virgules. Par défaut, *tous les fichiers ouverts son fermés*.

Exemples d'instructions Close

`Close #1`	Fermeture du fichier portant le numéro 1.
`Close 1, 3`	Fermeture des fichiers portant le numéro 1 et le numéro 3.
`Close`	Fermeture de tous les fichiers ouverts.

L'instruction Write (fichiers séquentiels)

L'instruction **Write** crée un enregistrement sur un fichier séquentiel qui doit être ouvert en mode Output.

Syntaxe : **Write #***filenumber*, [*outputlist*]

L'argument *filenumber* est un numéro de fichier valide.

L'argument *outputlist* peut contenir des noms de champs, des expressions numériques ou des expressions de chaîne, séparés par des virgules.

L'instruction Write insère une virgule entre chacun des éléments et les chaînes de caractères sont encadrées de guillemets. Les données enregistrées sur un fichier par l'instruction **Write** sont par la suite lues avec l'instruction **Input** qui reconnaît la

virgule comme étant un séparateur de champs et les guillemets comme délimiteur de chaînes.

Exemples d'instructions Write

`Write #1, txtNom.Text, txtAge.Text`	Écrit le contenu des zones de texte txtNom et txtAge dans un fichier séquentiel.
`Write #2, strNom, intAge, sngSalaire`	Écrit le contenu des variables strNom, intAge et sngSalaire dans un fichier séquentiel.
`Write #3,`	Écrit un enregistrement à blanc dans un fichier séquentiel.

Exemples d'enregistrements créés par l'instruction suivante

```
Write #1, strNom, intAge, sngSalaire, blnMembre, dtmNais

"Marie-France Goulet",24,567.88,#TRUE#,#1978-02-27#
"Louis-André Lecours",37,786.39,#FALSE#,#1965-11-30#
"Guy Larivière",49,972.11,#FALSE#,#1953-04-27#
```

Une paire de guillemets délimite une chaîne, une virgule sépare chaque élément, le point décimal représente le symbole décimal, les valeurs #TRUE# et #FALSE# représentent les valeurs booléennes et le caractère # délimite une date.

L'instruction Input et la fonction EOF

L'instruction **Input** lit les données dans un fichier séquentiel, ouvert en mode Input et affecte chaque donnée lue à une variable.

Syntaxe : **Input #***filenumber, varlist*

L'argument *filenumber* est un numéro de fichier valide.

L'argument *varlist* représente une liste de variables séparées par des virgules. Les variables doivent être citées dans le même ordre que les données enregistrées par l'instruction Write et les types de données doivent être compatibles.

Un fichier dont la fin a été atteinte ne doit plus être lu sans causer l'erreur d'exécution 62 "L'entrée dépasse la fin du fichier."

La fonction **EOF** (End Of File) permet de déterminer la fin d'un fichier ouvert en mode Input ou Random. Elle retourne la valeur True lorsque la fin du fichier a été rencontrée.

Syntaxe : **EOF(***filenumber***)**

L'argument *filenumber* est un numéro de fichier valide.

Exemples de lectures d'un fichier séquentiel

Exemple 1

```
Do While Not EOF(1)
  Input #1, strNom, intAge, sngSalaire, blnMembre, dtmNais
  .
  .
Loop
```

Le fichier est lu tant que la fin du fichier n'ait pas été détectée.

Exemple 2

```
Do Until EOF(1)
  Input #1, strNom, intAge, sngSalaire, blnMembre, dtmNais
  .
  .
Loop
```

Le fichier est lu jusqu'à ce que la fin du fichier soit rencontrée.

L'instruction Print et les fonctions Spc et Tab

L'instruction **Print** permet d'enregistrer des données formatées pour l'affichage ou l'impression, sur un fichier séquentiel qui doit être ouvert en mode Output. Un enregistrement correspond à une ligne lorsque le fichier est affiché ou imprimé.

Syntaxe : **Print #***filenumber*, [*outputlist*]

L'argument *filenumber* est un numéro de fichier valide.

L'argument *outputlist* est une liste d'éléments à imprimer ou à afficher séparés par des virgules. Cette liste peut contenir la fonction **Spc(n)** (n : nombre d'espaces) et la fonction **Tab(n)** (n : numéro de colonne).

Utilisée avec la virgule comme séparateur d'éléments, l'instruction Print insère une tabulation entre les différents éléments, à tous les 7 caractères. La fonction **Tab** sans argument peut être employée au lieu d'une virgule. Le point-virgule peut aussi être utilisé à la place d'une virgule mais dans ce cas, l'élément suivant sera placé à la position suivante.

Les chaînes de caractères ne sont pas délimitées. Un nombre est précédé d'un caractère d'espacement s'il est positif et du signe – s'il est négatif. Un nombre est suivi d'un caractère d'espacement et le point est utilisé comme symbole décimal.

Une instruction Print sans argument *outputlist* crée un enregistrement à blanc ou une ligne vide. Une instruction Print qui se termine par une virgule ou par un point-virgule

spécifie d'enregistrer les éléments de la prochaine instruction Print dans le même enregistrement ou sur la même ligne.

Exemples d'instructions Print

```
Print #1, txtNom.Text, txtAge.Text

Claire Laverdure    50
```

```
Print #2, strNom, intAge, sngSalaire, blnMembre, dtmNais

Claire Laverdure    50      567.67   Vrai       1952-10-22
Guy Roy     20      888.97  Faux     1982-05-09
```

```
Print #2, strNom, Tab(25), intAge, Spc(2), sngSalaire, blnMembre; dtmNais

Claire Laverdure       50␣␣567.67␣     Vrai1952-10-22
Guy Allard             20␣␣888.97␣      Faux1982-05-09
```

```
Print #2, strNom; intAge; sngSalaire
Print #2, blnMembre, dtmNais

Guy Allard␣20␣␣888.97␣
Faux   1982-05-09
```

```
Print #1 strNom;
Print #1 intAge

Guy Allard␣20
```

L'instruction **Print** peut être utilisée comme *méthode* des objets **Form**, **PictureBox**, **Debug** et **Printer**. La méthode Print imprime directement sur la feuille (Form), dans la fenêtre d'exécution (Debug), sur l'imprimante (Printer) ou dans un contrôle PictureBox.

Syntaxe : *objet*.**Print** [*outputlist*]

Exemples

```
Me.Print "Texte affiché sur la feuille"
Debug.Print "Contenu de la variable : "; sngVariable
Printer.Print Tab(40); "Rapport mensuel"
picAffichage.Print "Texte affiché dans un contrôle PictureBox"
```

L'instruction Line Input

L'instruction **Line Input** lit une ligne ou un enregistrement sur un fichier créé par l'instruction Print. L'information lue est placée dans une variable de type String.

Syntaxe : **Line Input #***filenumber*, *varname*

L'argument *filenumber* est un numéro de fichier valide.

L'argument *varname* est un nom de variable dans laquelle les données lues sont stockées.

Exemple de lecture d'un fichier séquentiel avec l'instruction Line Input

```
Dim strLigne As String
Open "Fichier.txt" For Input As #1
Do While Not EOF(1)
  Line Input #1, strLigne
  .
Loop
Close #1
```

La fonction Input() et la fonction LOF

La fonction **Input()** retourne une chaîne de caractères contenant les caractères lus dans un fichier séquentiel ouvert en mode Input ou dans un fichier binaire ouvert en mode Binary.

Syntaxe : **Input(***number*, [**#**]*filenumber***)**

L'argument *number* est une valeur numérique qui spécifie le nombre de caractères à retourner.

L'argument *filenumber* est un numéro de fichier valide.

Les données lues à l'aide de la fonction **Input()** ont généralement été enregistrées par une instruction Print dans le cas d'un fichier séquentiel ou par l'instruction Put dans le cas d'un fichier binaire.

Utilisez la fonction **LOF** (Length Of File) pour lire tout le contenu d'un fichier. La fonction LOF retourne une valeur de type Long représentant la taille en octets d'un fichier ouvert par l'instruction Open.

Syntaxe : **LOF(***filenumber***)**

L'argument *filenumber* est un numéro de fichier valide.

Exemples d'utilisation

Exemple 1

```
Open "A:\Exercices\Texte.txt" For Input As #1
txtZoneTexte.Text = Input(LOF(1), #1)
Close #1
```

Le fichier séquentiel de type texte est ouvert en mode Input par l'instruction Open. Son contenu est lu par la fonction Input et affecté à la zone de texte txtZoneTexte. Il est par la suite fermé par l'instruction Close.

Exemple 2

```
Open "A:\Exercices\Texte.txt" For Output As #1
Print #1, txtZoneTexte.text
Close #1
```

Le fichier séquentiel de type texte est ouvert en mode Output par l'instruction Open. Le contenu de la zone de texte txtZoneTexte est enregistré dans le fichier par l'instruction Print. Par la suite, le fichier est fermé par l'instruction Close.

Exemple 3

Dans le cas d'un **fichier binaire**, codez comme suit :

```
Open "A:\Exercices\Texte.txt" For Binary As #1
txtZoneTexte.Text = Input(LOF(1), #1)
Close #1

Dim strVariable As String
strVariable = txtZoneTexte.Text
Open "A:\Exercices\Texte.txt" For Binary As #1
Put #1, , strVariable
Close #1
```

Un fichier binaire doit être enregistré par l'instruction Put qui requiert une variable d'enregistrement.

L'instruction Put

L'instruction **Put** crée un enregistrement sur un fichier à accès direct (mode Random) ou un fichier binaire (mode Binary) à partir d'une variable d'enregistrement.

Syntaxe : **Put** [**#**]*filenumber*, [*recnumber*], *varname*

L'argument *filenumber* est un numéro de fichier valide.

L'argument *recnumber* est une valeur de type Long qui représente la position de l'enregistrement pour un fichier à accès direct et la position de l'octet pour un fichier binaire. Par défaut, l'écriture commence à la suite de la position précédente ou de l'octet suivant. Notez que l'argument omis doit être remplacé par une virgule.

L'argument *varname* est le nom de la variable d'enregistrement qui contient les données à enregistrer sur le fichier. Dans le cas d'un fichier à accès direct, il s'agit généralement d'une donnée de type personnalisé définie par l'instruction Type.

Les données enregistrées par l'instruction Put sont généralement lues par l'instruction Get. Dans le cas d'un fichier binaire la fonction Input() peut aussi être utilisée.

Exemple : Création d'un enregistrement sur un fichier à accès direct

```
Private Type Etudiant
  Nom As String * 30
  CodeReg As String * 3
  Telephone As String * 7
  Sexe As Boolean              '2 octets
  DateNais As Date             '8 octets
  CodePerm As String * 12
End Type

Dim udtEtudiant As Etudiant           'Variable d'enregistrement

Open "fichier.dat" For Random As #1 Len = Len(udtEtudiant)
.
udtEtudiant.Nom = txtNom.Text
udtEtudiant.CodeReg = txtCodeReg.Text
udtEtudiant.Telephone = txtTel.Text
udtEtudiant.Sexe = blnSexe
udtEtudiant.DateNais = dtmDateNais
udtEtudiant.CodePerm = txtCodePerm.Text
.
Put #1, , udtEtudiant
.
Close #1
```

Une variable d'enregistrement udtEtudiant est déclarée avec le type de donnée personnalisé Etudiant.

Le fichier est ouvert en mode Random et la longueur de chaque enregistrement est égale à la longueur de la variable udtEtudiant.

Les données à enregistrer proviennent de zones de texte et de variables. Elles sont transférées dans la variable d'enregistrement et cette dernière est enregistrée dans le fichier à accès direct par l'instruction Put à la position 1. Le second enregistrement sera placé à la position 2 et ainsi de suite.

Logiquement, le fichier est ouvert et fermé une seule fois dans un même programme. Il est ouvert avant d'enregistrer le premier enregistrement et fermé après avoir enregistré le dernier enregistrement. Ouvrir et fermer un fichier pour chacun des enregistrements est peu efficace. Par exemple, un programme crée 50 enregistrements, ce qui requiert 52 accès au fichier : un pour l'ouverture, un pour chacune des écritures et un pour la fermeture. Ouvrir et fermer un fichier pour chaque enregistrement requiert alors 150 accès au fichier.

L'instruction Get

L'instruction **Get** lit des données sur un fichier à accès direct ouvert en mode Random ou un fichier binaire ouvert en mode Binary. Les données qui sont lues sont placées dans une variable d'enregistrement.

Syntaxe : **Get** [**#**]*filenumber*, [*recnumber*], *varname*

Les arguments *filenumber*, *recnumber* et *varname* sont semblables à ceux de l'instruction Put décrite précédemment. Si l'argument *recnumber* est omis, la lecture débute à la position 1 dans le cas d'un fichier à accès direct et à l'octet 1 dans le cas d'un fichier binaire, accédant ainsi aux données de façon séquentielle.

Les données lues avec l'instruction Get ont généralement été enregistrées par l'instruction Put.

Exemples

```
Private Type Etudiant
  Nom As String * 30
  CodeReg As String * 3
  Telephone As String * 7
  Sexe As Boolean
  DateNais As Date
  CodePerm As String * 12
End Type

Dim udtEtudiant As Etudiant           'Variable d'enregistrement
```

Exemple d'une lecture séquentielle d'un fichier à accès direct

```
Open "fichier.dat" For Random As #1 Len = Len(udtEtudiant)
Do Until EOF(1)
  Get #1, , udtEtudiant
  .
  .
Loop
Close 1
```

Exemple d'une lecture directe sur un fichier à accès direct

```
Open "fichier.dat" For Random As #1 Len = Len(udtEtudiant)
Get #1,7,udtEtudiant
```

L'enregistrement stocké à la position 7 est lu et les données sont placées dans la variable d'enregistrement udtEtudiant.

Exercice 23 Modifier le projet *Inscription*

Étape 1 Préparation

1. À partir de l'Explorateur de Windows, copiez tous les fichiers de votre dossier .../Visual Basic /Exercice 3 Inscription dans un nouveau dossier .../Visual Basic /Exercice 23 Fichier sequentiel.
2. Dans Visual Basic, ouvrez le projet **Inscription.vbp** stocké dans le dossier .../Visual Basic /Exercice 23 Fichier sequentiel.
3. Afin que le contrôle cboVille ne soit pas modifiable, modifiez sa propriété Style pour 2-DropDownList.

Étape 2 Saisir et enregistrer des données dans un fichier séquentiel

1. Lors de l'événement Form_Load, ajoutez les 2 instructions suivantes :

```
cboVille.Text = "Montréal"
Open "Etudiant.txt" For Append As #1
```

> La valeur par défaut "Montréal" est affectée à la propriété Text du contrôle cboVille.
>
> Le fichier séquentiel Etudiant.txt est ouvert en mode *ajout* (Append) dans le dossier courant du projet. En mode Append, si le fichier n'existe pas, il est créé. S'il existe, les nouveaux enregistrements sont ajoutés à la fin.

2. Lors de l'événement Form_Unload, enlevez le message de fin et ajoutez la ligne de code suivante :

```
Close #1
```

> Le fichier séquentiel Etudiant.txt est fermé à la fin du programme. Le fichier est donc ouvert une seule fois au début de l'exécution et fermé seulement à la fin de l'exécution.

3. Dans la procédure cmdEnregistrer_Click, remplacez la ligne de code qui affiche le message "Étudiant enregistré" par celle-ci :

```
Call EnregistrementEtudiant
```

4. Ajoutez la procédure personnalisée EnregistrementEtudiant :

```
Private Sub EnregistrementEtudiant()
  Write #1, txtNumero.Text, txtNom.Text, _
              cboVille.ListIndex, lstOrientation.ListIndex, _
              optGarcon.Value, txtCourriel.Text
End Sub
```

> L'instruction Write enregistre les données d'un étudiant dans un fichier séquentiel. Le contenu des zones de texte est enregistré intégralement tandis que seulement les index de la ville et de l'orientation sont conservés. Le sexe est stocké sous forme booléenne, #TRUE# ou #FALSE#.

5. Exécutez le projet et enregistrez 5 étudiants. Variez les données : villes, orientations, sexe, avec courriel, sans courriel, etc.

6. À partir de l'Explorateur de Windows, double-cliquez sur le fichier Etudiant.txt afin de démarrer l'application Bloc-Notes. Observez les éléments suivants :

   - Un enregistrement par ligne ;
   - Les champs sont séparés par une virgule ;
   - Les chaînes de caractères sont encadrées de guillemets ;
   - Les index des villes et orientations ;
   - Les valeurs #TRUE# et #FALSE# ;
   - La valeur chaîne de longueur nulle ("") lorsque l'étudiant n'a pas de courriel.

Étape 3 Gérer les erreurs d'accès au fichier de données

> Pour plus de détails sur la gestion des erreurs, consultez la section 5.17 La gestion des erreurs.

1. Ajoutez la constante suivante :

```
Const cstrNomFichier As String = "Etudiant.txt"
```

2. Ajoutez ou modifiez les lignes en gras dans la procédure suivante :

```
Private Sub Form_Load()
  lstOrientation.Text = "Informatique"
  cboVille.Text = "Montréal"
  On Error GoTo Erreur
  Open cstrNomFichier For Append As #1
  Exit Sub
```

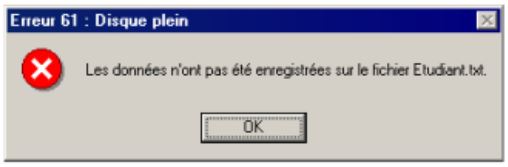
```
Erreur:
  MsgBox "Le fichier des étudiants " & cstrNomFichier & _
         " ne peut être ouvert.", vbExclamation, _
         "Erreur " & Err.Number
End Sub
```

Une constante est utilisée pour conserver le nom du fichier, car celui-ci est utilisé dans l'instruction Open et dans la fonction MsgBox.

Lors de l'ouverture du fichier, un message d'erreur significatif est affiché en cas de problème d'accès. Le numéro de l'erreur est affiché dans la barre de titre du message.

Il est très important de gérer les erreurs d'accès aux fichiers de données et d'afficher des messages clairs à l'utilisateur, *même si le risque d'erreur est minime*. L'erreur est ainsi interceptée *sans plantage* et même si l'utilisateur ne peut intervenir pour la corriger, il peut rapporter le message à un spécialiste, qui retrouvera rapidement l'endroit dans le code où l'erreur s'est produite et la cause de celle-ci.

3. Dans la procédure EnregistrementEtudiant, affichez le message qui suit en cas de problème d'accès lors de l'enregistrement des données. Affichez le numéro de l'erreur dans la barre de titre suivi de la description de celle-ci.



7.2 Production de rapports à l'aide de l'objet Printer

L'objet **Printer** permet de communiquer avec une imprimante qui par défaut est celle spécifiée dans les paramètres des imprimantes de Windows. La collection **Printers** regroupe toutes les imprimantes rattachées à un système, chacune d'elles étant identifiée par un index. Exemple : Printers(0) à Printers(Printers.Count – 1).

L'objet Printer comporte plusieurs propriétés et méthodes. Dans les prochaines sections, les plus importantes d'entre elles y sont détaillées.

Les principales propriétés de l'objet Printer

Printer.ColorMode [= *valeur*]	Mode Couleur. vbPRCMColor (*valeur* = 2) pour une impression couleur ; vbPRCMMonochrome (*valeur* = 1) pour une impression noir & blanc.
Printer.Copies [= *nombre*]	*Nombre* de copies à imprimer.
Printer.Duplex [= *valeur*]	Impression recto-verso. vbPRDPSimplex (*valeur* = 1) pour une impression recto avec l'orientation en cours ; vbPRDPHorizontal (*valeur* = 2) pour une impression recto avec l'orientation portrait ; vbPRDPVertical (*valeur* = 3) pour une impression recto-verso avec l'orientation paysage.
Printer.Font.*propriété* [= *valeur*]	**Font** est un objet qui permet d'accéder aux propriétés suivantes : Bold — style gras Charset — jeu de caractères Italic — style italique Name — nom de la police Size — taille de la police Strikethrough — effet barré Underline — style souligné Weight — épaisseur des caractères
Printer.FontBold [= *boolean*] **Printer.FontItalic** [= *boolean*] **Printer.FontStrikethru** [= *boolean*] **Printer.FontUnderline** [= *boolean*]	Impression en gras (FontBold), en italique (FontItalic), barré (FontStrikethru), ou souligné (FontUnderline) si la valeur est True.
Printer.FontCount	Nombre de polices disponibles pour l'imprimante.
Printer.FontName [= *font*]	Nom de la police de caractères. L'argument *font* est une chaîne de caractères.
Printer.Fonts(*index***)**	Retourne tous les noms de polices disponibles pour l'imprimante. L'argument *index* varie de 0 à FontCount – 1.
Printer.FontSize [= *points*]	Taille de la police de caractères exprimée en *points*. Le nombre maximal est 2160 points.
Printer.hDC	Identificateur de l'imprimante qui provient du système d'exploitation.
Printer.Orientation [= *valeur*]	Orientation de l'impression. vbPRORPortrait (*valeur* = 1) pour une orientation portrait ; vbPRORLandscape (*valeur* = 2) pour une orientation paysage.
Printer.Page	Numéro de la page en cours.

Printer.PaperBin [= *valeur*]	Bac d'alimentation de l'imprimante. La *valeur* peut varier de 1 à 14 et 14 constantes sont disponibles. Consultez l'aide MSDN.
Printer.PaperSize [= *valeur*]	Format du papier (+ de 40 formats prédéfinis). vbPRPSLetter (*valeur* = 1) pour un format 8.5 x 11 ; vbPRPSLegal (*valeur* = 5) pour un format 8.5 x 14. Consultez l'aide MSDN pour les autres formats.
Printer.PrintQuality [= *valeur*]	Résolution de l'imprimante. vbPRPQDraft (*valeur* = -1) pour une résolution brouillon ; vbPRPQLow (*valeur* = -2) pour une basse résolution ; vbPRPQMedium (*valeur* = -3) pour une résolution moyenne ; vbPRPQHigh (*valeur* = -4) pour une haute résolution.

Les principales méthodes de l'objet Printer

Printer.NewPage	Provoque un saut de page et la propriété Page est augmentée de 1.
Printer.Print [*outputlist*]	Envoie une ligne à l'objet Printer. La méthode Print de l'objet Printer s'utilise de la même façon que l'instruction Print (réf. page 359, L'instruction *Print* et les fonctions *Spc* et *Tab*). Au lieu d'enregistrer les informations sur un disque, celles-ci sont dirigées vers l'objet Printer. Lorsque la virgule est utilisée comme séparateur dans *outputlist*, les tabulations sont fixées à toutes les 14 positions. Par exemple, le premier élément débute en colonne 1, le deuxième en colonne 15, le troisième en colonne 29 et ainsi de suite. La structure "With/End With" ne peut être utilisée avec la méthode Print de l'objet Printer ; une erreur de compilation est générée.
Printer.EndDoc	Met fin aux impressions dirigées vers l'objet Printer et envoie le document à l'imprimante ou au "*spooler*" qui gère les files d'impression. Si la méthode n'est pas invoquée, le document sera envoyé à l'imprimante à la fin de l'exécution.

Exemple 1 : **Utilisation des fonctions Spc, Tab et du point-virgule**

Instructions	Résultat
`Printer.Print "ABC"; Spc(3); "DEF"`	`ABC␣␣␣DEF`
`Printer.Print Tab(4); "ABC";` `Printer.Print "DEF"`	`␣␣␣ABCDEF`
`Printer.Print "A"; "M"`	`AM`
`Printer.Print 1; 342.67; -5; 879.5`	`␣1␣␣342,67␣-5␣␣879,5␣`

- La fonction Spc spécifie d'imprimer 3 espaces et la fonction Tab spécifie d'imprimer l'élément qui suit dans la colonne 4.
- Les chaînes de caractères sont imprimées sans espacement.
- Les valeurs numériques positives sont précédées d'une espace.
- Les valeurs numériques sont suivies d'une espace.
- Une ligne de code qui se termine par un séparateur spécifie d'imprimer les prochains éléments sur la même ligne.
- La virgule est utilisée comme symbole décimal.

Exemple 2 **: Utilisation de la virgule et des propriétés de l'objet Printer**

Instructions

```
Printer.FontName = "Courier New"
Printer.FontSize = 9
Printer.Print "12345678901234567890123456789012345678901234567890"
Printer.Print "A", "B", "C"
Printer.Print 1, -5, 8 * 9
Printer.Print "ABC",
Printer.Print "DEF", String(5, "_")
Printer.Print     'ligne vide
Printer.FontBold = True
Printer.FontItalic = True
Printer.FontSize = 12
Printer.Print "Fin de l'impression"
Printer.EndDoc
```

Résultats

```
12345678901234567890123456789012345678901234567890
A             B             C
 1            -5            72
ABC           DEF            _____

Fin de l'impression
```

- La police utilisée est Courier New avec une taille de 9.
- À chaque fois qu'une virgule est rencontrée, l'élément suivant est imprimé à la prochaine tabulation (colonnes 1, 15 ou 29).
- `ABC` et `DEF` sont imprimés sur la même ligne parce que l'impression de `ABC` se termine par une virgule.

- La fonction String permet de tracer une ligne de 5 caractères.
- La méthode Print sans argument imprime une ligne blanche.
- La dernière ligne est imprimée avec une taille de 12, avec les styles gras et italique.

Imprimer des nombres en colonne (instructions *Rset* et *Lset)*

Il est souvent nécessaire de produire des rapports dont les nombres doivent être alignés à droite et sur le symbole décimal.

Exemple

```
Résultats alignés :

Solde                              1245,78
Dépôt              900,12          2145,90
Retrait           1800,00           345,90
Retrait            300,00             5,90

Résultats non alignés :

Solde                              1245,78
Dépôt              900,12          2145,9
Retrait            1800            345,9
Retrait            300             5,9
```

Pour aligner des nombres dans une colonne :

- Utilisez une fonction de formatage pour imprimer un nombre fixe de décimales. Exemple : FormatNumber(*nombre*, 2).
- Utilisez une police de caractères à *espacement fixe* au lieu d'une *police proportionnelle*. Avec une police à espacement fixe, tous les caractères occupent le même espacement. Exemple : Courier New.
- Alignez les nombres à droite avec l'instruction RSet.

Les instructions RSet et LSet

Les instructions **RSet** (**Right Set**) et **LSet (Left Set)** alignent respectivement à droite et à gauche, une chaîne à l'intérieur d'une chaîne *de longueur fixe*.

Syntaxe : **RSet** *chaînefixe* = *chaîne*

LSet *chaînefixe* = *chaîne*

L'argument *chaîne* représente l'expression qui doit être alignée.

L'argument *chaînefixe* est une variable de type String dont la longueur est fixe.

Exemples d'utilisation des instructions RSet et LSet

Instructions	Résultat
`Dim strFixe As String * 10`	␣␣␣␣␣␣␣␣␣␣
`RSet strFixe = "ABC"`	␣␣␣␣␣␣␣ABC
`LSet strFixe = "ABC"`	ABC␣␣␣␣␣␣␣
`RSet strFixe = Format(5.6, "Fixed")`	␣␣␣␣␣␣5,60
`RSet strFixe = Format(6665.6, "Fixed")`	␣␣␣6665,60
`RSet strFixe = Format(0.689, "Fixed")`	␣␣␣␣␣␣0,69
`RSet strFixe = Format(-85.6, "Fixed")`	␣␣␣␣-85,60

Exemple d'un rapport tabulé comportant 3 colonnes

```
Dépôt              500,00 $          500,00 $
Dépôt            6 000,90 $        6 500,90 $
Retrait          6 342,88 $          158,02 $
Dépôt               56,73 $          214,75 $
```

```
Private Sub ImprimeLigne()
  Dim strFixeMontant As String * 12
  Dim strFixeSolde As String * 14
  Dim sngMontant As Single
  Static sngSolde As Single

  sngMontant = CSng(txtMontant.Text)
  RSet strFixeMontant = FormatCurrency(sngMontant)
  If txtTypeTransac.Text = "Retrait" Then
    sngSolde = sngSolde - sngMontant
  Else
    sngSolde = sngSolde + sngMontant
  End If
  RSet strFixeSolde = FormatCurrency(sngSolde)
  Printer.Print Tab(10); txtTypeTransac.Text; _
                Tab(25); strFixeMontant; _
                Tab(40); strFixeSolde
End Sub
```

- Le type de transaction (Dépôt ou Retrait) et le montant sont saisis à l'écran.
- Le solde prend la valeur 0 lors de la déclaration et il est réduit ou augmenté selon le type de transaction.

- Les montants sont imprimés avec le format Currency et sont alignés à droite par l'instruction RSet.

Exercice 24 Imprimer le fichier de données du projet *Inscription*

Étape 1 Préparation

1. À partir de l'Explorateur de Windows, copiez le fichier Etudiant.txt du volume, stocké dans le dossier Chapitre7 /Exercice24 /Debut dans un nouveau dossier .../Visual Basic /Exercice 24 Impression.

2. Dans Visual Basic, créez un nouveau projet et nommez-le **Impression**. *Supprimez* Form1 du projet et enregistrez le projet sous le nom **Impression.vbp** dans le dossier .../Visual Basic /Exercice 24 Impression.

Étape 2 Créer un projet sans feuille

1. Ajoutez un *module standard* au projet, nommez-le **Rapport** et enregistrez-le dans le dossier du projet sous le nom **Rapport.Bas**.

2. Ajoutez une procédure **Sub Main** dans le module standard :

```
Private Sub Main()
  Call Initialisations
  Call DebutRapport
  Call Rapport
  Call FinRapport
  End
End Sub
```

> L'exécution du projet démarre par la procédure **Sub Main**. Par la suite, les procédures Initialisations, DebutRapport, Rapport et FinRapport sont appelées successivement et l'exécution se termine à l'instruction End.

3. Déclarez des variables et tableaux de portée module :

```
Dim mstrNomFichier As String          'chemin d'accès et nom de fichier
Dim mstrVille(6) As String            'tableau des villes
Dim mstrOrientation(4) As String      'tableau des orientations
Dim mintLigne As Integer              'compteur de lignes
Dim mintFille As Integer              'compteur de filles
Dim mintGarcon As Integer             'compteur de garçons
```

4. Ajoutez la procédure Initialisations :

```
Private Sub Initialisations()
  mstrNomFichier = App.Path & "\Etudiant.txt"
  mstrVille(0) = "Chicoutimi"
  mstrVille(1) = "Gaspé"
  mstrVille(2) = "Granby"
  mstrVille(3) = "Montréal"
  mstrVille(4) = "Québec"
  mstrVille(5) = "Sherbrooke"
  mstrVille(6) = "Trois-Rivières"
  mstrOrientation(0) = "Bureautique"
  mstrOrientation(1) = "Graphisme"
  mstrOrientation(2) = "Informatique"
  mstrOrientation(3) = "Multimédia"
  mstrOrientation(4) = "Réseautique"
End Sub
```

Le chemin d'accès et le nom du fichier sont placés dans la variable mstrNomFichier. Pour retrouver le fichier lors de l'ouverture, le chemin d'accès doit être précisé. Une constante ne peut être utilisée dans le cas présent, parce que la valeur de la propriété Path ne peut être affectée à une constante lors de la déclaration de celle-ci.

Les valeurs des tableaux mstrVille et mstrOrientation sont chargées dans ceux-ci. Dans l'exercice 23, lors de la création du fichier Etudiant.txt, seulement les index des villes et des orientations ont été enregistrés. Les tableaux mstrVille et mstrOrientation seront utilisés pour imprimer les noms de villes et les orientations associés aux index.

Étape 3 Produire un rapport tabulé avec pagination et totalisations

1. Ajoutez la procédure DebutRapport :

```
Private Sub DebutRapport()
On Error GoTo Erreur
  Open mstrNomFichier For Input As #1
  Exit Sub
Erreur:
  MsgBox "Le fichier des étudiants " & mstrNomFichier & _
         " ne peut être ouvert.", vbExclamation, _
         "Erreur " & Err.Number & " : " & Err.Description
End Sub
```

L'instruction **Open** ouvre le fichier en mode **Input**. Un gestionnaire d'erreurs permet d'intercepter toute erreur d'accès au fichier.

2. Testez les procédures Initialisations et DebutRapport avant de poursuivre.

3. Ajoutez la procédure Rapport :

```
Private Sub Rapport()
  Dim strNo As String
  Dim strNom As String * 25, strCourriel As String * 30
  Dim intVille As Integer, intOrient As Integer
  Dim blnSexe As Boolean, strSexe As String * 1
```

Des variables de type String avec une longueur fixe (strNom et strCourriel) sont utilisées pour limiter la longueur des valeurs lues et imprimées. En supposant qu'un long nom de 35 caractères soit enregistré dans le fichier et qu'une variable de longueur variable soit utilisée, les éléments qui suivent le nom sur la même ligne seront décalés vers la droite et le rapport ne sera pas tabulé correctement.

```
  Call EnteteRapport
  On Error GoTo Erreur

  Do While Not EOF(1)
    Input #1, strNo, strNom, intVille, intOrient, _
                             blnSexe, strCourriel
    If blnSexe = True Then
      strSexe = "F"
      mintFille = mintFille + 1
    Else
      strSexe = "M"
      mintGarcon = mintGarcon + 1
    End If
    Printer.Print strNom; Tab(27); mstrVille(intVille); _
                  Tab(43); mstrOrientation(intOrient); _
                  Tab(58); strSexe, Tab(63); strCourriel
    mintLigne = mintLigne + 1
    If mintLigne = 30 Then
      Printer.NewPage
      Call EnteteRapport
    End If
  Loop
  Exit Sub
```

```
Erreur:
  MsgBox "Une erreur s'est produite, " & _
         "la production du rapport est interrompue. ", _
          vbCritical, _
         "Erreur " & Err.Number & " : " & Err.Description
End Sub
```

Les données sont lues par l'instruction **Input** et affectées à des variables.

Des compteurs sont utilisés pour calculer le nombre de filles et le nombre de garçons (mintFille et mintGarcon).

Pour chaque enregistrement, le nom, la ville, l'orientation, le sexe (F ou M) et le courriel sont imprimés sur une même ligne. Une page contient au plus, 30 enregistrements. Lorsque 30 étudiants sont imprimés, la méthode **NewPage** provoque un saut de page et la procédure EnteteRapport est appelée pour générer l'entête de page.

4. Ajoutez la procédure EnteteRapport :

```
Private Sub EnteteRapport()
  mintLigne = 0
  Printer.FontName = "Courier New"
  Printer.FontBold = True
  Printer.FontSize = 12
  Printer.Print
  Printer.Print Tab(20); "Liste des étudiants inscrits"; _
                         Tab(60); "Page"; Printer.Page
  Printer.Print
  Printer.FontSize = 9
  Printer.FontBold = False
  Printer.Print "NOM"; Tab(27); "VILLE"; Tab(43); _
                "ORIENTATION"; Tab(56); "SEXE"; _
                Tab(63); "COURRIEL"
  Printer.Print String(85, "_")
  Printer.Print
End Sub
```

La police à espacement fixe Courier New est utilisée afin de produire un rapport tabulé dont les résultats sont bien alignés. Le style **FontBold** et la taille **FontSize** sont définis et la propriété **Page** de l'objet Printer retourne le numéro de la page en cours.

La fonction **String** permet de tracer une ligne sous les entêtes.

5. Ajoutez la procédure FinRapport :

```
Private Sub FinRapport()
  Dim strFixe As String * 7
  Printer.Print
  Printer.Font.Bold = True
  RSet strFixe = mintFille
  Printer.Print Tab(30); "Nombre de filles :"; Tab(50); strFixe
  RSet strFixe = mintGarcon
  Printer.Print Tab(30); "Nombre de garçons :"; Tab(50); strFixe
  RSet strFixe = mintGarcon + mintFille
  Printer.Print Tab(30); "Nombre d'étudiants :"; Tab(50); strFixe
  Printer.EndDoc
  Close #1
End Sub
```

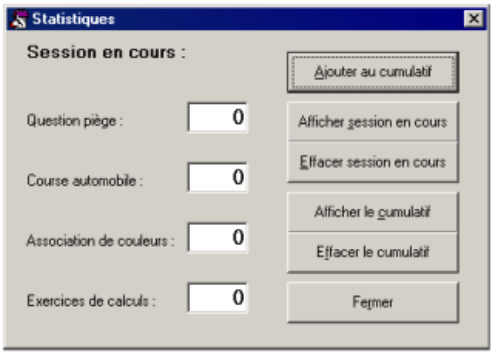
> L'instruction **RSet** est utilisée pour aligner à droite les trois totaux. La méthode **EndDoc** envoie le rapport à l'imprimante et l'instruction **Close** ferme le fichier de données.

6. Exécutez le projet, observez attentivement les résultats en les comparant aux données enregistrées sur le fichier Etudiant.txt.

Exercice 25 Modifier le projet *Regroupement de projets*

Étape 1 Préparation

1. À partir de l'Explorateur de Windows, copiez tous les fichiers du dossier .../Visual Basic /Exercice 17 Regroupement dans un nouveau dossier .../Visual Basic /Exercice 25 Statistiques.
2. Copiez les deux fichiers du volume stockés dans le dossier Chapitre7 /Exercice25 /Debut, dans le dossier.../Visual Basic /Exercice 25 Statistiques.
3. Ouvrez le projet **Regroupement.vbp** stocké dans le dossier .../Visual Basic /Exercice 25 Statistiques.
4. Ajoutez au projet le fichier **Statistiques.frm** stocké dans le dossier du projet.
5. Observez les éléments de la feuille Statistiques.frm.



La fenêtre Statistiques doit afficher le nombre de fois que l'utilisateur a choisi les différents projets : "Question piège", "Course automobile", "Association de couleurs" et "Exercices de calculs". Par défaut, les statistiques sont affichées pour la session de travail en cours. En cliquant sur le bouton "Afficher le cumulatif", l'utilisateur peut visualiser les statistiques accumulées pendant plusieurs sessions de travail.

Les étiquettes qui afficheront les résultats, forment le groupe de contrôles lblTotal (0) à lblTotal(3).

Étape 2 Programmation des statistiques de la session en cours

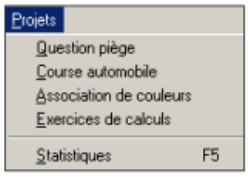
1. Ajoutez un module standard au projet, nommez-le ModuleStandard et enregistrez-le dans le dossier du projet sous le nom ModuleStandard.Bas.

2. Dans le module standard, déclarez un tableau à une dimension de portée globale **gintSession(3)**, qui servira à accumuler les statistiques de la session en cours. Le tableau doit être global parce qu'il sera utilisé par au moins 2 feuilles.

   Tableau gintSession(3) :

(0)	(1)	(2)	(3)

3. Ajoutez le choix Statistiques au menu Projets. Nommez-le **mnuStatistiques**.



4. Lorsque le choix Statistiques est sélectionné, affichez le feuille modale frmStatistiques. Codez et testez.

5. Dans la feuille frmPrincipale, accumulez les statistiques dans le tableau global gintSession. Ajoutez l'instruction qui suit dans la procédure cmdProjet_Click :

```
gintSession(Index) = gintSession(Index) + 1
```

> À chaque fois que l'utilisateur sélectionne l'un des 4 projets, la valeur 1 est ajoutée à l'élément correspondant du tableau.

6. Ajoutez la procédure personnalisée AfficherSession et déclarez la variable I au niveau module :

```
Private Sub AfficherSession()
  lblTypeStats.Caption = "Session en cours : "
  For I = 0 To 3
    lblTotal(I).Caption = gintSession(I)
  Next
End Sub
```

> Tous les résultats du tableau gintSession sont placés dans les étiquettes correspondantes.

7. Lorsque la forme frmStatistiques est affichée, affichez les statistiques de la session en cours (procédure déjà codée).

8. Exécutez le projet et sélectionnez plusieurs fois les différents projets. Par la suite, sélectionnez le choix Statistiques et observez les résultats.

9. Codez le bouton cmdAfficher qui fait appel à la procédure AfficherSession.

10. Codez le bouton cmdEffacer qui doit accomplir les tâches suivantes :

    - Initialiser le tableau des statistiques ;
    - Afficher de nouveau les statistiques pour démontrer à l'utilisateur qu'elles sont bien effacées ;
    - Rendre inaccessibles les boutons cmdAjouter et cmdEffacer.

11. Testez les boutons cmdAfficher et cmdEffacer.

Étape 3 Programmation des statistiques cumulatives

1. Dans le *module standard* :

   - Déclarez un tableau global **glngCumulatif(3)** qui servira à accumuler les statistiques pendant plusieurs sessions de travail.

     Tableau glngCumulatif(3) :

(0)	(1)	(2)	(3)

   - Déclarez trois variables de portée module avec le mot clé Dim : mstrCheminAcces, mstrMessage et I.

   - Ajoutez la procédure *publique* ChargementStats :

```
Public Sub ChargementStats()
  On Error GoTo GestionErreur
  mstrCheminAcces = App.Path & "\Stats.txt"
  Open mstrCheminAcces For Input As #1
  I = 0
  Do While Not EOF(1)
    Input #1, glngCumulatif(I)
    I = I + 1
  Loop
  Close #1
 If I < 4 Then
    Err.Raise 700, , "Fichier incomplet"
  End If
  Exit Sub
```

Le fichier des statistiques est ouvert, lu et fermé. La variable I est utilisée pour compter le nombre d'enregistrements dans le fichier. Celui-ci devrait comporter 4 enregistrements, un par projet disponible. Si ce n'est pas le cas, la méthode **Raise** de l'objet **Err** est utilisée pour générer l'erreur 700 qui sera traitée par le gestionnaire d'erreurs.

La méthode Raise de l'objet Err est traitée dans la section L'objet *Err* et les principaux codes d'erreur, page 277.

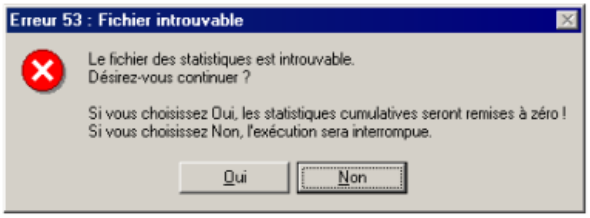
```
GestionErreur:
  Select Case Err.Number
    Case 53
      mstrMessage = _
          "Le fichier des statistiques est introuvable."
```

```
      Call AffichageErreur
    Case 700
      mstrMessage = _
        "Il manque des enregistrements dans le " & _
          " fichier des statistiques."
      Call AffichageErreur
      Close #1
    Case Else
      mstrMessage = _
        "Problème non identifiable avec le fichier" & _
            " des statistiques."
      Call AffichageErreur
      Close #1
    End Select
End Sub
```

Des messages significatifs sont affectés à la variable mstrMessage. Dans le cas de l'erreur 53, il ne faut pas fermer le fichier parce qu'étant introuvable, il n'est certainement pas ouvert. Dans les deux autres cas, le fichier doit être fermé par l'instruction Close.

- Codez la procédure publique AffichageErreur qui doit accomplir les tâches suivantes :

  Affichez le message qui suit :



  La première ligne du message est le contenu de la variable mstrMessage. La barre de titre contient le numéro et la description de l'objet Err. Le bouton Non doit être le bouton par défaut parce qu'il est le moins destructif.

  Lorsque l'utilisateur répond **Oui**, faites appel à une procédure publique **EffacerCumulatif** qui initialisera à zéro tous les éléments du tableau **glngCumulatif**. Codez la procédure EffacerCumulatif.

Lorsque l'utilisateur répond **Non**, arrêtez immédiatement l'exécution par l'instruction End.

- Ajoutez la procédure publique EnregistrementStats :

```
Public Sub EnregistrementStats()
  On Error GoTo GestionErreur
  Open mstrCheminAcces For Output As #1
  For I = 0 To 3
    Write #1, glngCumulatif(I)
  Next
  Close #1
  Exit Sub
GestionErreur:
  MsgBox "L'enregistrement des statistiques " & _
         "s'est terminé anormalement.", vbExclamation, _
         "Erreur " & Err.Number & " : " & Err.Description
End Sub
```

Cette procédure est appelée à la fin de l'exécution du projet. Le fichier des statistiques est créé (mode Output) et 4 enregistrements y sont stockés.

2. Dans la forme frmPrincipale, lors de l'événement Form_Load, faites appel à la procédure publique ChargementStats. Lors de l'événement Form_Unload, faites appel à la procédure publique EnregistrementsStats.

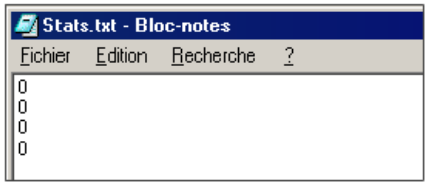
Note : Au démarrage du projet, le fichier des statistiques est ouvert, lu, fermé et les valeurs sont chargées dans le tableau des statistiques cumulatives.

À la fin de l'exécution, les statistiques cumulatives sont enregistrées sur le fichier afin qu'elles soient à jour lors d'une prochaine exécution.

3. Exécutez le projet. Le message d'erreur 53 devrait s'afficher lors de l'ouverture du fichier des statistiques. Cette erreur est normale puisque le fichier n'existe pas. Répondez **Non** au message et assurez-vous que l'interruption de l'exécution se déroule normalement.

4. Exécutez une seconde fois et répondez **Oui** lors de l'affichage du message d'erreur. Par la suite, sélectionnez le choix Quitter dans la forme frmPrincipale.

5. À partir de l'Explorateur de Windows, observez le contenu du fichier Stats.txt. Assurez-vous que le fichier contient 4 enregistrements avec la valeur 0 :



6. Exécutez le projet une troisième fois. Le message d'erreur 53 ne devrait plus être affiché parce que cette fois, le fichier existe. Sélectionnez le choix Quitter.

7. Pour tester l'erreur 700, procédez comme suit :
   - À l'aide de l'application Bloc-Notes de Windows, enlevez 2 enregistrements dans le fichier Stats.txt.
   - Exécutez et observez le message d'erreur 700. Répondez Oui pour continuer et par la suite, sélectionnez le choix Quitter.

8. Codez le bouton cmdAjouter qui doit accomplir les tâches suivantes :
   - Accumuler toutes les statistiques de la session en cours dans le tableau glngCumulatif ;
   - Appeler la procédure cmdEffacer_Click afin d'effacer les statistiques de la session, celles-ci ayant déjà été ajoutées aux statistiques cumulatives.

9. Testez le bouton cmdAjouter. Choisissez plusieurs projets avant de sélectionner le bouton cmdAjouter.

10. Codez le bouton cmdAfficherCumul qui doit accomplir les tâches suivantes :
    - Afficher le titre Cumulatif : dans l'étiquette lblTypeStats ;
    - Afficher tous les éléments du tableau glngCumulatif dans les étiquettes lblTotal.

11. Testez le bouton cmdAfficherCumul.

12. Codez le bouton cmdEffacerCumul qui doit accomplir les tâches suivantes :
    - Initialiser à zéro le tableau glngCumulatif (procédure déjà codée) ;
    - Afficher les statistiques cumulatives pour que l'utilisateur puisse constater qu'elles sont bien effacées (procédure déjà codée) ;

- Rendre inaccessible le bouton cmdEffacerCumul, mais seulement si le bouton cmdAjouter n'est pas accessible.

13. Testez le bouton cmdEffacerCumul.

14. Exécutez le projet et testez minutieusement les statistiques en cours et les statistiques cumulatives.

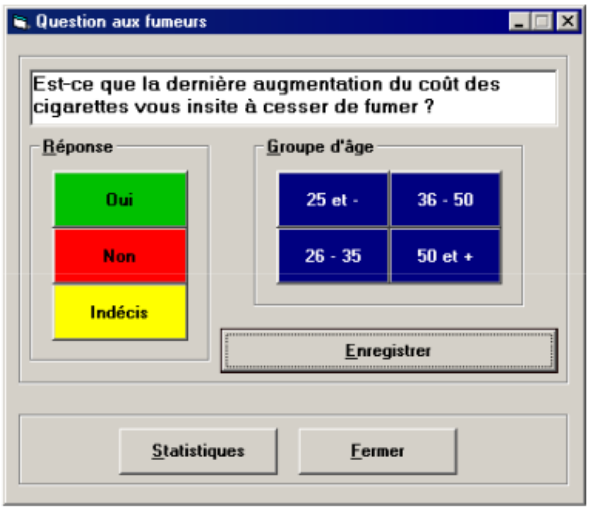
Exercice 26 Créer le projet *Sondage*

Étape 1 Préparation et observation

1. À partir de l'Explorateur de Windows, copiez tous les fichiers du volume, stockés dans le dossier Chapitre7 /Exercice26 /Debut, dans un nouveau dossier .../Visual Basic /Exercice 26 Sondage.

2. Ouvrez le projet **Sondage.vbp** stocké dans le dossier .../Visual Basic /Exercice 26 Sondage. Il comporte 2 feuilles ainsi qu'un module standard.

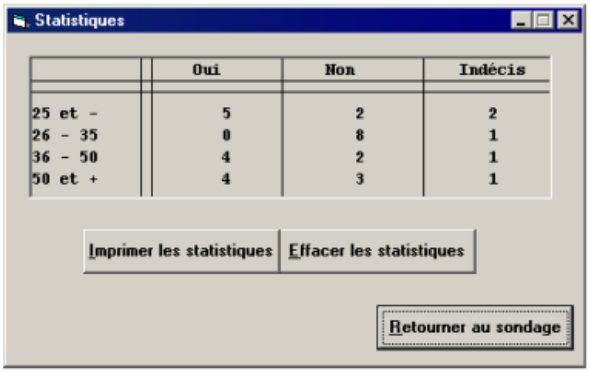
3. Observez la feuille frmSondage :



- Cette interface graphique sert à une personne qui effectue un sondage téléphonique auprès des fumeurs. À chaque fois qu'un individu répond aux questions, l'utilisateur clique sur un bouton dans chacun des deux groupes et termine le sondage en cliquant sur le bouton Enregistrer. Les réponses sont alors enregistrées dans un fichier séquentiel.

- Notez que les boutons utilisés sont des boutons d'option avec la propriété **Style = 1-Graphical**. Deux groupes de boutons ont été créés et dans chacun des groupes, *un seul bouton* peut être sélectionné.
- Les index des groupes de boutons débutent à 1. On retrouve ainsi les contrôles optRep(1) à optRep(3) et optAge(1) à optAge(4).

4. Observez la feuille frmStats :



- Les statistiques par groupe d'âges et par réponse sont affichées dans un contrôle PictureBox par la méthode Print. Notez que cette méthode s'utilise de la même façon qu'avec l'objet Printer.

Étape 2 Enregistrer des statistiques sur un fichier séquentiel

1. Déclarez les éléments suivants dans le module standard :

```
Option Base 1

Public gstrNomFichier As String  'chemin d'accès et nom du fichier
Public gstrAge(4) As String      'tableau des descriptions d'âges
```

25 et -	26 - 35	36 - 50	50 et +

```
Public gstrRep(3) As String      'tableau des choix de réponses
```

Oui	Non	Indécis

La ligne de code `Option Base 1` spécifie que l'indice minimal des tableaux du module est 1.

Deux tableaux à une dimension sont utilisés pour afficher les titres sur la feuille frmStats et dans un rapport imprimé.

2. Dans la forme frmSondage, ajoutez les instructions ou traduisez le pseudo-code en Visual Basic :

```
Option Base 1
Dim mintAge As Integer, mintRep As Integer

Private Sub Form_Load()
  Affecter le chemin d'accès et le nom du fichier de données "Sondage.txt"
  à la variable globale gstrNomFichier.

  Charger les valeurs dans les tableaux gstrAge et gstrRep. Exemple :
  gstrRep(2) = "Non".
End Sub

Private Sub Form_Activate()
  Ouvrir le fichier de données en mode ajout.
End Sub
```

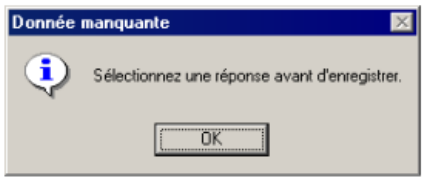
Le fichier est ouvert lors de l'événement Activate, c'est-à-dire au démarrage et après qu'une feuille frmStats ait été affichée. Le fichier étant fermé dans le module de code de la feuille frmStats, il doit être nécessairement ouvert de nouveau pour enregistrer d'autres données. Un fichier doit être fermé avant d'être réouvert dans un autre mode.

3. Lors de l'ouverture du fichier de données, codez un gestionnaire d'erreurs afin d'intercepter toute erreur éventuelle. Dans un tel cas, affichez un message significatif contenant le numéro de l'erreur et sa description.
4. Lorsque l'utilisateur ferme la feuille par le bouton Fermer ou par le bouton X de la barre de titre, *fermez* le fichier ouvert avant de décharger la feuille.
5. Lorsque l'utilisateur sélectionne un groupe d'âges et une réponse, conservez la valeur des index dans les variables mintAge et mintRep. Ce sont ces deux variables qui seront enregistrées sur le fichier de données.

6. Ajoutez les instructions ou traduisez le pseudo-code en Visual Basic :

```
Private Sub cmdEnregistrer_Click()
  Afficher le message suivant si aucun groupe d'âges n'a été sélectionné et
  sortir de la procédure :
```

```
  Afficher le message suivant si aucune réponse n'a été sélectionnée et
  sortir de la procédure :
```




```
  On Error GoTo Erreur
    Enregistrer les variables mintAge et mintRep sur le fichier séquentiel.
    Relâcher les 2 boutons d'option qui ont été pressés en vue d'un
    prochain sondage.
    Exit Sub
Erreur:
  MsgBox "Les données n'ont pas été enregistrées", _
         vbInformation, "Erreur " & _
         Err.Number & " " & Err.Description
End Sub
```

7. Codez la procédure du bouton cmdStats qui doit accomplir les tâches suivantes :
   - Fermer le fichier Sondage.txt ;
   - Cacher la feuille frmSondage à l'utilisateur ;
   - Afficher la feuille modale frmStats.

Étape 3 Utiliser un tableau en mémoire à 2 dimensions

1. Ajoutez les déclarations suivantes dans la feuille frmStats :

```
Option Base 1
Dim mintTableau(4, 3) As Integer
Dim A As Integer, R As Integer        'Indices du tableau mintTableau
```

Le tableau mintTableau à 2 dimensions est utilisé pour accumuler les données enregistrées par groupe d'âges et par réponse.

Tableau mintTableau :

	Oui	Non	Indécis
25 et -			
26 - 35			
36 - 50			
50 et +			

Les indices A et R représentent respectivement l'âge et la réponse.

2. Ajoutez les instructions ou traduisez le pseudo-code en Visual Basic :

```
Private Sub Form_Load()
  Dim intAge As Integer, intRep As Integer
  Call InitialiserTableau
  Ouvrir le fichier de données en mode lecture.

  Lire les données intAge et intRep dans chacun des enregistrements du
  fichier et les accumuler dans le tableau mintTableau. Instruction pour
  accumuler :

    mintTableau(intAge, intRep) = mintTableau(intAge, intRep) + 1

  Fermer le fichier de données.
  Call Affichage
End Sub
```

Le tableau mintTableau a une portée module et est *initialisé seulement au début de l'exécution*, d'où la nécessité de l'initialiser à chaque fois que la feuille est chargée. Les données étant accumulées dans le tableau, après quelques affichages de la feuille frmStats, les statistiques auront doublées et par la suite, triplées.

3. Ajoutez la procédure personnalisée InitialiserTableau. Elle doit affectée la valeur 0 à tous les éléments du tableau mintTableau.

0	0	0
0	0	0
0	0	0
0	0	0

```
Public Sub InitialiserTableau()
  For A = 1 To 4
    For R = 1 To 3
      mintTableau(A, R) = 0
    Next
  Next
End Sub
```

4. Ajoutez la procédure personnalisée Affichage :

```
Private Sub Affichage()
  Dim strFixe As String * 7
  picStats.Print ,
  For R = 1 To 3
    picStats.Print "    " & gstrRep(R),
  Next
  picStats.Print
```

Les lignes de code qui précèdent, forment une boucle qui affiche les 3 réponses (Oui, Non, Indécis) sur une même ligne du contrôle PictureBox. En terminant la méthode Print par une virgule, la prochaine impression est effectuée sur la même ligne.

```
  picStats.Print
  For A = 1 To 4
    picStats.Print gstrAge(A),
    For R = 1 To 3
      RSet strFixe = mintTableau(A, R)
      picStats.Print strFixe,
    Next
    picStats.Print  'saut de ligne à la fin d'un groupe d'âges
  Next
End Sub
```

Les lignes de code qui précèdent, forment une boucle imbriquée qui affiche les 4 lignes du tableau. Chaque ligne débute par le groupe d'âges `gstrAge(A)`, qui est suivi des 3 totaux pour ce groupe d'âges `mintTableau(A, R)`. Les totaux sont alignés à droite par l'instruction RSet.

5. Codez le bouton cmdRetourner qui doit décharger la feuille frmStats et afficher la feuille frmSondage.

6. Exécutez le projet et enregistrez plusieurs sondages. Sélectionnez le bouton Statistiques et assurez-vous que les résultats affichés correspondent aux données entrées. Par la suite, sélectionnez le bouton "Retourner au sondage" et entrez d'autres sondages. Vérifiez de nouveau les statistiques et corrigez toute anomalie avant de continuer.

Étape 4 Imprimer un rapport

1. Codez la procédure du bouton cmdImprimer qui doit produire le rapport suivant :

Statistiques du sondage effectué le 2002-07-12

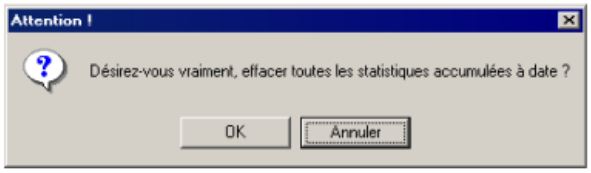
	25 et -	26 - 35	36 - 50	50 et +	Total
Oui	13	2	2	0	17
Non	0	2	0	1	3
Indécis	0	0	0	0	0

   - La police utilisée est Courier New, la taille des caractères est 10 et le rapport est entièrement imprimé avec le style gras.
   - La date du jour apparaît dans le titre.
   - Tous les résultats numériques incluant le total sont alignés à droite.
   - Pour vous guider, utilisez le code de la procédure Affichage. Cependant, certains ajustements sont nécessaires parce que le tableau n'est pas imprimé dans le même sens que sur la feuille frmStats.

2. Testez le bouton cmdImprimer.

3. Codez la procédure du bouton cmdEffacer. Ajoutez les instructions ou traduisez le pseudo-code en Visual Basic :

```
Private Sub cmdEffacer_Click()
  Afficher le message qui suit à l'utilisateur. S'il presse le bouton
  "Annuler", sortir de la procédure. S'il presse le bouton Oui, les
  instructions suivantes seront exécutées.
```




```
  Close #1
  Kill (gstrNomFichier)
  cmdEffacer.Enabled = False
```

```
  cmdImprimer.Enabled = False
  Call InitialiserTableau
  picStats.Cls
  Call Affichage
End Sub
```

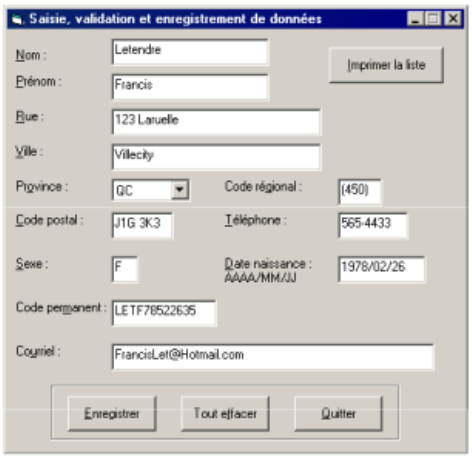
> L'instruction **Kill** supprime le fichier des statistiques qui doit d'abord être fermé.
>
> Le tableau des statistiques doit être initialisé et réaffiché après que le contenu du contrôle PictureBox ait été effacé par la méthode **Cls** (Clear Screen).

4. Testez le bouton cmdEffacer.
5. Testez l'ensemble du projet et assurez-vous de l'exactitude des résultats.

Exercice 27 Créer le projet *Validation de données*

Étape 1 Préparation

1. À partir de l'Explorateur de Windows, copiez tous les fichiers du volume, stockés dans le dossier Chapitre7 /Exercice27 /Debut, dans un nouveau dossier .../Visual Basic /Exercice 27 Validation.
2. Ouvrez le projet SaisieDonnees.vbp stocké dans le dossier.../Visual Basic /Exercice 27 Validation.
3. Observez l'interface graphique. Celle-ci sera complétée à l'étape 2 afin d'obtenir la fenêtre suivante :



Étape 2 Utiliser le contrôle MaskEdBox

1. Ajoutez un contrôle **MaskEdBox** à la boîte à outils par Projet /Composants /Microsoft Masked Edit Control 6.0.

Consultez la section 4.8, Les masques de saisie (contrôle MaskEditBox).

2. Utilisez un contrôle MaskEdBox pour la saisie du code régional, du code postal, du téléphone, de la date de naissance et du code permanent.

Code régional (nom : mskCodeReg)

- Le code régional est encadré de parenthèses et formé de 3 chiffres. Exemple : (514)

Code postal (nom : mskCodePostal)

- Le masque de saisie est formé de 7 caractères : lettre, chiffre, lettre, espace, chiffre, lettre et chiffre. Les lettres doivent être converties en majuscules lors de la saisie.

Téléphone (nom : mskTelephone)

- Le téléphone est saisi sous la forme 000-00000, 0 représentant un chiffre.

Date de naissance (nom : mskDate)

- La date de naissance est formée de 8 chiffres et est saisie sous la forme AAAA/MM/JJ.

Code permanent (nom : mskCodePerm)

- Le code permanent est saisi sous la forme AAAA0000000 (4 lettres majuscules suivies de 8 chiffres).

Le code permanent est construit comme suit :

Les 3 premières lettres correspondent aux 3 premières lettres du nom ;

La quatrième lettre correspond à la première lettre du prénom ;

Les 6 premiers chiffres correspondent à la date de naissance et identifient le sexe. Pour les femmes, le mois de la naissance est augmenté de 50 ;

Les 2 derniers chiffres forment un numéro séquentiel qui assure l'unicité de chaque code permanent.

Exemple :	Francis Letendre, né le 27 avril 1980	LETF80042712
	France Letendre, née le 27 avril 1980	LETF80542707

3. Pour la saisie de la province, un contrôle ComboBox non modifiable est utilisé. Observez son contenu. Il contient 12 codes représentant les 10 provinces canadiennes plus les Territoires du Nord-Ouest (NT) et le Yukon (YT). Ces codes sont ceux établis par la Société des Postes du Canada.

4. Fixez un nombre maximal pour le nombre de caractères saisis dans les zones de texte suivantes :

nom:	25 caractères
prénom :	25 caractères
rue :	30 caractères
ville :	30 caractères
sexe :	1 caractère
courriel :	40 caractères

> La propriété MaxLength des contrôles MaskEdBox est automatiquement ajustée avec celle de la propriété Mask.

5. Inscrivez la lettre F majuscule comme valeur par défaut dans la zone de saisie du sexe.

6. Testez l'ordre de déplacement de la touche Tab et apportez les modifications au besoin.

   Ordre de tabulation : nom, prénom, rue, ville, province, code postal, code régional, téléphone, sexe, date de naissance, code permanent, courriel et les boutons Enregistrer, Tout effacer, Quitter et Imprimer la liste.

Étape 3 Coder les procédures

1. La province sélectionnée par défaut est le Québec.

2. Déchargez la feuille de la mémoire lorsque l'utilisateur presse le bouton Quitter.

3. Ajoutez une procédure personnalisée ValiderNomPrenom pour valider les caractères dans les zones du nom et du prénom, lors de l'événement KeyPress. Seulement les lettres minuscules, les lettres majuscules, le caractère d'espacement, le tiret et la touche BackSpace sont acceptés. La procédure ValiderNomPrenom doit utiliser l'argument KeyAscii (réf. Exercice 14, Auto.vbp).

4. Testez la saisie du nom et du prénom.

5. Ajoutez une deuxième procédure personnalisée EpurerNomPrenom qui doit être appelée lorsque les zones du nom et du prénom perdent le focus. Cette nouvelle

procédure utilise la zone de texte à épurer comme argument et elle doit accomplir les tâches suivantes :

- Enlever tout caractère d'espacement inutile qui se trouve à *gauche* du nom ou du prénom ;
- Convertir la première lettre du nom ou du prénom en majuscule et les autres lettres en minuscules ;
- Réafficher le nom ou le prénom après modification.

Consultez la section Les fonctions de chaînes de caractères, page 287.

Il est souvent nécessaire de trier des données sur le nom des personnes afin de produire des listes alphabétiques. Si des noms débutent par des espaces, ceux-ci seront mal classés et difficilement retrouvables, d'où l'importance de supprimer les espaces au début des noms.

6. Testez les modifications effectuées au numéro 5.
7. Lorsque l'utilisateur quitte la zone de texte du sexe, convertissez le contenu de la zone en majuscule. Codez les instructions nécessaires.
8. Programmez le bouton "Tout effacer" :
   - Fixez les valeurs suivantes à la propriété **Tag** des différentes zones de saisie :

     valeur E nom, prénom, rue, ville, courriel, code postal, code régional, téléphone, date de naissance, code permanent

     valeur F sexe

     valeur QC province

La propriété **Tag** permet d'affecter une chaîne de caractères qui peut être éventuellement vérifiée dans le code de programmation.

Consultez la section La propriété *Tag*, page 195.

Dans le cas présent, la lettre E spécifie d'effacer le contenu de la zone de saisie, la lettre F spécifie d'inscrire la valeur F par défaut dans la zone de saisie du sexe et QC spécifie la province par défaut.

Lorsqu'un masque de saisie est utilisé, la propriété **PromptInclude** doit avoir la valeur **False** pour effacer le contenu de la zone de saisie sans effacer le masque.

9. Fixez la valeur **False** à la propriété **PromptInclude** de *tous les contrôles* MaskEdBox.

Étape 4 Accéder aux divers éléments de la collection Controls

1. Ajoutez le code de la procédure du bouton Tout effacer :

```
Private Sub cmdToutEffacer_Click()
  Dim Controle As Control
  For Each Controle In Me.Controls
    Select Case Controle.Tag
      Case "E"
        Controle.Text = ""
      Case "F"
        Controle.Text = "F"
      Case "QC"
        Controle.Text = "QC"
    End Select
  Next
  txtNom.SetFocus
End Sub
```

> La boucle **For Each** permet d'accéder à chacun des éléments de la collection **Controls**. Il est ainsi possible de vérifier la valeur de la propriété **Tag** de chacun des contrôles.

2. Testez le bouton "Tout effacer" et assurez-vous que toutes les zones de saisie sont initialisées correctement.

Étape 5 Créer un gestionnaire de clavier au niveau d'une feuille

Cette étape vise à créer un gestionnaire de clavier pour l'ensemble des zones de saisie de la feuille. Le gestionnaire de clavier doit permettre les options suivantes :

- Pour effacer le contenu d'une zone de saisie, l'utilisateur peut presser la touche **Esc**. Dans le cas de la province, la province par défaut QC doit être affichée.
- Pour atteindre la zone de saisie suivante, l'utilisateur peut utiliser la touche **Enter**.

Consultez la section Les événements clavier, Créer un gestionnaire de clavier au niveau d'une feuille, page 206.

1. Codez la touche **Esc** dans la procédure événementielle **Form_KeyPress** :

```
Private Sub Form_KeyPress(KeyAscii As Integer)
  Select Case KeyAscii
    Case vbKeyEscape
      With Me.ActiveControl
        Select Case .Tag
          Case "E", "F"
            .Text = ""
          Case "QC"
            .Text = "QC"
        End Select
      End With
      KeyAscii = 0
    Case vbKeyReturn
    'bloc de code complété au numéro 5
  End Select
End Sub
```

> La procédure Form_KeyPress est exécutée à chaque fois qu'une touche est pressée dans l'une des zones de saisie de la feuille.
>
> Dans l'expression Me.ActiveControl, **ActiveControl** représente le contrôle actif de la forme active **Me**, c'est-à-dire celui qui a le focus. Lorsque la touche Esc est pressée, la propriété **Tag** du contrôle actif est vérifiée et son contenu est effacé, ou initialisé avec la valeur QC dans le cas de la province.
>
> Lorsque la touche Esc a été traitée, l'instruction **KeyAscii = 0** permet d'annuler tout autre événement KeyPress sur une même zone de saisie, l'exécution d'une telle procédure devenant inutile.

2. Fixez la valeur **True** à la propriété **KeyPreview** de la feuille frmSaisie.

> L'affectation **KeyPreview = True** spécifie de reconnaître les événements clavier de la feuille *avant* les événements clavier de chacun des contrôles. Ainsi, la procédure Form_KeyPress est exécutée *avant* les procédures txtNom_KeyPress et txtPrenom_KeyPress.

3. Testez la touche **Esc** sur toutes les zones de saisie.

4. Avant de poursuivre, assurez-vous que la propriété **TabIndex** des différentes zones de saisie a bien les valeurs suivantes :

   nom = 1, prénom = 3, rue = 5, ville = 7, province = 9, code postal = 11, code régional = 13, téléphone = 15, sexe = 17, date de naissance = 19, code permanent = 21, courriel = 23

5. Codez la touche **Enter**. Ajoutez les lignes de code en gras à la procédure Form_KeyPress :

```
Private Sub Form_KeyPress(KeyAscii As Integer)
  Dim intIndexSuivant As Integer
  Dim Controle As Control
  Select Case KeyAscii
    Case vbKeyEscape

    Case vbKeyReturn
      If Me.ActiveControl.TabIndex = 23 Then
        intIndexSuivant = 1
      Else
        intIndexSuivant = Me.ActiveControl.TabIndex + 2
      End If
      For Each Controle In Me.Controls
        If Controle.TabIndex = intIndexSuivant Then
          Controle.SetFocus
          Exit For
        End If
      Next
      KeyAscii = 0
  End Select
End Sub
```

Lorsque la touche Enter est pressée, la zone suivante doit recevoir le focus. À l'aide de la propriété **TabIndex** du contrôle actif, il est possible de déterminer quelle est la zone suivante. Si sa valeur est 23, le contrôle actif est le courriel et la prochaine zone à recevoir le focus doit être le nom, dont la propriété TabIndex = 1. Dans tous les autres cas, la propriété TabIndex de la zone suivante a une valeur égale à la propriété TabIndex du contrôle actif + 2.

Une fois que la valeur de la propriété TabIndex de la zone suivante est trouvée et affectée à la variable intIndexSuivant, une boucle For est utilisée pour retrouver le contrôle correspondant dans la collection Controls. Lorsque le contrôle est trouvé, l'instruction Exit For permet de sortir de la boucle.

6. Testez la touche **Enter** sur toutes les zones de saisie et assurez-vous que l'exécution se déroule normalement.

Étape 6 Utiliser l'instruction SendKeys

Cette étape vise à coder la touche Enter en utilisant une deuxième méthode, l'instruction **SendKeys**.

1. Dans la procédure Form_KeyPress, mettez en commentaires toutes les lignes de code comprises entre **`Case vbKeyReturn`** et **`End Select`**.

2. Remplacez les lignes en commentaires par ces 2 lignes de code :

```
SendKeys "{tab}"
KeyAscii = 0
```

> L'instruction **SendKeys** permet d'envoyer une touche à la fenêtre active. La touche envoyée est traitée exactement comme si elle avait été pressée par l'utilisateur. Dans les lignes de code précédentes, la touche Tab est envoyée à la fenêtre active et la touche Enter est annulée par l'instruction `KeyAscii = 0`.
>
>
> Consultez la section Les événements clavier, l'instruction SendKeys, page 206.

3. Testez de nouveau la touche Enter sur toutes les zones de saisie. Cette fois, après la zone de saisie du courriel, le focus est donné au bouton Enregistrer, au lieu de la zone de saisie du nom, exactement comme si la touche Tab avait été pressée.

4. Étant donné que la première méthode est plus performante, mettez en commentaires les 2 dernières lignes de code qui ont été entrées et ne plus commenter les lignes de la première méthode.

Étape 7 Valider et générer des erreurs avec la méthode Raise de l'objet Err

Cette étape vise à valider les données lorsque l'utilisateur presse le bouton Enregistrer. Si celles-ci sont toutes valides, elles seront enregistrées sur un fichier (Étape 8).

1. Entrez les lignes de code suivantes ou traduisez le pseudo-code :

```
Private Sub cmdEnregistrer_Click()
  On Error GoTo MessageErreur
  If txtNom = "" Then
    txtNom.SetFocus
    Err.Raise 800, , "Le nom est obligatoire."
  End If
```

```
  If txtPrenom = "" Then
    txtPrenom.SetFocus
    Err.Raise 801, , "Le prénom est obligatoire."
  End If
  .
  .
  .
  'Call EnregistrementSurFichier     'en commentaire pour le moment
  Exit Sub
MessageErreur:
  Afficher le message d'erreur qui suit. Le numéro de l'erreur
  (Err.Number)apparaît dans la barre de titre et la description de l'erreur
  (Err.Description) est utilisée comme texte du message.
End Sub
```

2. Codez les validations des zones de saisie selon les spécifications qui suivent pour chacune des zones. Lorsqu'un message d'erreur est affiché, donnez le focus à la zone en erreur. Testez chacune des validations aussitôt que sa codification est complétée.

Nom

- obligatoire ;
- Erreur 800, message : "Le nom est obligatoire."

Prénom

- obligatoire ;
- Erreur 801, message : "Le prénom est obligatoire."

Rue

- obligatoire ;
- Erreur 802, message : "La rue est obligatoire."

Ville

- obligatoire ;
- Erreur 803, message : "La ville est obligatoire."

Code postal

- facultatif, mais s'il est présent, sa longueur doit être égale à 6 ;
- Erreur 804, message : "Le code postal est incomplet."

Code régional

- facultatif, mais s'il est présent, sa valeur doit être égale à l'un des 10 codes énumérés ci-après ;

514	418	450	819	506	902	709	416	647	613

- Erreur 805, message : "Le code régional est invalide."

> Utilisez un *tableau en mémoire* de portée module et charger les valeurs avec la fonction **Array** au démarrage. Codez une recherche dans le tableau à l'aide d'une boucle For Each ou For Next.

Téléphone

- facultatif, mais s'il est présent, sa longueur doit être égale à 7 ;
- Erreur 806, message : "Le téléphone est incomplet."

Sexe

- obligatoire et sa valeur doit être F ou M ;
- Erreur 807, message : "Le sexe est invalide."

Date de naissance

- facultative, mais si elle est présente, elle doit être une date valide et ne doit pas être plus grande que la date du jour ;
- Erreur 808, message : "La date est invalide." ou Erreur 809, message : "La date est plus grande que la date du jour."

Utilisez la fonction **IsDate** pour vérifier la validité de la date. Testez toutes les possibilités comme par exemple, mois = 13, jour = 32, 1999/02/29 etc. Avant de vérifier la validité de la date de naissance, affectez la valeur **True** à la propriété **PromptInclude** de la date et après la validation, réaffectez la valeur **False** (réf. 4.8, Les masques de saisie, la propriété PromptInclude).

Pour vérifier l'erreur 809, utilisez la fonction **Cdate** pour convertir la date de naissance et ainsi pouvoir la comparer avec la fonction **Date** qui retourne la date du jour du système.

Consultez la section Les fonctions de date et heure, page 292.

Code permanent

- facultatif, mais s'il est présent, sa longueur doit être égale à 12, les 3 premières lettres doivent être identiques aux 3 premières lettres du nom et la quatrième lettre doit être identique à la première lettre du prénom ;
- Erreur 810, message : "Le code permanent est incomplet." ou Erreur 811, message : "Le code permanent ne correspond pas au nom ou au prénom."

Courriel

- facultatif, mais s'il est présent, il doit contenir un caractère @ ;
- Erreur 812, message : "Le courriel ne contient pas de caractère @."

La validation des données est un moyen très efficace pour s'assurer de la robustesse d'un programme. Dans le présent projet, plusieurs validations pourraient encore être effectuées. Voici quelques exemples :

- La correspondance entre la date de naissance et le code permanent ;
- La correspondance entre le sexe et le code permanent ;
- Les lettres D, F, I, O, Q et U sont invalides dans un code postal ;
- La première lettre du code postal doit être conforme à la province. Par exemple, dans la province de Québec, tous les codes postaux commencent par la lettre G, H ou J et dans la province de Terre-Neuve, tous les codes postaux débutent par la lettre A.

Étape 8 Enregistrer des données sur un fichier à accès direct

1. Insérez les déclarations qui suivent au début du module de code :

```
'Structure d'enregistrement
Private Type Etudiant
  Nom As String * 25
  Prenom As String * 25
  Rue As String * 30
  Ville As String * 30
  Prov As String * 2
  CodePostal As String * 6
  CodeReg As String * 3
  Telephone As String * 7
  Sexe As String * 1
  DateNais As String * 8
  CodePerm As String * 12
  Courriel As String * 40
End Type

Dim mudtEtudiant As Etudiant  'variable d'enregistrement

Dim mblnOuvert As Boolean     'fichier déjà ouvert ou non
```

L'instruction **Type** définit une donnée de type personnalisé qui constitue une structure d'enregistrement. La variable d'enregistrement mudtEtudiant est déclarée avec le type de donnée personnalisé Etudiant.

Les enregistrements d'un *fichier à accès direct* doivent être de la *même longueur*. Toutes les variables de type String qui composent l'enregistrement doivent ainsi être de longueur fixe. Notez que les caractères qui forment les masques de saisie ne sont pas enregistrés. Par exemple, le code postal occupe une longueur de 6 octets sur le fichier, l'espace au centre n'étant pas enregistrée.

Consultez les sections 5.19, Les types de données personnalisés (instruction *Type*) et Les fichiers de données et leurs types d'accès, Accès à un fichier direct, page 354.

2. Enlevez l'apostrophe en avant de l'instruction qui fait appel à la procédure EnregistrementSurFichier dans la procédure cmdEnregistrer_Click.

3. Ajoutez la procédure personnalisée EnregistrementSurFichier :

```
Private Sub EnregistrementSurFichier()
  On Error GoTo ErreurEnregistrement
  If Not mblnOuvert Then
    mblnOuvert = True
    Open App.Path & "\Fichier.dat" For Random _
                  As #1 Len = Len(mudtEtudiant)
  End If
```

Le fichier est ouvert seulement s'il n'est pas ouvert. Il faut prévoir cette condition puisque ouvrir un fichier qui est déjà ouvert provoque une erreur d'exécution.

Le mode d'ouverture **Random** spécifie que le fichier est un fichier direct et qu'il est ouvert en mode lecture et écriture au début de celui-ci. Si le fichier n'existe pas lors de l'ouverture, il est automatiquement créé. La longueur doit obligatoirement être citée. Celle-ci est égale à la longueur (Len) de la variable mudtEtudiant.

```
  mudtEtudiant.Nom = txtNom.Text
  mudtEtudiant.Prenom = txtPrenom.Text
  mudtEtudiant.Rue = txtRue.Text
  mudtEtudiant.Ville = txtVille.Text
  mudtEtudiant.Prov = cboProv.Text
  mudtEtudiant.CodePostal = mskCodePostal.Text
  mudtEtudiant.CodeReg = mskCodeReg.Text
  mudtEtudiant.Telephone = mskTelephone.Text
  mudtEtudiant.Sexe = txtSexe.Text
  mudtEtudiant.DateNais = mskDate.Text
  mudtEtudiant.CodePerm = mskCodePerm.Text
  mudtEtudiant.Courriel = txtCourriel.Text

  Put #1, , mudtEtudiant
  MsgBox "Les données ont été enregistrées.", vbInformation, _
         "Confirmation d'enregistrement"
  Call cmdToutEffacer_Click
  Exit Sub
ErreurEnregistrement:
  MsgBox "Le fichier ne peut être ouvert ou une erreur " & _
         "est survenue lors de l'enregistrement. ", vbCritical, _
         "Erreur " & Err.Number & " : " & Err.Description
End Sub
```

Les données sont transférées dans la variable d'enregistrement avant d'être enregistrées. La première fois que l'instruction **Put** est rencontrée, un enregistrement est créé à la position 1 du fichier, c'est-à-dire au début de celui-ci. Dans les autres cas, les enregistrements sont stockés à la suite des précédents.

Un message de confirmation est affiché après l'enregistrement et les zones de saisie sont initialisées. En cas d'erreur, lors de l'ouverture du fichier ou de l'écriture d'un enregistrement, un message est affiché à l'utilisateur.

Consultez la section Les instructions et fonctions d'accès aux fichiers, L'instruction *Put*, page 355.

4. Ajoutez les instructions nécessaires pour fermer le fichier à la fin de l'exécution :

```
Private Sub Form_Unload(Cancel As Integer)
  If mblnOuvert Then
    Close #1
  End If
End Sub
```

À la fin de l'exécution, le fichier est fermé seulement s'il est ouvert.

5. Testez minutieusement l'enregistrement des données. Entrez des informations pour 5 ou 6 étudiants et assurez-vous que l'exécution se déroule normalement.

6. Afin d'observer le contenu du fichier, démarrez l'application Bloc-Notes et ouvrez le fichier Fichier.dat stocké dans le dossier courant du projet. Comme vous pouvez le constater, le contenu d'un fichier direct est différent de celui d'un fichier séquentiel. Il n'y a pas de saut de ligne ni de caractère de séparation entre les champs et les enregistrements.

8 Création d'un contrôle ActiveX

8.1 INTRODUCTION À LA TECHNOLOGIE ACTIVEX

La technologie ActiveX permet aux programmeurs de créer dans Visual Basic des contrôles personnalisés et de pouvoir les utiliser dans d'autres projets Visual Basic. Les contrôles ActiveX peuvent aussi être utilisés sur des pages Web visualisées par Internet Explorer, dans des applications de bases de données Access, dans des projets Visual C++ et dans la plupart des produits de développement Microsoft.

On retrouve sur le marché un grand choix de contrôles déjà programmés qui généralement suffisent aux besoins des développeurs. Il est cependant très intéressant de découvrir comment un contrôle ActiveX est créé avec ses propres propriétés, méthodes et événements.

L'exercice 28 vous permet de créer manuellement un contrôle personnalisé, de le tester et de l'utiliser dans un projet Visual Basic. Des explications détaillées vous sont fournies sur le code de programmation nécessaire à la conception du contrôle. Il est possible de créer plus rapidement un contrôle personnalisé, en utilisant l'Assistant Interface de contrôle ActiveX qui génère automatiquement le code de programmation. Cependant, le code généré est pratiquement incompréhensible pour une personne qui n'a jamais créé un contrôle ActiveX. Il faut donc créé manuellement un contrôle ActiveX avant d'utiliser l'Assistant Interface de contrôle ActiveX.

Exercice 28 Créer le projet *Contrôle Pointage*

Cet exercice permet de créer le contrôle ActiveX personnalisé **Pointage**. Ce contrôle servira à inscrire la note accordée par un juge, lors de compétitions sportives. Il est formé de 3 contrôles existants : **Label** pour afficher la note, **UpDown** pour sélectionner la note et **Timer** pour faire clignoter la note, dans le cas exceptionnel où l'athlète a obtenu une note parfaite.

Par défaut, les notes varient de 0 à 10. L'utilisateur utilise les flèches à droite pour inscrire la note obtenue. Lorsqu'une note parfaite (10) est inscrite, le contrôle Pointage clignote.

Le contrôle Pointage comporte 3 propriétés, 3 méthodes et il réagit à 2 événements :

3 Propriétés	**ValeurMinimum**	Note la plus petite qui peut être inscrite. (Valeur par défaut = 0)
	ValeurMaximum	Note la plus élevée qui peut être inscrite. (Valeur par défaut = 10)
	Valeur	Note affichée au démarrage. (Valeur par défaut = ValeurMinimum)
3 Méthodes	**Clignoter**	Déclenche le clignotement.
	ArreterClignoter	Interrompt le clignotement.
	Initialiser	Affiche la valeur au démarrage.
2 Événements	**Click**	Événement déclenché lors d'un clic sur la note.
	NoteParfaite	Événement déclenché lorsqu'une note parfaite est attribuée. Une note parfaite est égale à la valeur maximum.

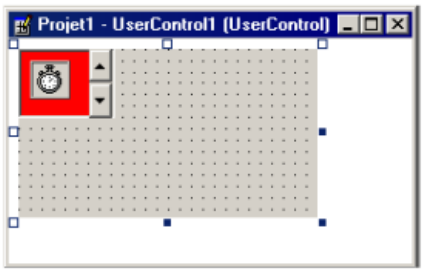
Étape 1 Démarrage et enregistrement

1. À partir de l'Explorateur de Windows, copiez le dossier du volume Chapitre8 /Exercice28 /Debut dans un nouveau dossier .../Visual Basic /Exercice 28 Controle ActiveX.

2. Démarrez Visual Basic. Dans la fenêtre Nouveau projet /Onglet Nouveau, sélectionnez **Contrôle ActiveX** et cliquez sur le bouton Ouvrir.

3. Observez la feuille de conception qui porte le titre *Projet1 – UserControl1*. Nommez le projet **CreationPointage** et donnez le nom **Pointage** à l'objet **UserControl**. Enregistrez **Pointage.ctl** et **CreationPointage.vbp** dans le dossier .../Visual Basic /Exercice 28 Contrôle ActiveX.

Étape 2 Dessiner le contrôle Pointage



1. Ajoutez un contrôle **Label** sur la feuille et fixez les propriétés suivantes :

Name :	lblNombre	Height :	740
Alignment :	2 - Center	BackColor :	Rouge (&H000000FF&)
BackStyle :	1 - Opaque	ForeColor :	Jaune (&H0080FFFF&)
BorderStyle :	1 - Fixed Single	Left :	0
Caption :		Top :	0
Font :	Arial, taille 24, Gras	Width :	650

2. Ajoutez un contrôle **UpDown** dans la boîte à outils par Projet /Composants /Microsoft Windows Common Controls-2 6.0. Dessinez une instance de ce contrôle à droite de l'étiquette. Fixez les propriétés suivantes :

Name :	updNote	Top :	0
AutoBuddy :	True	Width :	240
BuddyControl :	lblNombre	Min :	0
BuddyProperty :	Caption	Max :	10
Height :	720	Increment :	1
Left :	650	SyncBuddy :	True

Le contrôle **UpDown** permet de choisir des nombres de 0 (**Min**) à 10 (**Max**) avec une valeur d'incrémentation de 1 (**Increment**). La propriété **BuddyControl** lie le contrôle UpDown à la propriété Caption (**BuddyProperty**) du contrôle lblNombre. La propriété **SyncBuddy** assure la synchronisation entre les deux contrôles.

Consultez la section 4.9, Les contrôles *FlatScrollBar*, *Slider* et *UpDown*.

3. Ajoutez un contrôle **Timer**. Nommez-le tmrTemps et affectez la valeur 0 à sa propriété **Interval**.

> Lorsque la valeur de l'intervalle est égale à 0, l'événement Timer n'est pas déclenché. Cette valeur sera modifiée par programmation afin de déclencher l'événement Timer au moment désiré.

4. Fixez les propriétés suivantes au contrôle **Pointage** :

Height :	740
Width :	1150
BorderStyle :	1 - Fixed Single
ToolBoxBitmap :	fichier Pointage.bmp stocké dans le dossier du projet

> La propriété **ToolBoxBitmap** définit l'image graphique, qui représente le contrôle dans la boîte à outils des projets qui utiliseront ce contrôle.

5. Pour plus de sécurité, verrouillez les contrôles et enregistrez les fichiers .ctl et .vbp.

Étape 3 Définir les propriétés

1. Déclarez les variables qui contiennent les valeurs des propriétés :

```
Dim mintValeurMin As Integer
Dim mintValeurMax As Integer
Dim mintValeur As Integer
```

> Une variable est requise pour stocker la valeur d'un propriété. Dans le cas présent, les valeurs possibles étant entières, les variables sont déclarées avec le type Integer et ont une portée module.

2. Créez la propriété **ValeurMinimum** :

```
Public Property Get ValeurMinimum() As Integer
  ValeurMinimum = mintValeurMin
End Property

Public Property Let ValeurMinimum(ByVal NouvelleValeur As Integer)
  If NouvelleValeur < 0 Or NouvelleValeur > 5 Then
    MsgBox "La valeur minimum doit être comprise entre 0 et 5.", _
              vbCritical, "Valeur de la propriété invalide"
  ElseIf NouvelleValeur > mintValeurMax Then
    MsgBox "La valeur minimum doit être plus petite que la " & _
           "valeur maximum.", vbCritical, _
           "Valeur de la propriété invalide"
  Else
    mintValeurMin = NouvelleValeur
    updNote.Min = NouvelleValeur
    mintValeur = NouvelleValeur
```

```
    PropertyChanged "ValeurMinimum"
    PropertyChanged "Valeur"
  End If
End Property
```

La propriété **ValeurMinimum** est définie en lecture par une procédure **Property Get** et en écriture par une procédure **Property Let**. Cette propriété est ainsi modifiable par l'utilisateur. En l'absence d'une procédure **Property Let**, la propriété n'apparaîtrait pas dans la fenêtre des propriétés et celle-ci serait une *propriété définie en lecture seule*.

Une procédure **Property Get** permet au programme qui utilise le contrôle Pointage d'obtenir la valeur de la propriété :

Exemple : `intValeur = Pointage.ValeurMinimum` (lecture d'une propriété)

Une procédure **Property Let** permet au programme qui utilise le contrôle Pointage de modifier la valeur de la propriété, soit en mode création dans la fenêtre des propriétés ou soit en mode programmation lors de l'exécution.

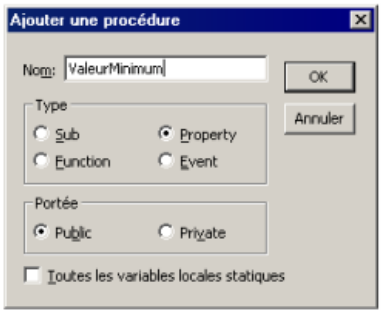
Exemple : `Pointage.ValeurMaximum = 7` (écriture d'une propriété)

Lorsque la propriété ValeurMinimum est modifiée, sa nouvelle valeur est d'abord validée et est ensuite affectée à la propriété **Min** du contrôle UpDown et à la propriété **Valeur** du contrôle Pointage. Ainsi, lorsque le contrôle Pointage sera ajouté sur une feuille de conception, ses propriétés ValeurMinimum et Valeur seront égales, mais cependant modifiables par la suite.

L'instruction **PropertyChanged**, qui s'utilise seulement dans une procédure Property Let, est nécessaire pour synchroniser la valeur affichée dans la fenêtre des propriétés avec la nouvelle valeur de la propriété.

Les procédures **Property Get** et **Property Let** doivent être *publiques* pour être affichées dans la fenêtre des propriétés du projet qui utilise le contrôle Pointage.

Notez qu'une procédure Property peut aussi être ajoutée à partir du menu Outils /Ajouter une procédure :



Une paire de procédures Get/Let sont alors ajoutées dans le module de code et les propriétés ainsi définies sont de type Variant :

```
Option Explicit

Public Property Get ValeurMinimum() As Variant

End Property

Public Property Let ValeurMinimum(ByVal vNewValue As Variant)

End Property
```

3. Créez la propriété **ValeurMaximum** :

```
Public Property Get ValeurMaximum() As Integer
  ValeurMaximum = mintValeurMax
End Property

Public Property Let ValeurMaximum(ByVal NouvelleValeur As Integer)
  If NouvelleValeur <= mintValeurMin Then
    MsgBox "La valeur maximum doit être plus grande que la " & _
           "valeur minimum.", vbCritical, _
           "Valeur de la propriété invalide"
  ElseIf NouvelleValeur > 10 Then
    MsgBox "La valeur maximum ne doit pas excéder 10.", _
            vbCritical, "Valeur de la propriété invalide"
  Else
    mintValeurMax = NouvelleValeur
    updNote.Max = NouvelleValeur
```

```
    PropertyChanged "ValeurMaximum"
  End If
End Property
```

La propriété **ValeurMaximum** est définie en lecture et en écriture par les procédures Property Get et Property Let.

Lorsque la propriété ValeurMaximum est modifiée, sa nouvelle valeur est validée et affectée à la propriété **Max** du contrôle UpDown.

L'instruction PropertyChanged permet que la nouvelle valeur soit affichée dans la fenêtre des propriétés du projet qui utilise le contrôle Pointage.

4. Créez la propriété **Valeur** :

```
Public Property Get Valeur() As Integer
  Valeur = mintValeur
End Property

Public Property Let Valeur(ByVal NouvelleValeur As Integer)
  If NouvelleValeur < mintValeurMin Or _
     NouvelleValeur >= mintValeurMax Then
     MsgBox "La note doit être plus grande ou égale à la valeur " _
            & vbNewLine & _
            "minimum et plus petite que la valeur maximum.", _
            vbCritical, "Valeur de la propriété invalide"
  Else
     mintValeur = NouvelleValeur
     updNote.Value = NouvelleValeur

     PropertyChanged "Valeur"
  End If
End Property
```

La propriété **Valeur** est définie en lecture et en écriture par les procédures Property Get et Property Let.

Lorsque la propriété Valeur est modifiée, sa nouvelle valeur est validée et affectée à la propriété **Value** du contrôle UpDown.

Étape 4 Coder les procédures événementielles de l'objet UserControl

1. Ajoutez le code de la procédure **UserControl_InitProperties** :

```
Private Sub UserControl_InitProperties()
  mintValeurMin = 0
  mintValeurMax = 10
  mintValeur = mintValeurMin
  lblNombre.Caption = mintValeur
End Sub
```

L'événement **InitProperties** se produit lorsqu'une *première* instance du contrôle Pointage est créée sur une feuille. La valeur par défaut est alors affectée à chaque propriété du contrôle Pointage et la valeur 0 est affectée à l'étiquette lblNombre dans laquelle elle sera affichée.

2. Ajoutez le code de la procédure **UserControl_Paint** :

```
Private Sub UserControl_Paint()
  updNote.Value = mintValeur
  updNote.Min = mintValeurMin
  updNote.Max = mintValeurMax
End Sub
```

L'événement **Paint** se produit lorsque le contrôle Pointage est dessiné sur une feuille. Les valeurs des propriétés du contrôle Pointage sont alors affectées aux propriétés correspondantes du contrôle UpDown (Value, Min et Max).

3. Ajoutez le code de la procédure **UserControl_Resize** :

```
Private Sub UserControl_Resize()
  Height = 740
  Width = 1150
End Sub
```

L'événement **Resize** se produit lorsque la taille du contrôle Pointage est redimensionnée. En spécifiant une hauteur et une largeur fixes, le programmeur ne peut modifier la taille du contrôle.

4. Ajoutez le code de la procédure **UserControl_WriteProperties** :

```
Private Sub UserControl_WriteProperties(PropBag As PropertyBag)
  PropBag.WriteProperty "ValeurMinimum", mintValeurMin, 0
  PropBag.WriteProperty "ValeurMaximum", mintValeurMax, 10
  PropBag.WriteProperty "Valeur", mintValeur, mintValeurMin
End Sub
```

L'événement **WriteProperties** se produit lorsqu'une propriété est modifiée ou lorsqu'une instance du contrôle Pointage est détruite.

La méthode **WriteProperty** de l'objet **PropertyBag** enregistre dans une copie mémoire du fichier .frm de la feuille, trois informations relatives à une propriété : son nom, sa valeur actuelle et sa valeur par défaut. L'objet PropertyBag sert à stocker des informations devant être enregistrées et par la suite récupérées.

Par exemple, un premier contrôle Pointage est dessiné sur une feuille de conception. Les valeurs des propriétés affichées sont les suivantes : ValeurMinimum = 0, Valeur = 0 et ValeurMaximum = 10. Le programmeur modifie ces valeurs comme suit : ValeurMinimum = 2, Valeur = 5 et ValeurMaximum = 8. Ces nouvelles valeurs sont enregistrées lors de l'événement WriteProperties et elles seront récupérées, lors de l'événement ReadProperties, lorsqu'un deuxième contrôle sera ajouté sur la même feuille de conception. Ainsi, les valeurs des propriétés du second contrôle Pointage seront 2, 5 et 8.

5. Ajoutez le code de la procédure **UserControl_ReadProperties** :

```
Private Sub UserControl_ReadProperties(PropBag As PropertyBag)
  mintValeurMin = PropBag.ReadProperty("ValeurMinimum", 0)
  mintValeurMax = PropBag.ReadProperty("ValeurMaximum", 10)
  mintValeur = PropBag.ReadProperty("Valeur", mintValeurMin)
End Sub
```

L'événement **ReadProperties** se produit lorsqu'une *deuxième* instance du contrôle Pointage est créée sur une feuille et pour chacune des créations suivantes.

La méthode **ReadProperty** de l'objet PropertyBag permet de récupérer les valeurs de la première instance qui ont été stockées dans une copie mémoire du fichier .frm lors du dernier événement WriteProperties. Le premier argument est le nom de la propriété et le deuxième est la valeur par défaut de la propriété. Cette valeur est utilisée seulement si la valeur enregistrée par la méthode WriteProperty est irrécupérable.

Étape 5 Définir les méthodes

1. Créez la méthode **Clignoter** :

```
Dim mblnClignote As Boolean    'à déclarer au début du module

Public Sub Clignoter()
 tmrTemps.Interval = 300
 mblnClignote = True
End Sub
```

Une méthode est définie par une procédure **Sub** *publique*. La procédure contient les instructions à exécuter lorsque la méthode est employée dans le code de programmation d'un module.

En fixant la valeur 300 à la propriété **Interval** du contrôle **Timer**, la procédure tmrTemps_Timer est exécutée environ 3 fois par seconde, lorsque la méthode **Clignoter** est invoquée.

La variable mblnClignote de portée module est utilisée pour vérifier si le contrôle clignote ou non.

```
Private Sub tmrTemps_Timer()
  If mblnClignote = True Then
    With lblNombre
      If .ForeColor = vbYellow Then
        .BackColor = vbYellow
        .ForeColor = vbRed
      Else
        .BackColor = vbRed
        .ForeColor = vbYellow
      End If
    End With
  End If
End Sub
```

Cette procédure produit un clignotement en intervertissant les couleurs du texte et de l'arrière-plan dans l'étiquette qui affiche la note.

2. Créez la méthode **ArreterClignoter** :

```
Public Sub ArreterClignoter()
  tmrTemps.Interval = 0
  mblnClignote = False
  lblNombre.BackColor = vbRed
  lblNombre.ForeColor = vbYellow
End Sub
```

En affectant la valeur 0 à la propriété **Interval** du contrôle Timer, l'événement Timer n'est plus déclenché et le clignotement est ainsi interrompu. Les couleurs sont initialisées à leurs valeurs originales.

3. Créez la méthode **Initialiser** :

```
Dim mintValeurMinInit As Integer
Dim mintValeurMaxInit As Integer
Dim mintValeurInit As Integer

Private Sub UserControl_Paint()
  .
  .
  mintValeurMinInit = mintValeurMin
```

```
  mintValeurMaxInit = mintValeurMax
  mintValeurInit = mintValeur
End Sub

Public Sub Initialiser()
  mintValeurMin = mintValeurMinInit
  mintValeurMax = mintValeurMaxInit
  mintValeur = mintValeurInit
  updNote.Value = mintValeur
End Sub
```

Des variables de portée module sont nécessaires afin de récupérer les valeurs originales des propriétés du contrôle Pointage lorsque la méthode **Initialiser** est invoquée. Lors de l'événement Paint, les valeurs sont conservées et lorsque la méthode **Initialiser** est utilisée, les valeurs originales sont affectées aux propriétés.

Étape 6 Déclarer les événements

1. Déclarez les événements **Click** et **NoteParfaite** dans la partie Déclarations du module :

```
Option Explicit

Public Event Click()
Public Event NoteParfaite()
```

L'instruction **Event** permet de déclarer un événement propre à un contrôle utilisateur. Un événement peut aussi comporter des arguments qui doivent être cités à l'intérieur des parenthèses. Pour obtenir des détails sur l'utilisation d'arguments, consultez la rubrique Event, instruction dans l'index de l'Aide MSDN.

2. Créez l'événement **Click** :

```
Private Sub lblNombre_Click()
  RaiseEvent Click
End Sub
```

Lorsqu'un clic est effectué sur l'étiquette lblNombre, l'instruction **RaiseEvent** déclenche l'événement **Click** du contrôle Pointage.

3. Créez l'événement **NoteParfaite** :

```
Private Sub updNote_Change()
  mintValeur = updNote.Value
  If mblnClignote = True Then
      Call ArreterClignoter
```

```
  ElseIf mintValeur = mintValeurMax Then
    RaiseEvent NoteParfaite
  End If
End Sub
```

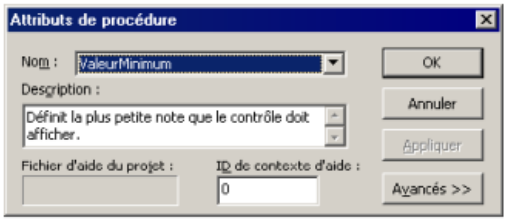
Lorsque l'utilisateur clique sur l'une des flèches du contrôle Pointage, la valeur affichée est modifiée et l'événement **Change** est déclenché. À ce moment, la nouvelle valeur est conservée dans la variable mintValeur.

Si le contrôle Pointage clignote, la procédure **ArreterClignoter** est appelée afin d'interrompre le clignotement puisque la note a été changée. Par exemple, si la note affichée est 10 et que le contrôle clignote, en sélectionnant la valeur 9, le clignotement doit cesser.

Lorsque le contrôle ne clignote pas et que la nouvelle valeur est égale à la valeur maximum, l'événement **NoteParfaite** est déclenché par l'instruction **RaiseEvent**.

Étape 7 Compléter la définition des propriétés

1. Placez le pointeur d'insertion dans la procédure Property Get ValeurMinimum() et à partir du menu, sélectionnez Outils /Attributs de procédure.
2. Dans la zone **Description**, entrez la description suivante : Définit la plus petite note que le contrôle doit afficher.



Le texte inscrit dans la zone **Description** est affiché au bas de la fenêtre des propriétés lorsque cette propriété est sélectionnée.

3. Sélectionnez la propriété **ValeurMaximum** dans la liste **Nom** et ajoutez la description suivante : Définit la note la plus élevée que le contrôle doit afficher.
4. Sélectionnez la propriété **Valeur** dans la liste **Nom** et ajoutez la description suivante : Définit la note affichée au démarrage.
5. Cliquez sur le bouton **Avancés**. Dans la liste **ID de la procédure**, sélectionnez le choix **(Par défaut)**.

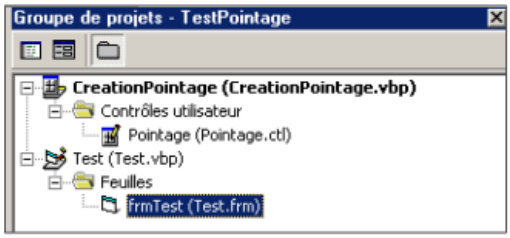
> La propriété Valeur est définie comme *propriété par défaut* du contrôle Pointage, tout comme la propriété Text qui est la propriété par défaut d'un contrôle TextBox.

6. Fermez la fenêtre **Attributs de procédure** en cliquant sur le bouton OK.
7. Enregistrez tous les fichiers du projet en cliquant sur la disquette de la barre d'outils Standard.
8. Fermez la fenêtre de conception *CréationPointage – Pointage (UserControl)* et la fenêtre de code *CréationPointage – Pointage (Code)*. Observez la boîte à outils, le nouveau contrôle **Pointage** devrait s'y trouver.

Étape 8 Tester le contrôle* Pointage *dans un groupe de projets (.vbg)

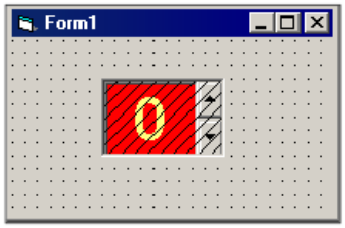
La meilleure façon de tester un contrôle personnalisé est d'utiliser le nouveau contrôle dans un petit projet de type EXE Standard. Pour se faire, un groupe de projets doit être créé afin de pouvoir simultanément tester le contrôle dans un projet de type EXE Standard et de le modifier dans un autre projet de type Contrôle ActiveX.

1. À partir du menu, sélectionnez Fichier /Ajouter un projet /Onglet nouveau /EXE Standard /Ouvrir.
2. Nommez le projet **Test** et donnez le nom **frmTest** à la feuille.
3. Sélectionnez **frmTest** dans la fenêtre **Groupe de projets** et enregistrez la feuille sous le nom **Test.frm**.
4. Sélectionnez **Test** dans la fenêtre **Groupe de projets** et enregistrez le projet sous le nom **Test.vbp**.
5. À partir du menu, sélectionnez Fichier /Enregistrer le groupe de projets. Enregistrez le groupe de projets sous le nom **TestPointage.vbg**. Le contenu de votre fenêtre **Groupe de projets** devrait être semblable à la figure suivante :



6. Afin de pouvoir utiliser le contrôle Pointage dans un type de projet EXE Standard, celui-ci doit être *public*. Sélectionnez le contrôle Pointage dans la fenêtre **Groupe de projets**, cliquez sur l'icône **Afficher l'objet** dans cette même fenêtre et affectez la valeur **True** à la propriété **Public**.

7. Au besoin, fermez la fenêtre de conception CréationPointage – Pointage (UserControl) et la fenêtre de code CréationPointage – Pointage (Code).

> Dans les prochaines étapes de l'exercice, si des erreurs sont détectées dans la conception du contrôle Pointage, pour modifier ou tester le contrôle Pointage, vous devez basculer d'un type de projet à l'autre (EXE Standard à Contrôle ActiveX et vice-versa), en sélectionnant le type de projet désiré dans la fenêtre **Groupe de projets**.
>
> Pour pouvoir tester un contrôle utilisateur dans un projet EXE Standard, les fenêtres **UserControl** et **Code** de celui-ci *doivent être fermées*. Visual Basic utilise deux façons de nous indiquer qu'un contrôle n'est plus utilisable suite à des modifications :
>
> - L'icône du contrôle est estompée dans la boîte à outils ou
> - Le contrôle est hachuré comme ceci :
>
> 
>
>
> Lorsque le contrôle est hachuré, supprimez-le et dessinez-en un nouveau avant de poursuivre les tests. Cette nouvelle instance comportera les dernières modifications qui ont été effectuées au contrôle Pointage.

8. Ajoutez un contrôle **Pointage** sur la feuille.

9. Dans la fenêtre des propriétés, observez les propriétés personnalisées **Valeur**, **ValeurMinimum** et **ValeurMaximum** ainsi que la description de chacune des propriétés qui est affichée dans le bas de la fenêtre des propriétés.

10. Avant d'exécuter, cliquez avec le bouton droit de la souris sur le projet **Test** dans la fenêtre **Groupe de projets** et sélectionnez **Projet de démarrage**.

Le projet Test doit être défini comme **projet de démarrage** pour tester le contrôle Pointage. Le projet de démarrage est celui qui est exécuté lorsque le choix **Exécuter** est sélectionné.

Notez que si le projet de démarrage est le projet de type Contrôle ActiveX, l'application Internet Explorer démarre. Le contrôle personnalisé peut ainsi être observé sommairement. Cependant, il ne peut être testé minutieusement.

11. Exécutez le projet Test et cliquez à plusieurs reprises sur les flèches. Assurez-vous que les nombres défilent de 0 à 10 avec un intervalle de 1.

12. Dans la fenêtre des propriétés, modifiez les valeurs des propriétés afin de tester les différentes validations. Observez attentivement les messages d'erreur qui sont affichés.

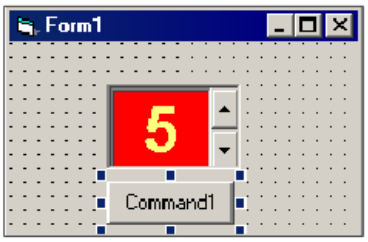
    - Entrez une valeur minimum plus grande que 5.
    - Entrez une valeur minimum plus grande ou égale à la valeur maximum.
    - Entrez une valeur maximum plus petite ou égale à la valeur minimum.
    - Entrez une valeur maximum plus grande que 10.
    - Entrez une valeur plus petite que la valeur minimum.
    - Entrez une valeur plus grande ou égale à la valeur maximum.

13. Dans la fenêtres des propriétés, modifiez les valeurs des propriétés comme suit :

    - ValeurMinimum = 1 ;
    - ValeurMaximum = 8 ;
    - Valeur = 5.

14. Exécutez et assurez-vous que la valeur minimum est 1, la valeur maximum est 8 et que la valeur 5 est affichée au démarrage.

15. Ajoutez un bouton de commande sur la feuille :



16. Ajoutez le code de programmation qui suit dans la fenêtre de code de la feuille frmTest. Lors de la saisie du code, vous pourrez remarquer que les propriétés, méthodes et événements du contrôle Pointage sont affichées dans les listes déroulantes lorsqu'un point est entré à la suite du nom de l'objet Pointage1.

```
Private Sub Command1_Click()
  Pointage1.Initialiser
End Sub

Private Sub Pointage1_Click()
  Pointage1.ArreterClignoter
End Sub

Private Sub Pointage1_NoteParfaite()
  Dim intRep As Integer
  intRep = MsgBox("Confirmation d'une note parfaite", vbYesNo)
  If intRep = vbYes Then
    Pointage1.Clignoter
  Else
    Pointage1.Initialiser
  End If
End Sub
```

17. Exécutez et effectuez les tests suivants :

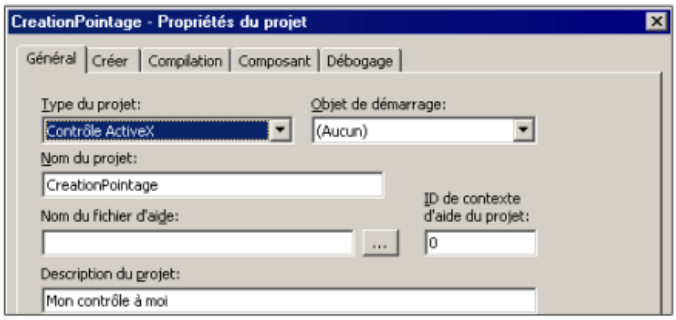
- Sélectionnez une note parfaite et répondez Oui au message. La note devrait clignoter puisque la méthode Clignoter est invoquée lorsque l'événement NoteParfaite est déclenché et que l'utilisateur répond Oui au message.
- Cliquez sur la note qui clignote, le clignotement devrait cesser puisque la méthode ArreterClignoter est invoquée lors d'un clic sur le contrôle Pointage1.
- Cliquez sur le bouton de commande, la note devrait être initialisée puisque la méthode Initialiser est invoquée lors d'un clic sur ce bouton.
- Sélectionnez une note parfaite et répondez Non au message. La note devrait être initialisée puisque la méthode Initialiser est invoquée lorsque l'événement NoteParfaite est déclenché et que l'utilisateur répond Non au message.

18. Enregistrez tous les fichiers du groupe de projets en cliquant sur la disquette de la barre d'outils Standard.

Étape 9 Créer le fichier Pointage.ocx et ajouter le composant Pointage

1. Sélectionnez le projet CreationPointage.vbp dans la fenêtre **Groupe de projets**. Cliquez avec le bouton droit de la souris et sélectionnez **Propriétés de**

CreationPointage... Dans la zone de texte **Description du projet**, entrez une brève description du contrôle comme par exemple : Mon contrôle à moi.



> La description du projet identifie le composant dans la fenêtre **Composants**, accessible par Projet /Composants.

2. Fermez la fenêtre **Propriétés du projet** en cliquant sur le bouton OK.
3. Sélectionnez **Pointage** dans la fenêtre **Groupe de projets** et à partir du menu, sélectionnez Fichier /Créer CréationPointage.ocx. Assurez-vous que le chemin d'accès est bien celui du projet, modifiez le nom pour **Pointage.ocx** et pressez le bouton OK.
4. Vérifiez si le contrôle Pointage est bien ajouté aux autres composants en sélectionnant au menu Projet /Composants. Dans la fenêtre **Composants**, recherchez l'élément **Mon contrôle à moi** qui devrait être présent. Toutefois, s'il n'est pas présent, des consignes seront fournies à l'étape 10 pour l'ajouter.
5. Enregistrez tous les fichiers du groupe de projets en cliquant sur la disquette de la barre d'outils Standard.

***Étape 10 Utiliser le contrôle Pointage dans le projet* Compétition**

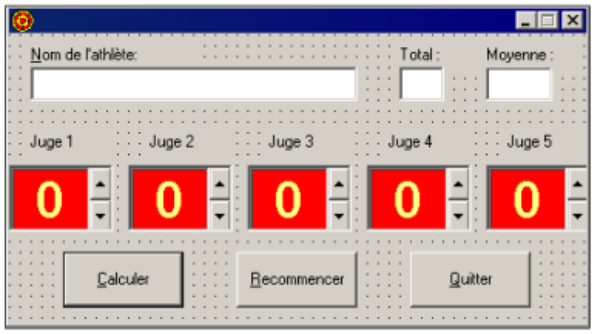
1. Ouvrez le projet **Competition.vbp** stocké dans le dossier .../Visual Basic /Exercice 28 Controle ActiveX.
2. Ajoutez le contrôle **Pointage** dans la boîte à outils en accédant à la fenêtre **Composants**. Sélectionnez l'élément **Mon contrôle à moi** et fermez la fenêtre.

> Si le contrôle ne se trouve pas dans la fenêtre, sélectionnez le bouton **Parcourir**, retrouvez le fichier Pointage.ocx et cliquez sur le bouton **Ouvrir**.

3. Ajoutez un contrôle **Pointage** sous l'étiquette **Juge 1** et donnez-lui le nom **ptgJuge**. Affectez la valeur **False** à la propriété **TabStop**.

   La touche Tab n'est pas utile sur le contrôle Pointage puisque la note doit être inscrite en cliquant sur les flèches.

4. Créez le groupe de contrôles ptgJuge(1) à ptgJuge(5).



5. Codez les instructions nécessaires pour inscrire le texte et la date du jour dans la barre de titre de la feuille.

   Exemple

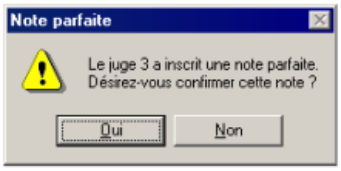
6. Programmez le bouton **Quitter** qui procède au déchargement de la feuille. Utilisez l'instruction **End** lors de l'événement Form_Unload afin de vous assurer que le contrôle Timer ne demeure pas fonctionnel après le déchargement. Testez la fin de l'exécution lorsque le contrôle Pointage clignote.

7. Programmez le bouton **Calculer**. Celui-ci doit calculer et afficher le *total des points* ainsi que la *moyenne des points*. La moyenne doit être affichée avec une seule décimale. Testez le bouton **Calculer**.

8. Programmez et testez le bouton **Recommencer** qui doit accomplir les tâches suivantes :

   - Initialiser les 5 pointages par la méthode Initialiser ;
   - Effacer le nom, le total et la moyenne ;
   - Donner le focus à la zone de texte du nom de l'athlète.

9. Codez l'événement NoteParfaite du contrôle ptgJuge :
   - Lorsqu'une note parfaite est inscrite, affichez le message qui suit. Notez que le *numéro du juge* est précisé dans le message.



   - Si l'utilisateur répond Oui, la note parfaite doit clignoter. Si l'utilisateur répond Non, la note doit être initialisée.
   - Testez l'événement NoteParfaite.
10. Lorsque l'utilisateur clique sur une note qui clignote, cessez le clignotement. Testez ce changement.
11. Testez le projet dans son ensemble et enregistrez tous les fichiers.

8.2 L'ASSISTANT INTERFACE DE CONTRÔLE ACTIVEX VB 6

L'Assistant Interface de contrôle ActiveX VB 6 permet de créer rapidement *des propriétés, des méthodes et des événements* propres à un nouveau contrôle utilisateur.

Pour accéder à cet assistant, sélectionnez au menu Compléments /Gestionnaire de compléments. Dans la fenêtre **Gestionnaire de compléments**, sélectionnez **Assistant Interface de contrôle ActiveX VB 6** et cochez la case **Charger/Décharger** afin d'ajouter le choix **Assistant Interface de contrôles ActiveX** au menu Compléments.

Pour démarrer l'Assistant, un projet de type Contrôle ActiveX doit obligatoirement être inclus dans le projet courant. Ainsi, créez d'abord l'interface graphique de votre contrôle personnalisé.

Une fois lancé, l'Assistant affiche successivement une série de fenêtres dans lesquelles vous devez sélectionner des options et, au besoin, entrer certaines spécifications. Par la suite, le code de programmation est généré automatiquement. Le code, ainsi généré, ressemble à celui du module qui a été créé manuellement lors de la conception du contrôle Pointage. Cependant, le code étant généré *sans normes de programmation*, celui-ci est difficilement déchiffrable.

Annexe A *Tableaux des principales constantes Visual Basic (Extrait de MSDN)*

A.1 Les constantes des codes de touches

vbKeyBack	RET.ARR (BackSpace)	**vbKeyCancel**	ANNULER (Ctrl+C)
vbKeyTab	TAB	**vbKeyEnd**	FIN (End)
vbKeyClear	EFFACER (Ctrl+L)	**vbKeyHome**	ORIGINE
vbKeyReturn	ENTRÉE (Enter)	**vbKeyLeft**	FLÈCHE GAUCHE ←
vbKeyShift	MAJ (Shift)	**vbKeyUp**	FLÈCHE HAUT ↑
vbKeyControl	CTRL	**vbKeyRight**	FLÈCHE DROITE →
vbKeyMenu	MENU	**vbKeyDown**	FLÈCHE BAS ↓
vbKeyPause	PAUSE	**vbKeySelect**	SÉLECTION
vbKeyCapital	FIX MAJ (Caps Lock)	**vbKeyPrint**	IMP.ÉCRAN (PrintScreen)
vbKeyEscape	ÉCHAP (Escape)	**vbKeyExecute**	EXÉCUTER (+)
vbKeySpace	ESPACE (SpaceBar)	**vbKeySnapshot**	SNAPSHOT(,)
vbKeyPageUp	PG.PRÉC (PageUp)	**vbKeyInsert**	INSER (Ins)
vbKeyPageDown	PG.SUIV (PageDown)	**vbKeyDelete**	SUPPR (Delete)
vbKeyNumlock	VERR.NUM (NumLock)	**vbKeyHelp**	AIDE (/)
vbKeyA à vbKeyZ	A à Z (majuscules)	**vbKeyLButton**	Souris : bouton gauche
vbKey0 à vbKey9	0 à 9	**vbKeyRButton**	Souris : bouton droit
vbKeyF1 à vbKeyF16	F1 à F16	**vbKeyMButton**	Souris : bouton centre

A.2 Les constantes des touches du clavier numérique

vbKeyNumpad0 à vbKeyNumpad9	0 à 9	**vbKeySubtract**	–
vbKeyMultiply	*	**vbKeyDecimal**	.
vbKeyAdd	+	**vbKeyDivide**	/
vbKeySeparator	ENTRÉE (Enter)		

A.3 Les constantes couleurs

vbBlack	Noir	**vbGreen**	Vert	**vbBlue**	Bleu	**vbCyan**	Cyan
vbRed	Rouge	**vbYellow**	Jaune	**vbMagenta**	Magenta	**vbWhite**	Blanc

A.4 Les Constantes couleurs système

vb3DDKShadow	Couleur de l'ombre la plus sombre des éléments 3D
vb3Dhighlight	Couleur de surligneur des éléments 3D
vb3Dlight	Couleur de l'ombre intermédiaire des éléments 3D
vbActiveBorder	Couleur de la bordure de la fenêtre active
vbActiveTitleBar	Couleur de la barre de titre de la fenêtre active
vbApplicationWorkspace	Couleur d'arrière-plan des applications MDI
vbButtonFace	Couleur d'ombrage de la surface des boutons de commande
vbButtonShadow	Couleur d'ombrage du bord des boutons de commande
vbButtonText	Couleur du texte des boutons d'option
vbDesktop	Couleur du bureau
vbGrayText	Texte grisé (désactivé)
vbHighlight	Couleur d'arrière-plan d'une sélection dans un contrôle
vbHighlightText	Couleur du texte des éléments sélectionnés dans un contrôle
vbInactiveBorder	Couleur de la bordure de la fenêtre inactive
vbInactiveCaptionText	Couleur de texte d'une légende inactive
vbInactiveTitleBar	Couleur de la barre de titre de la fenêtre inactive
vbInfoBackground	Couleur d'arrière-plan des info-bulles
vbInfoText	Couleur du texte dans les info-bulles
vbMenuBar	Couleur d'arrière-plan du menu
vbMenuText	Couleur du texte des menus
vbScrollBars	Couleur de la barre de défilement
vbTitleBarText	Couleur du texte de la barre de titre
vbWindowBackground	Couleur d'arrière-plan de la fenêtre
vbWindowFrame	Couleur du cadre de la fenêtre
vbWindowText	Couleur du texte de la fenêtre

A.5 Autres Constantes

vbBack	Chr(8)	Retour arrière (BackSpace)
vbCr	Chr(13)	Retour de chariot (Enter, Carriage Return)
vbCrLf	Chr(13) + Chr(10)	Retour chariot et de saut de ligne
vbLf	Chr(10)	Saut de ligne (Line Feed)
vbNewLine	Chr(13) + Chr(10)	Saut de ligne
vbNullChar	Chr(0)	Valeur ANSI = 0
vbNullString	""	Chaîne de longueur nulle ("")
vbTab	Chr(9)	Tabulation (touche Tab)

BIBLIOGRAPHIE

Microsoft Visual Basic 6, Micro Application PC Poche, 1999.

Visual Basic 6.0 cours et exercices, Diane Zak, Éditions Reynald Goulet Inc., 1999.

Microsoft Visual Basic 6, Micro Application Grand Livre, 1999.

Votre premier programme en Visual Basic, Rémy Lentzner, Éditions Eyrolles, 2000.

Guide du programmeur Microsoft Visual Basic 6.0, Microsoft Corporation, Microsoft Press, 1999.

INDEX

.bas 231, 232
.cls 231, 232
.ctl 232, 406
.ctx 232
.exe 14, 47-48, 232
.frm 7, 47, 231, 232
.frx 232
.log 232
.ocx 32, 232, 421-422
.vbg 46, 233, 417
.vbp 7, 46, 232
.vbw 232
: (séparateur d'instructions) 242
_ (continuité) 12, 241
' (apostrophe) 12, 242
& 19, 80, 166, 192-193, 255

A

Accumulateur 126
ActiveControl 396
Activate 151
ActiveX, *technologie* 405
Add 228
AddItem 98, 176
Affectation 251
Affichage, *menu* 52-54
Afficher l'instruction suivante 60
Aide, *menu* 66-67
Aide contextuelle 68
Ajouter
  des compléments 65
  un contrôle 33
  un contrôle utilisateur 55
  un espion 59
  un fichier 55
  un module 55, 112
  un module de classe 55
  un projet 46
  une feuille 55, 111
  une feuille MDI 55
  une fonction personnalisée 135
  une procédure 60, 101, 133
Ajuster à la grille 57
Ajuster la taille 35
Aligner des contrôles 56
Alignment 84, 190
AllowPrompt 222
Alt, *touche* 19, 192
Ancrable 40
Ancrage 63
Animations 151-152
Annuler une opération 48
ANSI, *codes* 101, 206, 207, 287, 428-429
App 156, 227
Appearance 191
Argument 133, 239-241
Array 137, 283
Arrêt, *mode* 57
Asc 101, 207, 287
Assistant contrôle ActiveX 405, 423
Attributs 20
Attributs de procédure 61, 417
AutoSize 167, 179

B

BackColor 10, 166, 191-192
BackStyle 16, 167
Barre de menus 28, 45-68, 214
Barre d'état, créer une 337-352
Barre d'outils 54
  créer une 337-352
  Débogage 52, 53, 57-60
  Éditeur de code de feuille 56, 57
  Édition 50-52, 54
  Standard 28, 38, 45-49, 57, 58, 61
Barres de défilement 181-183, 224
Bascule, programmer une 85
Basculer le point d'arrêt 59
BASIC 1
Beep 79, 274
Bibliothèque MSDN 66
Boîte à outils 30-32, 52
Boîtes de dialogue
  Enregistrer sous 312, 314-315
  Gestion du bouton Annuler 316
  Imprimer 312-314
  Ouvrir 312, 314-315
  Police 312, 313-314
  prédéfinies 255-261

standard de Windows 312-231
Boolean 243
BorderColor 180
BorderStyle 10, 159, 167, 168, 177, 179, 180
BorderWidth 88, 180
Boucle d'événements 200, 202
Boucles 137-141, 267-273
Do Loop 139, 140, 270-271
For Each 272-273, 395
For Next 125, 126, 268-270
While Wend 138, 272
Bouton
Commenter le bloc 52
Ne pas commenter le bloc 52
Bouton d'annulation 18, 166
Bouton de commande 10, 165-166
Bouton d'option 71, 170-171
Bouton par défaut 17, 166
Button 338, 340, 345
ButtonClick 340, 345
ByRef 239
Byte 243
ByVal 239

C

Cadre 36, 71, 176
Calculer
des taxes 99
un escompte 99
un paiement 106
Call 20, 102, 235, 236
Cancel, *argument* 80, 162
Cancel, *propriété* 18, 166
CancelError 316
Caption 9, 71, 192-193
Caractère de concaténation 80
Caractères de types de données 249
Case à cocher 72, 171-172
Case, Select 87
CausesValidation 212
CCur 106, 254, 286
CDate 287, 401
Centrer sur la feuille 57
Chaînes de caractères 244, 248
Change 102, 201-202, 310
Charger un fichier image 152
CheckBox 72, 171-172
Chemin d'accès du projet 227
Choose 303
Chr 207, 287
CInt 89, 90, 254, 286
Classe d'objet 21, 22
Clear 98, 124, 176, 228
Click 11, 199-200
Clignoter un contrôle 144
ClipBoard 123, 124, 228
Close 357
Cls 163, 391
Code de programmation 231
Codes de touches du clavier 425
Codes d'erreur 94
Collection 158
Columns 173
ComboBox 70, 172-176
CommandButton 10, 165-166
Commentaires 12, 86, 242
Commenter le bloc 52, 86
CommonDialog 312-313
Compilation 43
Compléments, *menu* 64-65
Compléter un mot 12, 51
Composants 20, 31, 32, 56
Compteur 127
Concaténation 80, 255
Concepteurs 56
Condition 261
Const 250
Constantes 249-251
autres 427
codes de touches du clavier 425
couleurs 425-426
définies par le programmeur 250
énumération de 250-251
intégrées de VB 22, 23, 249, 425-427
symboliques 99, 249-250
Conteneur 36, 170, 176
Continuité de ligne 12, 241
ControlBox 159

Contrôle de données 186
Contrôle utilisateur, *créer un* 405-424
Contrôles ActiveX 20, 31, 32, 56
  ADO Data 186
  Assistant Interface 405, 423
  CommonDialog 312-313
  création de 405-423
  DateTimePicker 77
  FlatScrollBar 224
  ImageList 337-338
  MaskEdBox 221-223, 392
  MCI (Media Control Interface) 152
  RichTextBox 332-337
  Slider 224-225
  SSTab 36
  StatusBar 341-343
  TabStrip 36
  ToolBar 338-341
  UpDown 225-226, 407
Contrôles intégrés 10, 20, 31
  CheckBox 72, 171-172
  ComboBox 70, 172-176
  CommandButton 10, 165-166
  Data 186
  DirListBox 183-185, 308
  DriveListBox 183-185, 308
  FileListBox 183-185, 308
  Frame 36, 71, 176-177
  HScrollBar 181-183
  Image 16, 178-179
  Label 11, 167
  Line 88, 180-181
  ListBox 71, 172-176
  Menu 215
  OLE 185, 186
  OptionButton 71, 170-171
  PictureBox 152, 178-179
  Shape 85, 180-181
  TextBox 70, 168-169
  Timer 77, 144, 177-178
  VScrollBar 181-183
Controls, *collection* 158, 159, 228, 395
Convention d'affectation des noms aux objets 24-27
Conversion numérique 254-255
Copier un contrôle 188
Couleurs
  créer une couleur 53
  palette de 53
Count 159
Couper/Copier/Coller, *programmation* 122, 228
Créateur de menus 61, 117, 215-219
Créer le fichier .exe 14, 47
Créer un gestionnaire de clavier 211-212, 395-397
CSng 91, 254, 286
Currency 243
Curseur, *contrôle* 224-225

D

Data 186
Date 97, 292, 401
Date/Time 243, 252
Dates, fonctions de 292-295
DblClick 200-201
Débogage
  exemple 90, 92
  menu 58-60
Debug 210, 360
Déclaration
  des variables 246-248
  obligatoire des variables 2, 61
Déclarations, *section* 100, 231, 247, 250
Default 17, 166
Définir l'instruction suivante 60
Définition, *menu Affichage* 52
Démarrer Visual Basic
Dépassement de capacité 92, 128
Déplacer
  des contrôles 35
  des objets 198
Dernier emplacement 52
Dim 78, 247
DirListBox 183-185, 308

DisabledPicture 196-197
Division par zéro 128
Do Loop 139, 140 270-271
DoEvents 307
Double 243
DownPicture 196-197
Drag 199
DragDrop 147, 204
DragIcon 204
DragMode 145, 204
DragOver 204
DrawMode 180
Drive 184, 310
DriveListBox 183-185, 308

E

Échange d'objets 186
Écran d'accueil 155
Éditeur RTF 332
Éditeur, paramètres 61, 62
Édition avec masque 221-223
Édition, *menu* 48-52
Effacer
  tous les points d'arrêt 60
  une zone de texte 73
Empty 248
Enabled 70, 177, 193
End 80, 162, 241
Enregistrer
  un projet 8, 46-47
  une feuille 8, 46-47
Entrer une ligne de code rapidement 51
Enum 154, 250-251
Énumération de constantes 154, 250-251
EOF 358-359
Err 94, 277-279, 380
Erreurs
  codes 278
  de compilation 43
  de syntaxe 12, 42
  d'exécution 90, 92, 94, 128, 150, 274-279
Error, On 93-94, 274-276
Escompte, calculer un 99
Espace de pile insuffisant 202
Espacement horizontal/vertical 57
Espion 53, 59
Espion express 59
Étiquette 11, 167
Événements 21, 22
  Activate 151, 161
  ButtonClick 340, 345
  Change 102, 201-202, 310
  clavier 206-212
  Click 11, 199-200
  créer un événement 416-417
  DblClick 200-201
  Deactivate 161
  DragDrop 147, 204
  DragOver 204
  d'une feuille 161-163
  GotFocus 203
  Initialize 161
  KeyDown 208-210
  KeyPress 101, 112, 113, 206-208
  KeyUp 208-210
  Load 75, 161
  LostFocus 129, 203
  MouseDown 203
  MouseMove 13, 203
  MouseUp 153, 203
  Paint 161
  QueryUnload 79, 162
  RaiseEvent 415
  Resize 161
  Scroll 206
  SelChange 333, 348
  Terminate 162
  Timer 77
  Unload 75, 162
  Validate 129, 212-213
  ValidationError 222
Event 415
Exécuter 29, 57
  avec compilation complète 58
  de nouveau 58
  jusqu'au curseur 58
Exécution, *menu* 57-58
Exit Sub 94, 234, 235
Explorateur de projets 7, 28-30
Explorateur d'objets 22-23, 52
Expression mathématique 253-255

F

Fenêtre 20
  de code 40-44, 52, 66
  de conception de feuille 28, 33-37
  Espions 53

Exécution 52
Explorateur de projets 28-30, 52
Explorateur d'objets 22-23, 52
Objet 52
Présentation des feuilles 28, 37-38, 52
principale 28-29
Propriétés 28, 38-40, 52
Variables locales 53
Fenêtre, *menu* 65-66
Feuille 20
Fichier, *menu* 45-48
Fichiers de données 353-364
binaires 355
directs 354-355
instructions et fonctions 355-364
séquentiels 354
structure 353
Fichiers VB, types de 232-233
FileListBox 183-185, 308
FileName 184, 315
FillColor 180
FillStyle 180
Filter 315, 321, 322
Fin, *menu Exécution* 58
Flags 313-315, 319-321
FlatScrollBar 224
Focus 197
Fonction personnalisée 135
Fonctions
Array 137, 283
Asc 101, 207, 287
CCur 106, 287
CDate 287, 401
Choose 303
Chr 207, 287
CInt 89, 90, 287
CSng 91, 287
Date 97, 292, 401
DoEvents 307
EOF 358-359
Format 296-302
FormatCurrency 93, 295
FormatDateTime 78, 97, 296
FormatNumber 93, 127, 296
FormatPercent 295
FreeFile 356
IIf 302
Input 361
InputBox 97, 259-261
InStr 113, 288
InStrRev 288
Int 143, 289
IsDate 287, 291, 405
IsEmpty 290
IsNull 291, 335
IsNumeric 92, 290
LBound 283
LCase 287
Left 287
Len 123, 287
LoadPicture 152, 303, 311
LOF 361
LTrim 287
Mid 287
MsgBox 12, 244-259
Now 294
Pmt 106, 290
QBColor 146, 305
Replace 288
RGB 144, 306
Right 287
Rnd 143, 304
Round 127, 289
RTrim 287
Shell 307
Space 288
Spc 359, 369
StrConv 288
String 288
StrReverse 288
Switch 302
Tab 359,369
Time 78, 294
Trim 287
TypeName 292
UBound 283
UCase 267, 287
Val 89-91
VarType 291-292
Fonctions regroupées 286-307
de chaînes de caractères 287-289
de conversion 287
de types de données 286-287
de dates et heures 292-295
de formatage 295-302
de sélection 302-303
financières 290
mathématiques 289-290
Font 10, 191-192
For Each 272-273, 395

For Next 125,126, 268-270
ForeColor 11, 191-192
Form 20, 33, 36, 158-164
Form_Load 75, 161
Form_Unload 75, 162
Format, *fonction* 296-302
Format, *menu* 56-57
Formatage des résultats 295-302
FormatCurrency 93, 295
FormatDateTime 78, 97, 296
Formater
  des chaînes de caractères 302
  des dates et des heures 299-301
  des nombres 297-299
FormatNumber 93, 127, 296
FormatPercent 295
Formes graphiques 180-181
Forms, *collection* 158
Frame 36, 71, 176-177
FreeFile 355-356
Function 136, 237

G

Générer aléatoirement
  des couleurs 144
  des nombres 142, 143
Gestion des erreurs d'exécution 274-279
Gestionnaire de clavier 211-212, 395-397
Gestionnaire de compléments 64
Gestionnaire de données 64
Gestionnaire d'erreurs 93-94
Get 364
GetText 123, 228
Glisser/déplacer des objets 147, 199, 204
GotFocus 203
GoTo 94, 274, 276
Grille de points 37
Groupe de constantes 22
Groupe de projets 46
Groupes de contrôles 82, 84, 86, 186-189

H

Height 189
Heures, *fonctions* 292-295
Hide 154, 163
HideSelection 168
HScrollBar 181-183

I

Icon 10, 159
Icônes, chemin d'accès 77
If 13, 19, 79, 261-265
IIf 302
Image 16, 178-179
ImageList 337-338
Imprimer
  des nombres en colonne 371-373
  des rapports 367-377
  un projet 47
  une rubrique d'aide 67
Incorporation d'objets 185
Incrément 225-226
Index de référence MSDN 66
Index, *argument* 84, 87
Index, *propriété* 83
Indice 114, 280
Info Express 51
Info Paramètres 51
Initialisation des variables 248
Initialize 161
Input, *fonction* 361-362
Input, *instruction* 358-359
InputBox 97, 259-261
Insérer un fichier 50
Instance 21
InStr 113, 288
InStrRev 288
Instructions
  Beep 79, 274
  Call 20, 102, 235, 236
  Case, Select 87, 265-267
  Close 357
  Const 250
  Dim 78, 247
  Do Loop 139, 140, 270-271
  End 80, 162, 241
  Enum 154, 250-251
  Event 416
  For Each 272-273, 395
  For Next 125, 126, 268-270
  Get 364
  If 13, 19, 79, 261-265
  Input 358-359
  Kill 391

Line Input 360-361
Load 164, 189
LSet 289, 371-373
On Error 93-94, 274-276
Open 355-357
Option Base 1 282
Option Explicit 2, 247
Print 359-360
Private 247, 250, 251
PropertyChanged 409
Public 247, 250, 251
Put 362-363
Randomize 143, 304
ReDim 282
Resume 276-277
RSet 289, 371-373
Select Case 87, 265-267
SendKeys 210-211, 398
Set 229-230
Static 247
Stop 241
Sub 234, 235
Type 284-285, 402
Unload 17, 162, 189, 241
While Wend 138, 272
With 98, 273
Write 357-358
Int 143, 289
Integer 243
Intercepter des touches 206
Interface graphique utilisateur 20, 33, 158
Interpréteur de VB 42
Interval 77
Invite d'enregistrement des modifications 2, 61
IsDate 287, 291, 401
IsEmpty 290
IsNull 291, 335
IsNumeric 92, 290

K

Key 337, 340, 342, 345
KeyAscii 101, 112, 113
KeyDown 208-210
KeyPress 101, 112, 206-208
KeyPreview 159, 206, 396
KeyUp 208-210
Kill 391

L

Label 11, 167
LargeChange 182
Largeur de la tabulation 3, 61
LBound 283
LCase 287
Left 86, 160, 189, 287
Len 123, 287
Liaison d'objets 185
Ligne graphique 180-181
Line 88, 180-181
Line Input 360-361
List 70, 174, 184
ListBox 71, 172-176
ListCount 174, 185
Liste
de dossiers 183-185
de fichiers 183-185
de lecteurs 183-185
de sélection des objets 20, 39
d'images 337-338
Objet 41
Procédure 41
ListImage 337
ListIndex 174, 185
Load 164, 189
LoadFile 334, 336
LoadPicture 152, 303, 311
Locked 168
LOF 361-362
Long 243
LostFocus 129, 203
LSet 289, 371-373
LTrim 287

M

Majuscules 25, 42
Mask 221-222
MaskEdBox 221-223, 392
Masques de saisie 221-223, 392
Max 182
MaxButton 69, 160
MaxLength 92, 168
MCI, contrôle multimédia 152
MDI 29, 55
MDIChild 160
Me 159
Menu
Aide (?) 66-67
Affichage 52-54

Compléments 64-65
Débogage 58-60
Édition 48-52
Exécution 57-58
Fenêtre 65-66
Fichier 45-48
Format 56-57
Outils 60-64
Projet 55-56
Menu contextuel, créer un 153, 219-220
Menu système 159
Menu, créer un 117, 214-220
Méthodes 21, 22
Add 228
AddItem 98, 176
Clear 98, 124, 176, 228
Cls 163, 391
Créer une méthode 413-415
Drag 199
GetText 123, 228
Hide 154, 163
LoadFile 334, 336
Move 198, 346
PopupMenu 153, 163, 219-221
Print 163, 210, 360, 369
PrintForm 100, 101, 164
Raise 278-279, 380
Remove 229
RemoveItem 176
SaveFile 334, 336
SelPrint 334, 337
SetFocus 73, 161, 197-198
SetText 123, 228
Show 110, 121, 164
ShowColor 312, 317, 318
ShowFont 312, 318, 319
ShowOpen 312, 322
ShowPrinter 312, 320
ShowSave 312, 321
Mettre en retrait 12, 49-50, 86
Microsoft sur le Web 67
Mid 287
Min 182
MinButton 160
Minuscules 25, 42
Minuterie 177
Mod 25
Modale, feuille 121, 164
Mode
création 20-21, 29, 57, 58
exécution 20-21, 29, 57
pas à pas 58, 60, 126
programmation 40
Modèles de feuilles 63, 155
Modification d'une propriété 39
Modifier un espion 59
Modules 231-232
de classe 231
de code 40, 231
de feuille 231
standard 112, 231
Mot clé 243
MouseDown 203
MouseIcon 142, 194
MouseMove 13, 203
MousePointer 110, 142, 194
MouseUp 153, 203
Move 198, 346
Moveable 160
MSDN 66
MsgBox 12, 255-259
MultiLine 82, 168
MultiSelect 174, 185

N

Name 10, 24-27
Ne pas commenter le bloc 52, 86
NewIndex 175
Nommer
un projet 7
une feuille 7
Noms
des constantes 250
des éléments de VB 242-243
des objets 24-27
des variables 246
Normes de programmation
instruction If 263
noms des objets 25
noms des variables 246
retraits 49-50
Notation scientifique 92
Nouveau projet 45
Now 294
Number 94

O

Object 244
Objet

App 156, 227
Button 338, 340, 345
ClipBoard 123, 124, 228
Debug 210, 360
Err 94
ListImage 337
Panel 341-343
Printer 360, 367-371
Screen 346
UserControl 406, 411-413
OLE 185-186
On Error 93-94, 274-276
Open 355-357
Opérateurs
d'affectation 251
de concaténation 255
logiques 262-263
mathématiques 253-254
relationnels 262
Option Base 1 282
Option Explicit 2, 247
Optional 239
OptionButton 71, 170-171
Ordre de tabulation 72, 195
Orienté objet 20
Outils, *menu* 60-64
Ouvrir un projet 45
erreur d'ouverture 47

P

Pages de propriétés 53
Paint 161
Palette de couleurs 53
Panel 341-343
ParamArray 239
Paramètres
de configuration 2-4
d'environnement 63, 64
généraux de l'éditeur 62
régionaux de Windows 4-6
Pas à pas, *mode* 58
PasswordChar 168
Path 183, 185, 227, 310, 311
Pattern 185, 312
Permuter deux valeurs 148
Personnalisé 39
Personnaliser le pointeur de la souris 142, 194
Picture 16, 109, 160, 179, 196
PictureBox 152, 178-179
Pile des appels 53
Plan 57
Pmt 106, 290
Poignées de dimension 34
Point d'arrêt 44, 59, 60
Pointeur de sélection 31
PopupMenu 153, 163, 219-221
Portée
des constantes 250
des procédures 238-239
des variables 245-246
Préfixes des noms d'objets 26-27
Présentation des feuilles 37-38
Presse-papiers 123, 124, 228
Print, *instruction* 359-360
Print, *méthode* 163, 210, 360, 369
Printer 360, 367-371
PrintForm 100, 101, 163, 164
Private 239, 247, 250, 251
Procédure 231, 234-241
événementielle 21, 231, 234
Function 237
personnalisée 101, 133, 235
Programmation d'une bascule 85
Programmation orientée objet (OO) 20
Projet de démarrage 418-419
Projet, *menu* 55-56
PromptChar 222
PromptInclude 223, 385
Property Get 408-409
Property Let 408-409
PropertyChanged 409
Propriété par défaut 89, 107, 165
Propriétés 20, 22
Alignment 84, 190
AllowPrompt 222
Appearance 191
AutoSize 167, 179
BackColor 10, 166, 191-192
BackStyle 16, 167
BorderColor 180

BorderStyle 10, 159, 167, 177, 179
BorderWidth 88, 180
Cancel 18, 166
CancelError 316
Caption 9, 71, 192-193
CausesValidation 212-213
Columns 173
ControlBox 159
Controls 159
Count 159
Créer une propriété 408
Default 17, 166
DisabledPicture 196-197
DownPicture 196-197
DragIcon 204
DragMode 145, 204
DrawMode 180
Drive 184, 310
d'un projet 56
Enabled 70, 177, 178, 193
Fenêtre des 38-40
FileName 184, 315
FillColor 180
FillStyle 180
Filter 315, 321, 322
Flags 313, 315, 319-321
Font 10, 191-192
ForeColor 11, 191-192
Height 189
HideSelection 168
Icon 10, 159
Index 83, 186-189
Interval 77, 178
Key 337, 340, 342, 345
KeyPreview 159, 206, 396
LargeChange 182
Left 86, 160, 189
List 70, 174, 184
ListCount 174, 185
ListIndex 174, 185
Locked 168
Mask 221-222
Max 182
MaxButton 69, 160
MaxLength 92, 168
MDIChild 160
Min 182
MinButton 160
Modification des 39
MouseIcon 142, 194
MousePointer 110, 142, 194
Moveable 160
Multiline 82, 168
MultiSelect 174, 185
Name 10, 24-27
NewIndex 175
Number 94
PasswordChar 168
Path 183, 185, 227, 310, 311
Pattern 185, 312
Personnalisé 39
Picture 16, 109, 160, 179, 196
PromptChar 222
PromptInclude 223, 385
ReadOnly 185
ScaleMode 160
ScrollBars 82, 169
Selected 175, 185
SelLength 123, 169
SelStart 123, 169
SelText 122, 169
Shape 85, 180
ShowInTaskBar 160
SmallChange 182
Sorted 71, 175
StartUpPosition 10, 38, 160
Stretch 179
Style 16, 98, 171, 172, 175
TabIndex 72, 195
TabStop 195
Tag 195-196, 394
Text 70, 169, 175
ToolTipText 77, 196
Top 11, 160, 189
TopIndex 176
Value 71, 166, 171, 172, 182
Visible 16, 177, 194
Width 189
WindowState 160
WordWrap 167
Public 112, 238, 247, 250, 251
Put 362-363

Q

QBColor 146, 305
QueryUnload 79

R

Raccourci clavier 19, 166, 192-193

Sub Main 233, 373
Raise 278-279, 380
RaiseEvent 415-416
Randomize 143, 304
Rapports, produire des 367-377
ReadOnly 185
Rechercher /remplacer 48-49
Rechercher MSDN 67
ReDim 282
Refaire une opération 48
Références 55
Règles de syntaxe
des noms des objets 25
du langage VB 42, 241-243
Regroupement des contrôles 36-37
Remove 229
RemoveItem 176
Répertorier
les constantes 50
les propriétés/méthodes 50
Replace 288
Resize 161
Resume 276-277
Resume Next 275
Retraits 49-50, 86
RGB 144, 306
RichTextBox 332-337
Right 287
Rnd 143, 304
Round 127, 289
RSet 289, 371-373
RTrim 287

S

SaveFile 334, 336
ScaleMode 160
Screen 346
Scroll 206
ScrollBars 82, 169
SDI 29
Section Déclarations 100
SelChange 333, 348
Select Case 87, 265-267
Selected 175, 185
Sélection de contrôles 31, 34-35, 48
Sélectionner un texte par programmation 123, 169
SelLength 123, 169
SelPrint 334, 337
SelStart 123, 169
SelText 122, 169
SendKeys 210-211, 398
Séparateur de fenêtre 43-44
Set 229-230
SetFocus 73, 161, 197-198
SetText 123, 228
Shape 85, 180-181
Shell 307
Show 110, 121, 164
ShowColor 312, 317, 318
ShowFont 312, 318, 319
ShowInTaskBar 160
ShowOpen 312, 322
ShowPrinter 312, 320
ShowSave 312, 321
Signet 44, 51
Single 243
Slider 224-225
SmallChange 182
Sommaire MSDN 66
Sons 151-152
Sorted 71, 175
Space 288
Spc 359, 369
SSTab 36
StartUpPosition 10, 38, 160
Static 239, 247
StatusBar 341-343
Stop 241
StrConv 288
Stretch 179
String, *fonction* 288
String, *type de données* 244
StrReverse 288
Structure With 98, 273
Structures de décision 261-267
If 13, 19, 79, 261-265
Select Case 87, 265-267
Structures de données 284
Structures répétitives 267-273
Do Loop 139, 140, 270-271
For Each 272-273, 395
For Next 125, 126, 268-270
While Wend 138, 272
Style 16, 98, 171, 172, 175
Sub 234, 235

Sub Main 233, 373
Support technique 67
Supprimer
  un contrôle 35, 48
  un contrôle par programmation 228
  un élément du projet 55
  un projet 46
  un texte sélectionné 48
  une ligne de code 48
Switch 302
Symbole
  décimal 4, 91, 252, 255
  monétaire 5
Syntaxe
  de la programmation des objets 23
  du langage VB 241-243

T

Tab, *fonction* 359,369
Tab, *touche* 18, 72, 195
TabIndex 72, 195
Tableau des caractères ANSI 428-429
Tableaux en mémoire 113-115, 132, 137, 280-284
TabStop 195
TabStrip 36
Tabulation
  largeur 12
  ordre de 72, 195
Tag 195-196, 394
Taille des contrôles 35
Taxes, calculer des 99
Terminate 162
Text 70, 169, 175
TextBox 70, 168-169
Texte sélectionné 169
Time 78, 294
Timer 77, 144, 177-178
ToolBar 338-341
ToolTipText 77, 196
Top 11, 160, 189
TopIndex 176
Touches du clavier, codes 425
Transmission des arguments 133, 134, 239-241
Trim 287
Twip 190
Type, *instruction* 284-285, 402
Type défini par l'utilisateur 244
Type incompatible 90, 91
TypeName 292
Types de données personnalisés 284-285
Types de données, tableau 243-244
Types de fichiers d'un projet 243-244

U

UBound 283
Ucase 267, 287
Uniformiser la taille 57
Unload 17, 162, 189, 241
Until 270, 271
UpDown 225-226, 407
UserControl 406, 411-414

V

Validate 129, 212-213
Validation
  alphabétique 208
  numérique 207
ValidationError 222
Value 71, 166, 171, 172, 182
Variables
  chaînes de caractères 244, 248
  déclaration 246-248
  définition 243
  initialisation 248
  noms 246
  numériques 95, 244, 248
  portée globale 112, 245
  portée locale 80, 245
  portée module 78, 245
  publiques 112, 245
Variant 244
VarType 291-292
VBA 2
vbCenter 191
vbChecked 172
vbGrayed 172
vbKeyBack 113, 208
vbKeyEscape 119, 208
vbKeyReturn 112, 211
vbKeySpace 112, 208
vbLeftJustify 191
vbNewLine 94, 256
vbRightJustify 191
VBScript 2
vbUnChecked 172
Verrouiller les contrôles 57, 118

Visdata 64
Visible 16, 177, 194
VScrollBar 181-183

W

Web, Microsoft sur le 67
While 270, 271
While Wend 272
Width 189
WindowState 160
With 98, 273
WordWrap 167
Write 357-358

Z

Zone d'image 178
Zone de liste 71, 172-176
Zone de liste modifiable 70, 172-176
Zone de texte 70, 168-169