VISUAL BASIC 6

Jean-Louis Boimond

1 INTRODUCTION

Visual Basic s'est développé à partir du langage BASIC (Beginner's All-purpose Symbolic Instruction Code, Larousse : " Langage de programmation conçu pour l'utilisation interactive de terminaux ou de micro-ordinateurs ", 1960). Le but du langage BASIC était d'aider les gens à apprendre la programmation. Son utilisation très répandue conduisit à de nombreuses améliorations. Avec le développement (1980-1990) de l'interface utilisateur graphique (*Graphical User Interface - GUI*) de Windows Microsoft, BASIC a naturellement évolué pour donner lieu à Visual Basic (1991). Depuis, plusieurs versions ont été proposées, Visual Basic 6 est apparue en 1998.

Visual Basic est un langage de programmation existant actuellement en trois versions (*Learning, Professional, Entreprise*). Les programmes (aussi appelées *applications*) sont créés dans un environnement de développement intégré (*Integrated Development Environment - IDE*), ceci dans le but de créer, exécuter et déboguer les programmes d'une manière efficace. Ce langage est réputé pour permettre un développement rapide d'applications. Outre une interface utilisateur graphique, il dispose de caractéristiques telles que la manipulation d'événements, un accès à Win32 API, la programmation orientée objet, la gestion d'erreurs, la programmation structurée. C'est un langage interprété, notons que les éditions *Professional* et *Entreprise* permettent une compilation en *code natif* (*code machine*) (voir annexe A pour plus de détails).

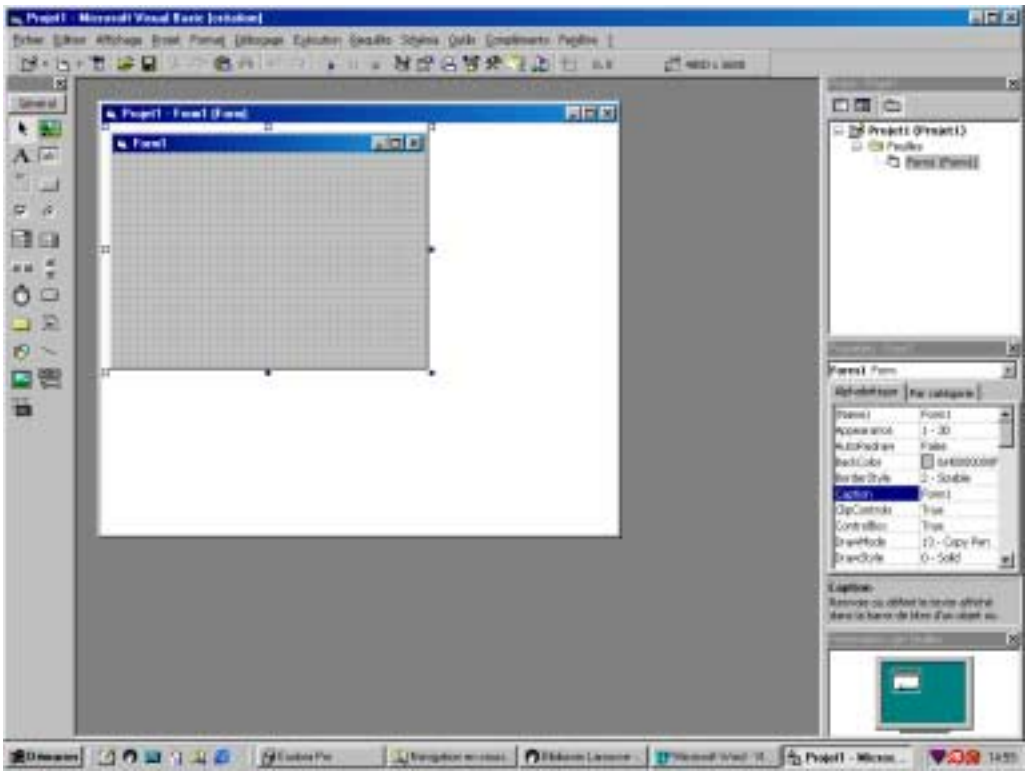
2 ENVIRONNEMENT DE DÉVELOPPEMENT INTÉGRÉ (EDI)

L'environnement de développement intégré de Visual Basic permet de créer, exécuter et déboguer des programmes Windows dans une seule application (à savoir Visual Basic).

2.1 Présentation de l'Environnement de Développement Intégré (EDI)

Au démarrage de Visual Basic, la boîte de dialogue suivante, intitulée *Nouveau projet* (cf. barre de titre), s'affiche. Elle permet de choisir le type de projet que l'on souhaite créer.

Double-cliquer sur l'option *EXE Standard* (surlignée par défaut) de l'onglet *Nouveau* afin de créer un projet (ensemble de fichiers permettant de créer une application Windows). Certaines options de cette boîte de dialogue sont brièvement décrites en annexe B. L'interface de l'EDI de Visual Basic s'affiche (voir la recopie d'écran dans la figure suivante).

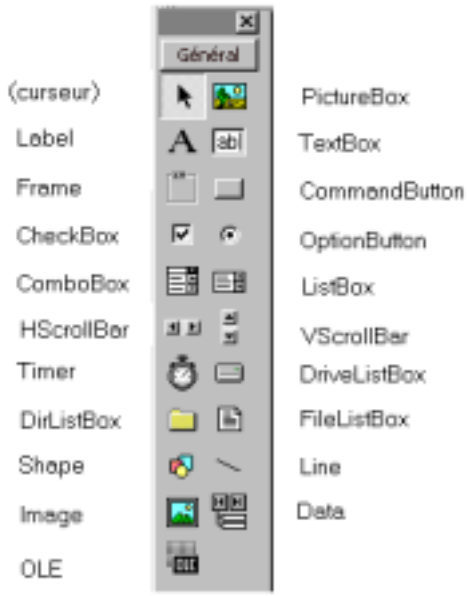


Cet environnement, situé dans une fenêtre intitulée *Projet1 - Microsoft Visual Basic [création]* (cf. barre de titre), est formé d'une *barre de menu*, d'une *barre de contrôles* et de plusieurs fenêtres.

- Le tableau suivant récapitule les menus accessibles à partir de la *barre de menu*, située en haut de l'écran. En dessous sont situés, dans une barre d'outils, les raccourcis graphiques (icônes) représentant les commandes les plus courantes.

Menu	Description
Fichier	Contient des options pour ouvrir, fermer, écrire, … des projets.
Edition	Contient des options telles que couper, copier, coller, etc.
Affichage	Contient des options concernant l'EDI.
Projet	Contient des options pour ajouter au projet des particularités telles que des feuilles.
Format	Contient des options pour aligner et verrouiller les contrôles d'une feuille.
Débogage	Contient des options pour déboguer le programme.
Exécution	Contient des options pour exécuter, stopper, … un programme.
Requête	Contient des options pour manipuler des données récupérées d'une base de données.
Schéma	Contient des options pour éditer des bases de données.
Outils	Contient des options pour particulariser l'EDI.
Compléments	Contient des options pour utiliser, installer et désinstaller des compléments (programmes visant à augmenter la performance de Visual Basic).
Fenêtres	Contient des options pour disposer les fenêtres à l'écran.
Aide	Contient des options pour obtenir de l'aide.

- La *barre de contrôles*, située dans la partie gauche de l'écran, contient des contrôles permettant de réaliser l'Interface Utilisateur Graphique (IUG) (i.e., la partie de la feuille visible par l'utilisateur). Ces derniers permettent une programmation accélérée. La figure et le tableau suivants résument les contrôles contenus dans cette barre.



Contrôle	Description
Curseur	Permet d'interagir avec les contrôles de la feuille (en fait, ce n'est pas un contrôle).
PictureBox (zone d'image)	Permet l'affichage d'un fichier image (bmp, ico, wmf, …).
Label (étiquette)	Permet l'affichage d'un texte non modifiable directement par l'utilisateur.
TextBox (zone de saisie)	Permet à l'utilisateur d'entrer du texte.
Frame (cadre)	Permet de regrouper d'autres contrôles tels que des cases à option.
CommandButton (bouton)	Ce contrôle est représenté par un bouton que l'utilisateur peut presser, ou cliquer, pour exécuter une action sous-jacente.
CheckBox (case à cocher)	Permet de fournir un bouton de choix (*checked* ou *unchecked*).
OptionButton (case à option)	Les cases à option sont utilisées en groupe (dans le même cadre), sachant que seulement une case peut être sélectionnée (*True*) à la fois.
ListBox (liste)	Permet l'affichage de différents items (éléments).
ComboBox (liste combinée)	Permet de combiner les fonctionnalités des zones de saisie et des listes.
HScrollBar	Une barre de défilement horizontale.
VScrollBar	Une barre de défilement verticale.
Timer (horloge)	Permet la répétitivité de tâches (ce contrôle est non visible pour l'utilisateur).
DriveListBox	Permet un accès aux différents disques du système.
DirListBox	Permet un accès à des répertoires.
FileListBox	Permet d'accéder aux fichiers d'un répertoire.
Shape	Permet de dessiner des cercles, des rectangles, des carrés ou des ellipses.
Line	Permet de dessiner des lignes.
Image (dessin)	Similaire au contrôle PictureBox avec des capacités moindres.
Data	Permet la connexion à une base de données.
OLE	Permet d'interagir avec d'autres applications Windows.

- L'EDI d'un projet *EXE Standard* contient les fenêtre suivantes :
  - *Projet1 – Form1 (Form)*
  - *Présentation des feuilles*

- *Propriétés – Form1*
- *Projet – Projet1*

- La fenêtre *Projet1 – Form1 (Form)* contient une feuille (en anglais *form*) vierge nommée, par défaut, *Form1*, dans laquelle le programme de l'IUG sera conçu. L'IUG est la partie visible du programme (boutons, cases à cocher, cases à option, zones de saisie, etc.), elle permet :
  - à un utilisateur de fournir des données (appelées *entrées*) au programme,
  - au programme de restituer des résultats (appelés *sorties*) à l'utilisateur.
- La fenêtre *Présentation des feuilles* (en anglais *Form Layout*) permet de spécifier - à l'aide de la souris - la position souhaitée de la feuille (dans le cas présent *Form1*) sur l'écran lors de l'exécution du programme.
- La fenêtre *Propriétés – Form1* décrit les propriétés (taille, position, couleur, etc.) d'une feuille ou d'un contrôle. Dans le cas présent, sont listées les propriétés liées à la feuille *Form1*. Chaque type de contrôle a son propre ensemble de propriétés. Certaines propriétés, telles que *Height* et *Width* sont communes aux feuilles et aux contrôles, alors que d'autres propriétés sont uniques à une feuille, ou un contrôle.
- La fenêtre *Projet – Projet1* est l'explorateur de projets, elle regroupe par type les différents fichiers composant le projet en cours. La barre de menu de cette fenêtre comporte 3 boutons : *Code* afin de visualiser la fenêtre contenant le code du programme, *Afficher l'objet* pour faire apparaître la feuille, *Basculer les dossiers* cache, ou montre, le dossier *Feuilles*. Par défaut, le projet a pour nom *Projet1* et est constitué d'une seule feuille, nommée *Form1*.

2.2 Un premier exemple : Affichage d'une ligne de texte

Réalisons un programme qui écrit le texte " Premier exemple " à l'écran. Ce programme, de part sa simplicité, se fait sans écrire une seule ligne de code. En fait, nous allons utiliser les techniques de la programmation visuelle dans laquelle, à travers différentes manipulations (tel qu'un clic de souris), vont être fournies les informations suffisantes à Visual Basic pour qu'il puisse automatiquement générer le code de notre programme.

Voici la démarche. Nous allons ajouter un contrôle *Label* sur l'IUG d'une feuille : Le fait de double-cliquer sur ce contrôle fait apparaître un contrôle *Label*, nommé de manière standard *Label1*, au centre de l'IUG de la feuille. L'utilisation de la souris permet de le positionner et de le dimensionner à volonté (la grille, visible en mode *création*, permet un alignement des contrôles, cf. onglet *Général* du menu *Outils* | *Options*).
La propriété *Caption* du contrôle *Label1*, visible dans la fenêtre des propriétés (fenêtre *Propriétés – Label1*), détermine le texte affiché par ce contrôle. Il suffit donc de remplacer le texte " Label1 ", mis par défaut, par celui souhaité (à savoir " Premier exemple ").

Remarques :
- La propriété *Caption* ne doit pas être confondue avec la propriété *Name* du contrôle, bien que leurs contenus soient par défaut les mêmes !
- Le fait de préfixer le *Name* de chaque *Label* avec les lettres (minuscules) *lbl* permet une meilleure identification des contrôles *Label*. Il en sera de même pour les autres contrôles. Cette convention, largement adoptée, permet une meilleure " lisibilité " du programme.

Attention : Avant de tenter une exécution, il est prudent de sauvegarder le projet (cf. menu *Fichier* | *Enregistrer le projet sous …*). En ce qui concerne notre exemple, la sauvegarde du projet, nommé *Projet1* par défaut, donne lieu à la création de deux fichiers (texte) :

- l'un, avec l'extension *.frm*, pour mémoriser les propriétés de la feuille, nommée *Form1* par défaut. En fait, à chaque élément du projet (feuille ou module) va correspondre un fichier (*.frm* ou *.bas*).
- l'autre, avec l'extension *.vbp*, pour mémoriser les paramètres des variables d'environnement et des noms de fichiers des éléments constituant le projet.

Jusqu'à présent, nous avons travaillé dans le mode *création* de l'EDI (i.e., le programme n'est pas exécuté). En plaçant l'EDI en mode *exécution* (obtenu en cliquant le bouton *Exécuter*, ou en activant le menu *Exécution | Exécuter*), le programme est exécuté (avec une possible interaction *via* le programme de l'IUG). Il s'ensuit alors que :

- la fenêtre de l'EDI est intitulée *Projet1 - Microsoft Visual Basic [exécution]* (cf. barre de titre),
- la plupart des fenêtres utilisables dans le mode *création* sont indisponibles (c'est le cas, par exemple, de la fenêtre *Propriétés*). Notons l'apparition d'une fenêtre nommée *Exécution*, habituellement utilisée pour déboguer le programme.

3 INTRODUCTION A LA PROGRAMMATION DE VISUAL BASIC

La réalisation d'une application s'appuie essentiellement sur l'association de deux éléments : Les objets et les événements.

3.1 La programmation orientée objet

Visual Basic permet le développement orienté objet. Au lieu de résoudre un problème en le décomposant en problèmes plus petits, il s'agit de le scinder sous forme d'objets qui existent chacun indépendamment les uns des autres. Chaque objet possède certaines caractéristiques (appelées *propriétés*) et fonctions qu'il serait en mesure d'effectuer (appelées *méthodes*).
A titre d'illustration, la feuille " *Form1* " est un objet ayant, entre autres, une propriété appelée " *Caption* " qui permet de spécifier le texte affiché dans la barre de titre de la feuille, et deux méthodes appelées " *Show* " et " *Hide* " qui permettent respectivement d'afficher et de cacher la feuille.

Très brièvement, on peut dire que les objets " encapsulent " les données (les attributs) et les méthodes (les comportements), sachant que les données et les méthodes sont intimement liées. Les objets ont la propriété de " cacher l'information ", au sens où la communication d'un objet avec un autre ne nécessite pas de connaître comment ces objets sont " implémentés " (à titre d'illustration, il est possible de conduire une voiture sans connaître en détail son fonctionnement mécanique).

Alors que dans un langage procédural, la fonction joue le rôle de l'unité de programmation (les fonctions en langage objet sont appelées méthodes), l'unité de programmation en langage objet est la *classe* à partir laquelle les objets sont " instanciés " (créés) (tout objet appartient à une classe). En fait, la communication avec un objet se fait notamment à travers ses propriétés et ses méthodes. Une propriété a une valeur que le programmeur peut consulter, ou modifier ; une méthode est une procédure permettant d'agir sur les objets d'une classe. Le programmeur peut faire référence à la propriété *prop*, respectivement à la méthode *meth*, de l'objet *obj* en écrivant **obj.prop**, respectivement **obj.meth**.

De nombreuses classes sont prédéfinies dans Visual Basic. Par exemple, chaque icône de contrôle, situé dans la barre de contrôles de l'EDI, représente en fait une classe. Lorsqu'un contrôle est déplacé vers une feuille, Visual Basic crée automatiquement un objet de cette classe, vers lequel votre programme pourra envoyer des messages ; ces objets pouvant aussi générer des événements. Les versions récentes de Visual Basic permettent le développement de ses propres classes (et contrôles).

3.2 Programmation événementielle

Visual Basic permet la création d'IUG, sous une forme standard dans l'environnement Windows, par simple pointage et cliquage de la souris, ce qui a pour intérêt d'éliminer l'écriture du code permettant de générer l'IUG d'une feuille, de fixer ses propriétés, de créer les contrôles contenus dans l'IUG.
Il s'agit pour le programmeur de créer l'IUG d'une feuille et d'écrire le code décrivant ce qui se produit lorsque l'utilisateur interagit avec l'IUG (clic, double-clic, appuie sur une touche, etc.). Ces actions, appelées événements (en anglais *events*), sont reliées au programme *via* le système d'exploitation de Microsoft Windows (voir l'annexe C pour plus de précisions). En fait, il est possible pour chacun des contrôles contenus dans l'IUG de répondre, de manière appropriée, aux différents événements auxquels sont sensibles les contrôles.
Pour cela, il va correspondre à chacun de ces événements une procédure de réponse (en anglais *event procedure*), dans laquelle le programmeur insérera des lignes de code approprié. Une telle programmation de code en réponse à ces événements est dite programmation événementielle.
Ainsi, ce n'est plus le programmeur mais l'utilisateur qui maîtrise l'ordre d'exécution du programme (le programme ne dicte plus la conduite de l'utilisateur).
Dans un programme Visual Basic, tous les contrôles (zones de texte, boutons de commande, etc.) possèdent un ensemble d'événements prédéfinis auxquels on peut lier du code (pour ne pas réagir à un événement prédéfini, il suffit de ne pas écrire de code correspondant). A titre d'illustration, l'événement appelé " *Load* " de la feuille " *Form1* " est déclenché à chaque premier chargement de la feuille (ici " *Form1* "), juste avant son affichage. Il sert habituellement à fixer les valeurs des propriétés de la feuille, ou à exécuter du code lors de son apparition. Voir l'annexe D pour plus de précisions.

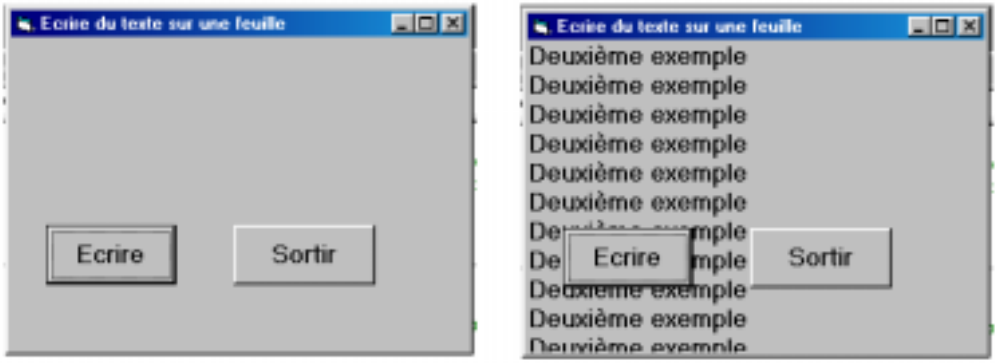
En bref, un programme (une application) Visual Basic est créé à partir d'un projet constitué, notamment, de feuilles, lesquelles comprennent :
- une IUG composée de contrôles,
- les procédures de réponses aux événements associées à ces contrôles.

Notons qu'une feuille peut aussi contenir des procédures et des déclarations locales à la feuille, sans lien direct avec un contrôle contenu dans l'IUG. Nous verrons par la suite, qu'en plus des feuilles, un projet peut être constitué de modules.

3.3 Deux exemples

3.3.1 Écriture d'une ligne de texte sur une feuille

L'IUG consiste en deux boutons : **Ecrire** et **Sortir**. Il s'agit d'écrire la ligne " Deuxième exemple " autant de fois que l'utilisateur clique sur le bouton **Ecrire**. Le fait de cliquer sur le bouton **Sortir** permet de terminer l'exécution du programme. Les recopies d'écran qui suivent montrent, à gauche, la feuille avant un clic sur le bouton **Ecrire**, à droite, la feuille après plusieurs clics sur ce bouton.



Le tableau qui suit liste les objets et certaines de leurs propriétés (seules les propriétés ayant été modifiées sont listées).

Objet	Propriété	Valeur de la propriété	Description
Form	Name	frmExemple2	Identification de la feuille.
	Caption	Ecrire du texte sur une feuille	Spécification du texte affiché dans la barre de titre de la feuille.
	Font	MS Sans Serif Gras 12 pt	Spécification de la police de caractères de la feuille.
CommandButton **Ecrire**	Name	cmdDisplay	Identification du bouton **Ecrire**.
	Caption	Ecrire	Texte qui apparaît sur le bouton.
	Font	MS Sans Serif Gras 12 pt	Spécification de la police de caractères du texte Caption.
	TabIndex	0	Numéro d'ordre de Tab.
CommandButton **Sortir**	Name	cmdExit	Identification du bouton **Sortir**.
	Caption	Sortir	Texte qui apparaît sur le bouton.
	Font	MS Sans Serif Gras 12 pt	Spécification de la police de caractères du texte Caption.
	TabIndex	1	Numéro d'ordre de Tab.

La propriété *TabIndex* permet de contrôler l'ordre dans lequel les contrôles, constituant l'IUG, reçoivent le *focus* (i.e., deviennent actifs). Le contrôle, ayant une valeur de *TabIndex* égale à 0, a le *focus* au départ.

Le code programme qui suit permet :
- d'afficher dans la feuille le texte " Deuxième exemple " avec la police de caractères souhaitée lors d'un clic sur la touche **Ecrire** (cf. la procédure de réponse à l'événement, nommée *cmdDisplay_Click*).
- de terminer le programme lors d'un clic sur la touche **Sortir** (cf. la procédure de réponse à l'événement, nommée *cmdExit_Click*).

```
Private Sub cmdDisplay_Click()
    ' A chaque fois que le bouton "Ecrire" est cliqué,
    ' le message "Deuxième exemple" est affiché sur la feuille
    Print "Deuxième exemple"
End Sub

Private Sub cmdExit_Click()
    End         ' Termine l'exécution du programme
End Sub
```

Private Sub et *End Sub* marquent respectivement le début et la fin d'une procédure. Le code que le programmeur veut exécuter lors d'un clic sur la touche **Ecrire**, est placé entre le début et la fin de la procédure, i.e., *Private Sub cmdDisplay_Click()* et *End Sub*.

```
    ' A chaque fois que le bouton "Ecrire" est cliqué,
    ' le message "Deuxième exemple" est affiché sur la feuille
```

sont des lignes de commentaires.

La ligne

Print "Deuxième exemple"

affiche le texte " Deuxième exemple " sur la feuille (en cours) en utilisant la méthode *Print*. En toute rigueur, il faudrait faire précéder la méthode *Print* du nom de la feuille, soit la ligne

frmExemple2.Print "Deuxième exemple" (nom de l'objet.nom de la propriété)

Notons que cette méthode n'est pas la plus appropriée lorsque la feuille contient des contrôles, il serait préférable d'écrire le texte dans un contrôle de type *TextBox* (détaillé au § 8.2).

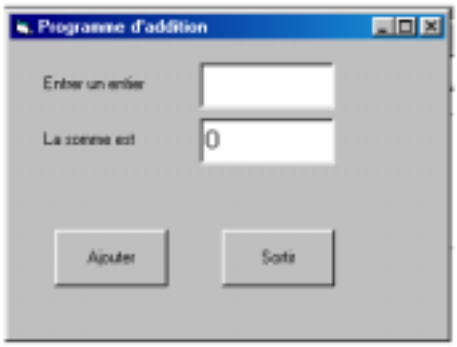
La ligne

End

située dans la procédure *cmdExit*, termine l'exécution du programme (i.e., place l'EDI en mode création).

3.3.2 Addition d'entiers

A partir d'entiers introduits par l'utilisateur, il s'agit d'afficher leur somme successive. L'IUG consiste en deux étiquettes (*Label*), deux zones de saisie (*TextBox*) et deux boutons (*CommandButton*), cf. la recopie d'écran dans la figure suivante.



Le tableau qui suit liste les objets et certaines de leurs propriétés (seules les propriétés ayant été modifiées sont listées).

Objet	Propriété	Valeur de la propriété	Description
Form	Name	frmExemple3	Identification de la feuille.
	Caption	Programme d'addition	Spécification du texte affiché dans la barre de titre de la feuille.
CommandButton **Ajouter**	Name	cmdAjouter	Identification du bouton **Ajouter**.
	Caption	Ajouter	Texte qui apparaît sur le bouton.
CommandButton **Sortir**	Name	cmdSortir	Identification du bouton **Sortir**.

	Caption	Sortir	Texte qui apparaît sur le bouton.
Label	Name	lbl1	Identification du label.
	Caption	Entrer un entier	Texte qui apparaît dans le label.
Label	Name	lbl2	Identification du label.
	Caption	La somme est	Texte qui apparaît dans le label.
TextBox	Name	txtEntree	Identification du TextBox.
	Font	MS Sans Serif Gras 12 pt	Spécification de la police de caractères.
	MaxLength	5	Limite le nombre maximum de caractères (la valeur 0, mise par défaut, indique qu'il n'y a pas de limite).
	TabIndex	0	Numéro d'ordre de Tab.
	Text	(vide)	Texte affiché.
TextBox	Name	txtSomme	Identification du TextBox.
	Font	MS Sans Serif Gras 12 pt	Spécification de la police de caractères.
	Text	0	Texte affiché.
	Enabled	False	Autorisation / non autorisation.

Le contrôle *TextBox* permet à l'utilisateur, *via* la propriété *Text*, d'entrer du texte (cf. *TextBox txtEntree*), il permet aussi d'afficher du texte (cf. *TextBox txtSomme*).
La propriété *Enabled* du contrôle *TextBox txtSomme* positionnée à *False* fait que ce contrôle ne réagit à aucun événement, il s'ensuit que le texte représentant la somme est de couleur grise.

Le code du programme est le suivant :

```
Dim sum As Integer

Private Sub cmdAjouter_Click()
    sum = sum + txtEntree.Text
    txtEntree.Text = ""
    txtSomme.Text = sum
End Sub

Private Sub cmdSortir_Click()
    End
End Sub
```

La ligne

```
Dim sum As Integer
```

déclare une variable, nommée *sum*, de type entier (2 octets, entre –32768 et 32767), initialisée par défaut à 0.

Remarque : La déclaration des variables dans Visual Basic est facultative, une variable non déclarée sera par défaut de type *Variant* (cf. § 4.4). Toutefois, il est conseillé, pour des raisons d'aide à la mise au point de programmes, de rendre obligatoire la déclaration des variables. Pour cela, il faut la présence dans la partie " déclaration générale " de la ligne suivante :

Option Explicit

Cette ligne peut être soit tapée directement dans la partie " déclaration générale ", soit introduite automatiquement par Visual Basic en ouvrant, avant toute programmation, le menu *Outils* | *Options*, puis en cliquant l'onglet nommé *Editeur* et en cochant la case *Déclaration des variables obligatoire* (case non cochée par défaut).

La ligne

sum = sum + txtEntree.Text

ajoute le texte de *txtEntree* à la variable *sum*, puis place le résultat dans la variable *sum*. Avant l'opération d'addition (+), la chaîne de caractères située dans la propriété *Text* est convertie implicitement en une valeur entière.
La ligne

txtSomme.Text = sum

met le contenu de la variable *sum* dans *txtSomme.Text*, Visual Basic convertit implicitement la valeur entière contenue dans *sum* en une chaîne de caractères.

Remarque : De manière plus rigoureuse que les conversions implicites (qui peuvent donner lieu à des méprises), il existe des fonctions Visual Basic permettant de convertir une chaîne de caractères en une valeur entière, et réciproquement (cf. fonctions *Val* et *Str$*).

3.4 Règles de priorité des opérateurs arithmétiques

Les opérations arithmétiques sont effectuées selon un ordre fixé par les règles de précédence d'opérations suivantes :

Opération	Ordre d'évaluation
() parenthèses	Evalué en 1er. S'il y a plusieurs paires de parenthèses " de même niveau ", elles sont évaluées de gauche à droite.
^ exponentielle	Evalué en 2e. S'il y en a plusieurs, elles sont évaluées de gauche à droite.
- négation	Evalué en 3e. S'il y en a plusieurs, elles sont évaluées de gauche à droite.
* ou / multiplication et division	Evalué en 4e. S'il y en a plusieurs, elles sont évaluées de gauche à droite.
\ division entière	Evalué en 5e. S'il y en a plusieurs, elles sont évaluées de gauche à droite.
Mod modulo (17 Mod 3 = **2** car 17 = 3*5 + **2**)	Evalué en 6e. S'il y en a plusieurs, elles sont évaluées de gauche à droite.
+ ou - addition et soustraction	Evalué en dernier. S'il y en a plusieurs, elles

	sont évaluées de gauche à droite.

3.5 Opérateurs de comparaison

Dans l'algèbre standard	Equivalent dans Visual Basic
a = b	a = b
c ≠ d	c < > d
e > f	e > f
g < h	g < h
i ≥ j	i >= j
k ≤ l	k <= l

Ces opérateurs permettent au programme de prendre une décision basée sur la vérification, ou non, (*True* ou *False*) d'une condition.

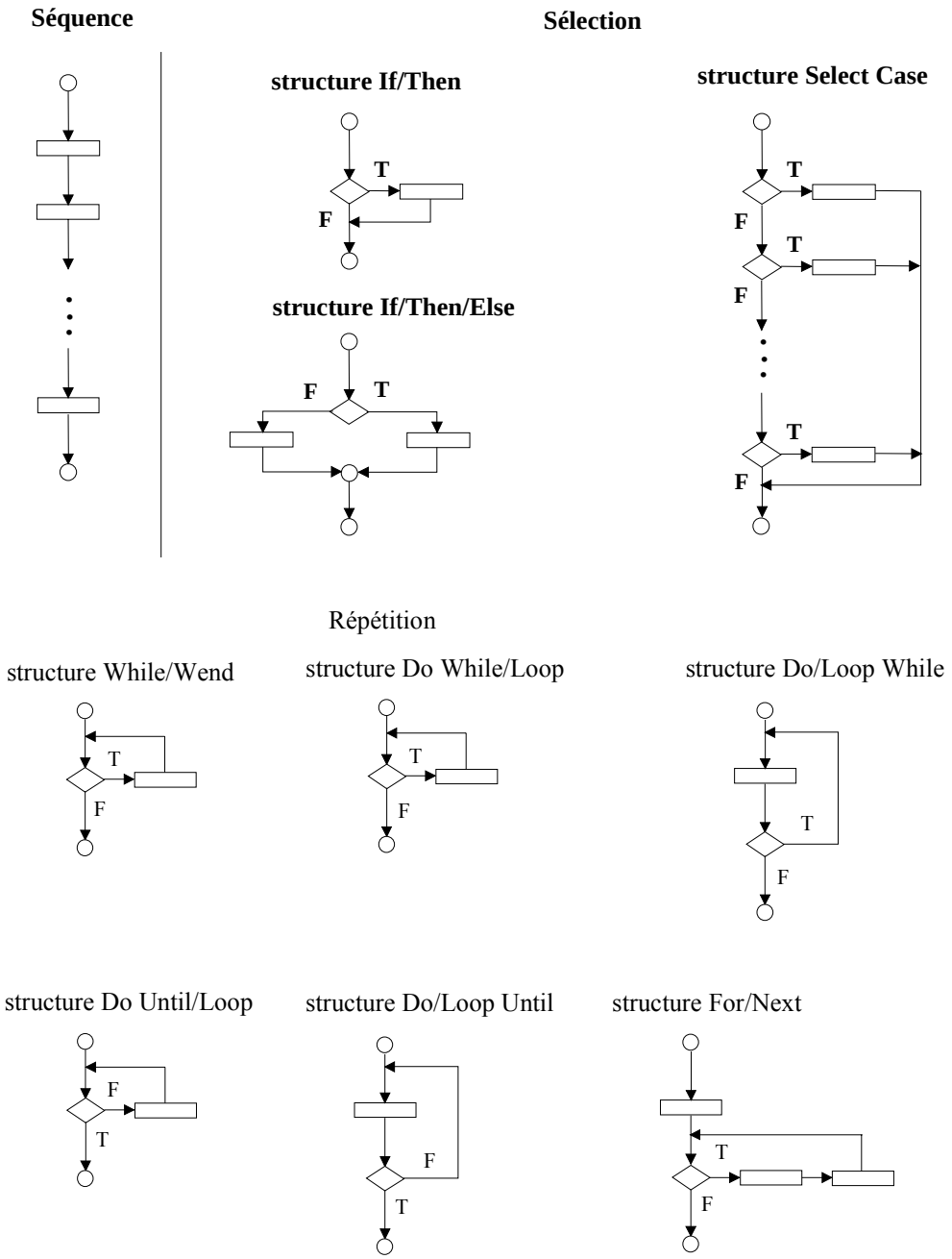
4 STRUCTURES DE CONTRÔLE

Afin de concevoir un programme, nous allons décrire les différents types de blocs de construction possibles et rappeler les principes de la programmation structurée. N'importe quel problème informatique peut-être résolu en exécutant une série d'actions dans un ordre spécifique, une telle procédure est appelée un *algorithme*.

Lorsque les instructions d'un programme sont exécutées l'une à la suite de l'autre, dans l'ordre où elles ont été écrites, l'exécution est dite *séquentielle*. Certaines instructions permettent au programmeur de spécifier que la prochaine instruction à exécuter peut être différente de celle qui est située dans la séquence, ces instructions effectuent un *transfert de contrôle*.
Durant les années 1960, l'utilisation de l'instruction de transfert de contrôle *goto* fut bannie comme étant la source de nombreux problèmes de programmation (difficultés vis-à-vis de la compréhension, du test, de la modification d'un programme). La programmation dite *structurée* (" *sans goto* ") fut présentée comme une alternative. En fait tous les programmes peuvent être construits par combinaison de seulement trois structures de contrôle : La structure *en séquence*, la structure *de sélection* et la structure *de répétition*.

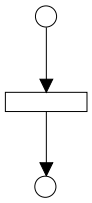
Visual Basic propose :
- trois types de structures de sélection : *If/Then, If/Then/Else, Select Case*,
- six types de structures de répétition : *While/Wend, Do While/Loop, Do/Loop While, Do Until/Loop, Do/Loop Until, For/Next.*

La figure suivante récapitule les différentes structures de contrôle de Visual Basic.



La connexion arbitraire de telles structures peut induire des programmes non structurés. Les règles suivantes permettent la construction de programmes structurés (le symbole rectangle peut être utilisé pour indiquer une action, y compris l'entrée/sortie) :

1. Commencer avec le diagramme (*flowchart*) suivant :



2. Tout rectangle (action) peut être remplacé par deux rectangles en séquence.

3. Tout rectangle (action) peut être remplacé par une structure de contrôle (séquence, *If/Then, If/Then/Else, Select Case, While/Wend, Do While/Loop, Do/Loop While, Do Until/Loop, Do/Loop Until, For/Next*).
4. Les règles 2 et 3 peuvent être appliquées autant de fois que nécessaire et dans n'importe quel ordre.

4.1 Structures de sélection

- Structure de type *If/Then*

```
If note >= 10 Then
        lblNote.Caption = "Accepté"
End If
```

- Structure de type *If/Then/Else*

```
If note >= 10 Then
        lblNote.Caption = "Accepté"
Else
        lbl.Caption = "Refusé"
End If
```

- Structure de type *Select Case*

Cette structure permet de ne pas utiliser des *If* imbriqués. Les conditions sont examinées dans l'ordre d'écriture, dès qu'une condition est vérifiée, les suivantes ne sont pas examinées.

```
Select Case code_d_acces
        Case Is < 1000
                Message = "Accès refusé"
        Case 1542, 1645 To 1689
                Message = "Personnel technique"
        Case Else
                Message = "Accès refusé"
End Select
```

4.2 Structures de répétition

- Structure de type *While/Wend*

```
Private Sub cmdBouton_Click()
    Dim produit As Integer
    produit = 2
    While produit <= 1000
            Print produit
            produit = produit * 2
    Wend
 End Sub
```

- Structure de type *Do While/Loop*

Cette structure se comporte comme la précédente.

```
Private Sub cmdBouton_Click()
    Dim produit As Integer
    produit = 2
    Do While produit <= 1000
            Print produit
            produit = produit * 2
    Loop
 End Sub
```

- Structure de type *Do/Loop While*

Cette structure est similaire à la précédente si ce n'est que la condition de test de la boucle est faite après que le corps de la boucle ait été exécuté.

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    compteur = 1
    Do
            Print compteur & Space$(2) ;        ' permet d'écrire 1  2  3  4  …  10
            compteur = compteur + 1
    Loop While compteur <= 10
End Sub
```

- Structure de type *Do Until/Loop*

```
Private Sub cmdBouton_Click()
    Dim produit As Integer
    produit = 2
    Do Until produit > 1000
            Print produit
            produit = produit * 2
    Loop
End Sub
```

Au contraire des trois précédentes structures, les instructions, situées dans le corps de la boucle, sont exécutées autant de fois que la condition de test de la boucle est fausse.

- Structure de type *Do/Loop Until*

Cette structure est similaire à la précédente si ce n'est que la condition de test de la boucle est faite après que le corps de la boucle ait été exécuté.

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    compteur = 1
    Do
            Print compteur & Space$(2) ;
            compteur = compteur + 1
    Loop Until compteur = 10
```

```
End Sub
```

- Structure de type *For/Next*

A titre d'exemple, réécrivons la portion de programme suivante qui utilise la structure *Do While/Loop* :

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    compteur = 3
    Do While compteur <= 20
            Print compteur
            compteur = compteur + 2
    Loop
End Sub
```

En fait l'initialisation, la condition de répétition et l'incrémentation sont incluses dans la structure d'entête *For* :

```
Private Sub cmdBouton_Click()
    Dim compteur As Integer
    For compteur = 3 To 20 Step 2
            Print compteur
    Next compteur        ' la présence du nom de la variable compteur est facultative
End Sub
```

Remarque : L'incrément, ici égal à 2, peut-être négatif. Par défaut, sa valeur est unitaire.

Remarque : Les instructions *Exit Do* et *Exit For* permettent la sortie immédiate de ces structures (*Exit For* est réservée à la structure *For/Next*).

4.3 Opérateurs logiques

Liste des opérateurs permettant d'effectuer des opérations logiques à partir d'expressions à valeurs booléennes *True* (vraie) ou *False* (fausse) :

Not *Expression*	Etablit la négation logique de *Expression*
E1 **And** *E2*	*True* si les conditions *E1* et *E2* sont *True*
E1 **Or** *E2*	*True* si l'une au moins des conditions *E1, E2* est *True*
E1 **Xor** *E2*	*True* si seulement une seule des deux conditions *E1, E2* est *True*

4.4 Types de données

- Les types de données décrivent l'information qu'une variable stocke. Les types prédéfinis dans Visual Basic sont listés dans le tableau ci-dessous.

Type	Occupation mémoire	Portée des valeurs
Boolean	2 octets	*True* ou *False*
Byte	1 octet	0 à 255 (2^8)
Currency	8 octets	-922 337 203 685 477.580 8 à 922 337 203 685 477.580 7
Date	8 octets	1er janvier 100 au 31 décembre 9999 0:00:00 à 23:59:59

Double	8 octets	-1.797 693 134 862 32 E308 à – 4.940 656 458 412 47 E-324 4.940 656 458 412 47 E-324 à 1.797 693 134 862 32 E308
Integer	2 octets	-32 768 à 32 767 (2^{16}, signé)
Long	4 octets	-2 147 483 648 à 2 147 483 647 (2^{32}, signé)
Object	4 octets	Pointeur sur un objet (voir chp. 12, annexe F)
Single	4 octets	-3.402 823 E38 à -1.401 298 E-45 1.401 298 E-45 à 3.402 823 E38
String	10 octets + taille de la chaîne	La longueur (variable) de la chaîne est composée d'au plus 2 147 483 648 caractères
String*_n_	Taille de la chaîne	La taille de la chaîne, fixée par *n*, est comprise entre 1 et 65536
Variant	16 octets	Une valeur parmi celles listées au-dessus, mise à part *String*n*

Currency est un type de données, stocké sur 8 octets, qui permet, notamment, des calculs monétaires précis dans le cas de nombres de moins de 4 décimales.
Variant est le type de données par défaut pour les variable dont le type n'est pas explicitement déclaré et stocke n'importe quel type de données (excepté *String*n*), y compris les types définis par le programmeur (voir ci-dessous). Ce type de données peut-être intéressant :
- dans le cadre de la conception d'une interface utilisateur, au sens où il permet de prendre en compte des données dont on ne connaît pas le type. Par exemple, à la question " entrer votre âge ", un utilisateur peut introduire un nombre, mais aussi du texte (" vingt-deux " au lieu de 22) ; le fait de stocker cette information dans une variable *Variant* permet, *via* les fonctions *VarType()* ou *TypeName()*, de connaître son type.
- pour construire des structures de données (listes chaînées, arbres), sachant que le code manipulant de telles structures peut être écrit indépendamment du type de données des éléments stockés.

- Visual Basic permet la création de ses propres types de données. Il s'agit de regrouper sous un seul type des collections de variables non nécessairement de même type (au contraire des tableaux).

Il suffit de répertorier, entre les instructions *Type* et *End Type*, l'ensemble des variables pour définir un nouveau type, voir l'exemple ci-dessous.

```
Type Client
        Nom As String*15
        Prénom As String*15
        Age As Integer
End Type
```

Le type *Client* étant défini, il est possible de déclarer une nouvelle variable basée sur ce type, par exemple :

```
Dim Untel As Client
Untel.Nom = "Dupond"
Untel.Prénom = "Léon"
Untel.Age = 40
```

Usuellement, ces types de données sont utilisés dans le cas de manipulation de fichiers à accès direct (fichiers constitués d'enregistrements de longueur fixe).

Remarque : Les définitions *Type* doivent être privées (*Private*) quand elles sont déclarées dans des modules feuilles, elles peuvent être publiques (*Public*) ou privées quand elles sont déclarées dans des modules code.

5 PROCÉDURES ET FONCTIONS

5.1 Introduction

L'expérience montre que le fait de fractionner un programme (important) en plusieurs parties facilite bien souvent son développement et sa maintenance. Cette approche, applicable, notamment, dans le cadre de Visual Basic, est appelée " *diviser et conquérir* ".

5.2 Les modules

Un projet est constitué de modules, tels que des modules *feuilles*, ou des modules *standards* (encore appelés modules *codes*). A la différence d'un module *feuille*, un module *standard* ne comporte pas d'IUG, il est uniquement composé de codes de programme.
En dehors de déclarations, les modules sont constitués de procédures, quatre types de procédures existent :

- les procédures *événementielles*, en anglais *event procedures*, en réponse à des événements (appui sur une touche du clavier, clic de souris, etc.) ;
- les procédures *Visual Basic*, en anglais *Visual Basic procedures*, (*Print, VarType*, etc.), fournies par Microsoft pour effectuer des tâches courantes (le but cherché étant de réduire le temps de développement) ;
- les procédures conçues par le programmeur (car non fournies par Visual Basic). Elles sont de deux types : Les procédures *Sub* et les procédures *Function*.

Un programmeur peut écrire des procédures pour exécuter des tâches spécifiques pouvant être utilisées à différents endroits dans un programme. A des fins de réutilisation éventuelle (en anglais, on parle de *software reusability*), chaque procédure devra être limitée à la résolution d'une tâche bien définie, il est de coutume que la taille d'une procédure ne dépasse pas une demie page.

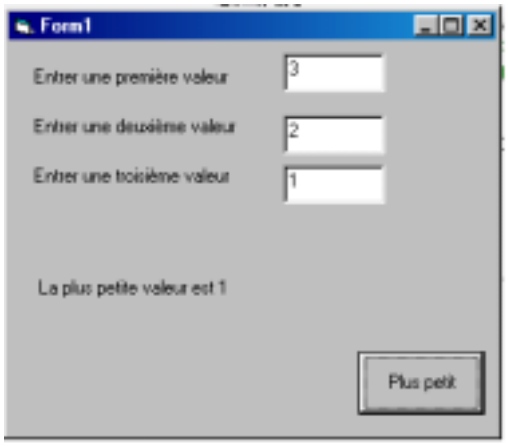
L'appel à une procédure se fait en spécifiant le nom de la procédure et en fournissant les informations (on parle de paramètres d'entrée) nécessaires à la procédure " appelée " pour réaliser le travail demandé (restitué au programme " appelant " *via* des paramètres de sortie).

5.3 Les procédures *Sub*

Voici un exemple de procédure *Sub* qui détermine la plus petite valeur parmi trois nombres entiers et l'affiche dans un *Label* (voir la recopie d'écran dans la figure qui suit) :

```
' Ce programme permet de trouver le minimum parmi trois nombres entiers
Option Explicit  ' Force les variables à être explicitement déclarées

Private Sub cmdPlusPetit_Click()
        Dim valeur1 As Long, valeur2 As Long, valeur3 As Long
        valeur1 = txtUn.Text
        valeur2 = txtDeux.Text
        valeur3 = txtTrois.Text
        Call Minimum(valeur1, valeur2, valeur3)   ' ou Minimum valeur1, valeur2, valeur3
End Sub

Private Sub Minimum (min As Long, y As Long, z As Long)
```

```
        If y < min Then
                min = y
        End If
        If z < min Then
                min = z
        End If
        lblPlusPetit.Caption = "La plus petite valeur est" & min
End Sub
```



Le corps de la procédure *Minimum* est placé entre l'entête et *End Sub*. A chaque appel de cette procédure (cf. instruction *Call Minimum(valeur1, valeur2, valeur3),* ou de manière équivalente, *Minimum valeur1, valeur2, valeur3*), le corps de la procédure est immédiatement exécuté, puis l'exécution du programme se poursuit par l'instruction située immédiatement après l'appel de la procédure. Toutes les entêtes des procédures contiennent des parenthèses, éventuellement vides ou contenant une liste de déclarations de variables, appelée liste des paramètres qui comprend des paramètres d'entrée et de sortie. En fait, une procédure peut modifier le contenu des variables passées en paramètre (c'est le cas du paramètre *min* de la procédure *Minimum*).

Remarques :
- Le fait que les procédures opèrent sur des données, explique que leurs noms correspondent bien souvent à des verbes. Il serait d'ailleurs préférable de renommer la procédure *Minimum* par *RechercherLeMinimum*.
- Par convention, une procédure débute par une lettre majuscule (par exemple : *Envoyer*).

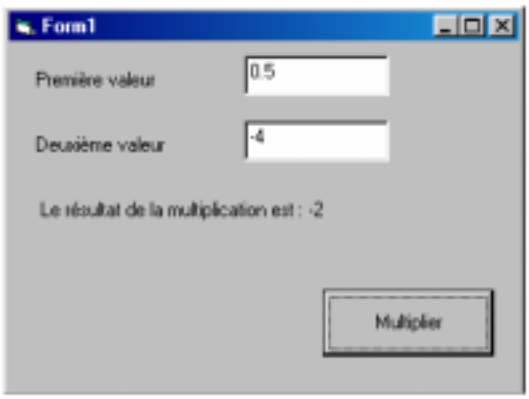
5.4 Les procédures *Function*

Les procédures *Function* partagent les mêmes caractéristiques que les procédures *Sub* si ce n'est que les procédures *Function* retournent une valeur, appelée *valeur de retour*. La plupart des procédures fournies par Visual Basic sont de ce type, par exemple, la fonction *VarType (Name)* (retourne un entier indiquant le type de la variable *Name* de type *Variant*).

Voici un exemple de procédure *Function* qui multiplie deux valeurs réelles et affiche le résultat dans un *Label* (voir la recopie d'écran dans la figure qui suit) :

```
' Ce programme permet de multiplier deux nombres
```

```
Option Explicit  ‘ Force les variables à être explicitement déclarées

Private Sub cmdMultiplier_Click()
        Dim valeur1 As Single, valeur2 As Single
        valeur1 = Val(txt1.Text)     ‘ La fonction Val convertit une chaîne en un nombre
        valeur2 = Val(txt2.Text)
        lblMultiplication.Caption = "Le résultat de la multiplication est : " & _
        Multiplication(valeur1, valeur2)
End Sub

Private Function Multiplication (x As Single, y As Single) As Single
        Multiplication = x*y
End Function
```



La valeur, retournée par la fonction *Multiplication*, est placée dans le nom de la fonction (on parle de *pseudo-variable*).

5.5 Appel par valeur, appel par référence

Le passage des paramètres d’une procédure peut se faire *par valeur* (en anglais *by value*) ou *par référence* (en anglais *by reference*). Dans les deux exemples précédents, chacun des arguments a été passé *par référence*, ce qui est le mode de passage par défaut. Avec un tel mode de passage, le programme " appelant " donne à la procédure " appelée " la possibilité d’*accéder directement* aux données de " l’appelant " et donc de modifier ces données. Quand un argument est passé *par valeur*, une copie de la valeur de l’argument est faite et passée à la procédure " appelée ". Cette dernière peut manipuler cette copie, mais ne peut pas manipuler la donnée d’origine de " l’appelant ". Notons que ce mode de passage amoindrit les performances vis-à-vis du temps d’exécution et de l’espace mémoire, dans le cas d’un grand nombre de paramètres à passer.

L’entête de la fonction suivante déclare deux variables :

```
Function Calcul (ByVal x As Long, y As Boolean) As Double
```

La fonction *Calcul* reçoit *x par valeur* et *y par référence*, cette entête peut aussi s’écrire :

Function Calcul (ByVal x As Long, ByRef y As Boolean) As Double

5.6 Exit Sub et Exit Function

Le fait d'exécuter l'instruction *Exit Sub* (respectivement *Exit Function*) dans une procédure *Sub* (respectivement *Function*) provoque une sortie immédiate de la procédure. Le contrôle est retourné à " l'appelant " et l'instruction située immédiatement en séquence après l'appel est exécutée.

5.7 Durée de vie d'une variable

Une variable possède différents attributs : Un nom, un type, une taille et une valeur. A cela s'ajoutent d'autres attributs : Une durée de vie (en anglais *storage class*), une portée (en anglais *scope*).
L'attribut *durée de vie* d'une variable détermine la période durant laquelle elle existe en mémoire, certaines existent brièvement, d'autres sont régulièrement créées et détruites, d'autres existent tout au long de la durée d'exécution du programme.
Les variables locales (i.e., déclarées dans une procédure ou une fonction) *non statiques* (aussi appelées *variables automatiques*) sont définies par défaut et sont créées quand la procédure devient active. Ces variables existent (seulement) jusqu'à la fin d'exécution de la procédure.
Le mot clé *Static* est utilisé pour déclarer des variables locales *statiques*. De telles variables sont toujours connues seulement dans la procédure dans laquelle elles ont été déclarées (on dit qu'elles ont la même portées), mais à la différence des *variables automatiques*, les variables *statiques* conservent leurs valeurs (i.e., celles acquises lors de la dernière sortie de la procédure). Ces variables sont implicitement initialisées à zéro lors du premier appel de la procédure. Ce type de variables permet de ne pas déclarer des variables globales lorsqu'une seule procédure en a besoin.

Exemple :

```
Private Sub Exemple_Click ()
        Static N As Integer   ' N est une variable entière statique
        Dim M As Integer      ' M est une variable entière non statique

        N = N + 1
        M = M + 1
End Sub
```

Lors du premier appel de cette procédure, les variables *N* et *M* sont égales à 0, aussi lors de la première sortie de cette procédure, les variables *N* et *M* sont égales à 1.
Lors du second appel, la variable *N* est égale à 1 (valeur acquise lors de la précédente sortie) alors que la variable *M* est toujours égale à 0. Aussi lors de la seconde sortie de cette procédure, la variable *N* est égale à 2 alors que la variable *M* est toujours égale à 1.

5.8 Portée d'une variable, d'une procédure, d'une fonction

L'étendue d'un *identificateur* (nom d'une variable, ou nom d'une procédure définie par le programmeur) est la région dans laquelle l'*identificateur* peut-être référencé. Par exemple, une variable déclarée en local dans une procédure peut seulement être référencée dans cette procédure. Les trois portées d'un *identificateur* sont la portée *locale* (en anglais *local scope*), la portée au niveau *module* (en anglais *module scope*) et la portée au niveau *public* (en anglais *public scope*).
La portée *locale* s'applique à des variables déclarées dans le corps d'une procédure. Les variables locales peuvent être référencées à partir de l'endroit où elles ont été déclarées jusqu'à la sortie de la procédure (ou de la fonction), i.e., *End Sub* (ou *End Function*).

La portée au niveau *module*, aussi appelée portée au niveau *privé* (en anglais *Private scope*), s'applique à des variables déclarées dans la partie " déclaration générale du module " *via* le mot clé *Dim*. Ces variables peuvent seulement être référencées dans le module dans lequel elles ont été déclarées.
La portée au niveau *public* fait référence à des variables déclarées en *public* dans un module. De telles variables sont accessibles pour tous les modules.

Remarque : Quand une variable est déclarée au niveau *module* et a le même nom qu'une variable déclarée en local, celle qui est déclarée au niveau *module* est " cachée " tant que la variable déclarée en local est active. Une alternative à cette source d'erreur consiste à ne pas utiliser des noms de variables dupliquées.

5.9 Les constantes

Visual Basic permet la création de variables dont la valeur ne peut pas changer durant l'exécution d'un programme. Ces variables, particulières, sont appelées des *variables constantes*[1] et sont souvent utilisées pour améliorer la " lisibilité " d'un programme. La déclaration d'une variable constante se fait à l'aide du mot clé *Const* :

Syntaxe

[**Public** | **Private**] **Const** *constname* [**As** *type*] **=** *expression*

Private est utilisé au niveau *module* pour déclarer les constantes uniquement disponibles dans le module dans lequel la déclaration est effectuée. Ce mot clé n'est pas autorisé dans les procédures.
Public est utilisé au niveau *module* pour déclarer des constantes disponibles pour toutes les procédures de l'ensemble des modules. Ce mot clé n'est pas autorisé dans les procédures.

Remarque : Utiliser le mot clé *Dim* avec *Const* dans une déclaration est une erreur de syntaxe.

A titre d'exemple, on peut donner les lignes suivantes :

```
Const pi As Double = 3.14159        ' il est obligatoire d'assigner une valeur
Const Deux_pi As Double = pi*2
Public Const RG = "Rouge"
```

En fait, de nombreuses constantes sont reconnues dans Visual Basic, elles sont généralement préfixées par les lettres *vb*. Par exemple, dans le cadre d'une boîte de dialogue, la constante *vbOK* (renvoyée par la fonction *MsgBox*) permet de détecter un appui (un clic) sur le bouton *OK*.

5.10 Paramètres optionnels

Il est possible de créer des procédures qui acceptent un ou plusieurs *paramètres optionnels*. Ces paramètres sont spécifiés dans l'entête de la procédure *via* le mot clé *Optional*. Considérons, par exemple, la procédure suivante :

```
Private Sub Ajout (x As Integer, Optional y As Integer = 1)
    x = x + y
End Sub
```

L'entête de cette procédure indique que le second argument peut ne pas être demandé lors d'un appel de la procédure *Ajout*, auquel cas il sera égal à 1. Considérons les appels suivants :

[1] oxymoron : Figure de style qui réunit deux mots en apparence condratictoires (silence éloquent).

Call Ajout
Call Ajout (a)
Call Ajout (a, b)

Le premier appel génère une erreur de syntaxe car un argument au minimum est réclamé. Le second appel est valide, l'argument optionnel n'est pas fourni. Au retour de la procédure, la variable entière *a* est incrémentée par défaut de 1. Le troisième appel est aussi valide, la variable entière *b* est fournie en tant qu'argument optionnel. Au retour de la procédure, la variable entière *a* est incrémentée par défaut de *b*.

5.11 Fonctions mathématiques de Visual Basic

Certaines fonctions de calcul mathématique sont disponibles :

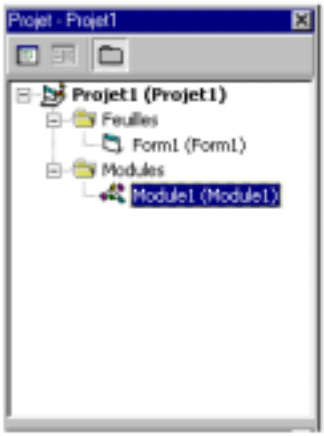
Fonction	Description	Exemple
Abs (x)	Valeur absolue de *x*	Abs (-2) est égal à 2, Abs (3) est égal à 3
Atn (x)	Arc tangente (en radians) de *x*	Atn (1) est égal à pi/4
Cos (x)	Cosinus (en radians) de *x*	Cos (0) est égal à 1
Exp (x)	Fonction exponentielle e^x	Exp (1.0) est égal à 2.71828
Int (x)	Retourne le plus grand entier inférieur ou égal à *x*	Int (-5.3) est égal à – 6 Int (0.893) est égal à 0 Int (76.45) est égal à 76
Fix (x)	Partie entière de *x*	Fix (-5.3) est égal à –5 Fix (0.893) est égal à 0 Fix (76.45) est égal à 76
Log (x)	Logarithme népérien de *x*	Log (2.718282) est égal à 1.0
Round (x, y)	Retourne une valeur approchée de *x* avec *y* décimales (si *y* est omis, *x* est retourné comme étant un entier)	Round (4.84) est égal à 4 Round (5.73839, 3) est égal à 5.738
Sgn (x)	Signe de *x*	Sgn (-19) est égal à –1 Sgn (0) est égal à 0 Sgn (3.4) est égal à 1
Sin (x)	Sinus (en radians) de *x*	Sin (0) est égal à 0
Sqr (x)	Racine carré de *x*	Sqr (9.0) est égal à 3
Tan (x)	Tangente (en radians) de *x*	Tan (0) est égal à 0

5.12 Module standard

Un module standard (ou module code) n'a pas d'IUG (au contraire des modules feuilles, en anglais *form modules*), il contient uniquement du code. Les procédures que le programmeur souhaite utiliser dans de nombreux projets (on parle de *briques logicielles*) sont souvent placées dans des modules standards. Comme pour un module feuille, un module code admet une partie " déclaration générale ".
Par défaut, les variables d'un module et les procédures événementielles sont privées (*Private*) au module dans lesquelles elles ont été définies : Une variable, ou une procédure, privée peut seulement être utilisée dans le module où elle a été déclarée. Si le programmeur souhaite permettre à un autre module l'utilisation d'une variable, ou procédure, le mot clé *Public* doit être utilisé dans la déclaration : Les variables, ou procédures, publiques sont accessibles dans chacun des modules du projet.

Remarques :

- Les modules standards sont tous placés dans un classeur nommé *Modules*, alors que les feuilles sont placées dans un classeur nommé *Feuilles* (cf. figure suivante).



- Les noms des fichiers de modules standards se terminent par **.bas**.
- Visual Basic ne propose pas de module standard par défaut dans un projet, il faut l'ajouter manuellement (voir le menu *Projet | Ajouter un module*).
- Un projet peut avoir plusieurs modules codes (aussi bien que plusieurs modules feuilles).

A titre d'exemple, le programme suivant contient un module feuille et un module code. Le fait de cliquer sur le bouton *Ecrire* de la feuille (voir la saisie d'écran dans la figure qui suit) appelle la procédure *ModuleEcrire*, déclarée en *public* et située dans un module standard.

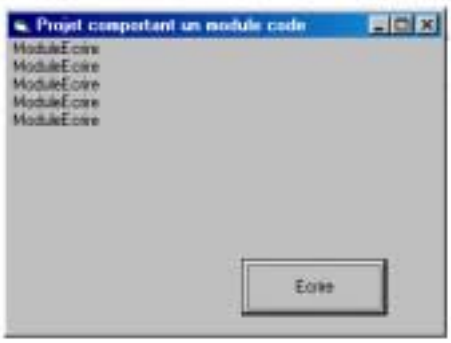
_____ *.frm* _____

```
' Utilisation d'un module code
Option explicit

Private Sub cmdEcrire_Click ()
        ' La procédure ModuleEcrire est définie dans le module code (modModule.bas)
        Call ModuleEcrire
End Sub
```

_____ *.bas* _____

```
' modModule.bas
Option Explicit

Public Sub ModuleEcrire ()
        frmFeuille.Print "ModuleEcrire"
End Sub
```

_____



6 LES TABLEAUX

Un tableau peut se représenter comme étant un groupe consécutif (une série) d'emplacements mémoires de même nom et de même type (voir § 4.4). La référence à un emplacement particulier, ou élément dans le tableau, se fait à travers le nom du tableau et sa position (son index) dans le tableau. Le traitement des variables contenues dans un tableau peut alors se faire à l'aide de boucle.

Par exemple, soit un tableau, mono dimensionnel, nommé *Nombre* contenant 6 éléments de type entiers :

Nombre (0)	55
Nombre (1)	22
Nombre (2)	15
Nombre (3)	1
Nombre (4)	89
Nombre (5)	56

Notons qu'il existe deux catégories de tableaux : Les tableaux *statiques* (en anglais, *static arrays*) dont le nombre maximum d'éléments est fixé, et les tableaux *dynamiques* (en anglais, *dynamic arrays*, aussi appelé *redimmable arrays*) dont le nombre d'éléments peut varier durant le déroulement du programme.

6.1 Leurs déclarations

Les tableaux peuvent être déclarés comme *Public* uniquement dans un module code. Les tableaux actifs au niveau d'un module sont déclarés dans la partie " déclarations générales " du module *via* le mot clé *Dim* ou *Private*. Les tableaux actifs au niveau *local* sont déclarés dans une procédure *via* le mot clé *Dim* ou *Static*.

La déclaration

Dim Nombre (5) As Integer

déclare un tableau *Nombre* constitué de six entiers. Les fonctions *LBound (Nombre)* et *UBound (Nombre)* retournent respectivement l'indice le plus petit et l'indice le plus grand du tableau, l'indice étant de type *Long*. Par défaut, le plus petit indice d'un tableau est égal à 0. Le fait de spécifier une valeur pour le plus grand indice d'un tableau, dans notre exemple 5, indique que le tableau est statique. Par défaut, les six entiers du tableau sont égaux à 0, la ligne

Nombre (0) = 55

permet de fixer la valeur 55 à l'élément du tableau *Nombre* d'indice 0.

La déclaration

Dim Tab (-5 To 4) As String

déclare un tableau *Tab* constitué de 10 chaînes de caractères, *LBound (Tab)* et *UBound (Tab)* sont respectivement égaux à –5 et 4.

Les tableaux peuvent être multi-dimensionnels, la déclaration

Dim Tab3 (50 To 100, 8, 7 To 15)

déclare un tableau *Tab* de dimension 3, *LBound (Tab, 3)* est égal à 7, *UBound (Tab, 1)* est égal à 100.

6.2 Les tableaux dynamiques

Les tableaux dont la taille peut augmenter, ou diminuer, en cours d'exécution sont appelés tableaux *dynamiques*. Aucune taille n'est donnée lors de la déclaration d'un tel tableau. Par exemple, la ligne

Dim TabDynamique () As Double

déclare le tableau *TabDynamique* comme étant un tableau dynamique constitué de réels doubles. La taille d'un tableau dynamique est spécifiée en cours d'exécution à l'aide du mot clé *ReDim* (notons que la dimension du tableau ne peut pas être modifiée). Par exemple, la ligne

ReDim TabDynamique (10)

alloue 11 éléments à *TabDynamique*. Le fait d'exécuter *ReDim* provoque la perte des valeurs contenues dans le tableau. Toutefois, les valeurs déjà dans le tableau peuvent être conservées en plaçant le mot clé *Preserve* après *ReDim*.

Le mot clé *Erase Tab* permet, en cours d'exécution, de supprimer de la mémoire le tableau dynamique *Tab*.

6.3 Passage de tableaux dans les procédures

Pour passer tous les éléments d'un tableau en paramètre à une procédure, il suffit de spécifier le nom du tableau suivi d'une paire de parenthèses vide. Par exemple, si le tableau *TemperatureMensuelle* est déclaré comme suit :

Dim TemperatureMensuelle (12) As Integer

l'appel

Call ModifierTableau (TemperatureMensuelle ( ))

passe tous les éléments du tableau *TemperatureMensuelle* à la procédure *ModifierTableau*. Les tableaux sont automatiquement passés par référence – l'appelé peut donc modifier la valeur des éléments du tableau fourni par l'appelant.
Pour passer un élément d'un tableau à une procédure, utiliser l'élément du tableau comme paramètre, par exemple :

Call PasserUnElement (TemperatureMensuelle (4))

Pour une procédure recevant un tableau, la liste des paramètres de la procédure doit spécifier le passage d'un tableau. Par exemple, l'entête de la procédure *ModifierTableau* devra s'écrire comme suit :

Private Sub ModifierTableau (a( ) As Integer)

ainsi, *ModifierTableau* s'attend à recevoir un tableau d'entier dans le paramètre *a* (la taille du tableau n'est pas spécifiée entre les parenthèses).

7 LES CHAÎNES

Chaque caractère est représenté en interne par un nombre entier, compris entre 0 et 255 (par exemple, 65 représente la lettre *A*). Cet ensemble de valeurs entières est appelé l'ensemble des caractères *American National Standards Institute (ANSI)*. Les 128 premiers caractères *ANSI* (de 0 à 127) correspondent aux valeurs de l'*American Standard Code for Information Interchange (ASCII)*. Les valeurs *ANSI* de 128 à 255 représentent un ensemble de caractères spéciaux, d'accents, de fractions, etc. Une chaîne est une série de caractères formant une entité. Une chaîne dans Visual Basic a un type de donnée *String*. Deux types de chaînes sont possibles : Les chaînes de longueur variable (en anglais, *variable-length strings*), les chaînes de longueur fixe (en anglais, *fixed-length strings*).
Par défaut, les variables *String* sont de longueur variable, exemple :

```
Dim ch As String
ch = "02 41 – abc"
```

la variable *ch* est déclarée de type *String*, elle contient la chaîne "02 41 – abc".
La chaîne (*NomVariable*) de longueur fixe (*TailleChaîne*) est déclarée comme suit :

Dim *NomVariable* **As String *** *TailleChaîne*

La fonction *Len* (pour *length*) est utilisée pour déterminer la longueur d'une chaîne, par exemple, *Len* ("*02 41 – abc*") est égal à 11.

7.1 Concaténation avec & (esperluette) et +

```
ch1 = "Pro"
ch2 = "gramme"
ch3 = ch1 & ch2
ch4 = ch1 + ch2
```

ch3 et *ch4* contiennent la chaîne "*Programme*". Les opérateurs & et + sont équivalents dans le cas où les opérandes sont des chaînes. L'utilisation de l'opération + entre des opérandes de type différent, par exemple, la ligne

ch = "bonjour" + 22

provoque une erreur car Visual Basic tente de convertir la chaîne "*bonjour*" en un nombre, de manière à l'additionner (+) à 22. Pour cette raison, il est préférable d'utiliser l'opération & pour concaténer des chaînes.

7.2 Comparaison de chaînes

Il est fréquent de devoir comparer deux chaînes. Pour cela, on dispose de la fonction *StrComp* et des opérateurs <, <=, >, >=, =, <>. La comparaison entre deux chaînes est basée sur les différentes valeurs *ANSI* codant les chaînes.
La fonction *StrComp (ch1, ch2)* retourne 0 lorsque les chaînes *ch1* et *ch2* sont égales, -1 si la chaîne *ch1* est inférieure à *ch2*, 1 si la chaîne *ch1* est supérieure à *ch2*. A titre d'exemple, *StrComp ("A", "B")* est égal à –1 car le code *ANSI* de la lettre *A* (= 65) est inférieur à celui de la lettre *B* (= 66). Une option permet d'être *case-sensitive* ou *case-insensitive*.

- L'opérateur *Like* fournit un autre moyen de comparer deux chaînes.

7.3 Manipulation de caractères dans une chaîne

Visual Basic fournit différents moyens permettant la manipulation de caractères dans une chaîne.

La fonction *Mid$ (ch, pos, nb)* retourne une sous-chaîne de la chaîne *ch* constituée de *nb* caractères à partir du caractère indiqué par la position *pos* (la position du premier caractère est égale à 1).
Par exemple :

```
sous_ch = Mid$ ("programme", 2, 3)      ' sous_ch = "rog"
```

Cette fonction permet aussi de remplacer une portion de chaîne par une autre. Par exemple, les lignes

```
ch = "Bonjour monsieur"
Mid$ (ch, 9, 1) = "M"
```

change le contenu de *ch* en "*Bonjour Monsieur*".

7.4 *Left$, Right$, InStr, InStrRev, Split, Join*

La fonction *Left$ (ch, nb)* retourne une chaîne composée de *nb* caractères situés dans la partie gauche de la chaîne *ch*, la fonction *Right$ (ch, nb)* fait de même pour la partie droite de la chaîne. A titre d'exemple, considérons les lignes suivantes

```
ch = "Bonjour madame"
ch1 = Left$ (ch, 7)          ' ch1 = "Bonjour"
ch2 = Right$ (ch, 6)         ' ch2 = "madame"
```

La fonction *InStr (pos, source, rech)* retourne, si possible, un nombre entier représentant la position dans la chaîne de base *source* de la chaîne cherchée *rech*, cette recherche débutant à partir de la position *pos*. Si la chaîne *rech* n'est pas détectée, la fonction *InStr* renvoie une valeur nulle. A titre d'exemple, considérons les lignes suivantes

```
ch = "Bonjour madame"
i = InStr (1, ch, "on")          ' i = 2
j = InStr (2, ch, "on")          ' j = 2
k = InStr (3, ch, "on")          ' k = 0
```

La fonction *InStrRev* a un mode de fonctionnement similaire à celui de *InStr*, si ce n'est que la recherche débute à partir de la fin de la chaîne (i.e., de droite à gauche dans la chaîne de base).

La fonction *Split (expression, délimiteur)* retourne dans un tableau (en fait un vecteur) des sous-chaînes de caractères. L'extraction des sous-chaînes de la chaîne *expression* se fait au vue du *délimiteur,* par exemple :

```
Dim tableau() As String
tableau = Split ("ab" "c" "d1f", " ")
```

retourne les sous-chaînes "*ab*" dans *tableau(0),* "*c*" dans *tableau(1)* et "*d1f*" dans *tableau(2).*

La fonction *Join (tableau source, délimiteur)* est la fonction duale de *Split.* Elle retourne une chaîne joignant les sous-chaînes contenues dans le tableau (en fait un vecteur) *tableau source,* les sous-chaînes étant séparées par un *délimiteur*. Par exemple,

```
Dim ch As String
ch = Join (tableau, ".")
```

retourne la chaîne "*ab.c.d1f*" dans la variable *ch*.

7.5 *LTrim$, RTrim$* et *Trim$*

Les fonctions *LTrim$, RTrim$* et *Trim$* (en anglais, le verbe *trim* signifie *couper, tailler*) ôtent les (éventuels) espaces situés respectivement à gauche, à droite, à la fois à gauche et à droite, d'une chaîne. A titre d'exemple, considérons les lignes suivantes

```
ch = "    Bonjour     "
ch1 = LTrim$ (ch)          ' ch1 = "Bonjour     "
ch2 = RTrim$ (ch)          ' ch2 = "    Bonjour"
ch3 = Trim$ (ch)           ' ch3 = "Bonjour"
```

7.6 *String$* et *Space$*

La fonction *String$ (nb, ascii)* retourne une chaîne de *nb* caractères dont le code *ASCII* est spécifié par *ascii*. A titre d'exemple, considérons les lignes suivantes

```
ch1 = String$ (4, "A")     ' ch1 = "AAAA"
ch1 = String$ (4, 66)      ' ch1 = "BBBB"
```

La fonction *Space$(nb)* retourne une chaîne composée de *nb* caractères d'espace.

7.7 Autres fonctions de traitement de chaînes

La fonction *Replace (source, ch_à_remplacer, ch_de_remplacement)* retourne une chaîne dans laquelle la sous-chaîne *ch_à_remplacer* a été remplacée plusieurs fois par la sous-chaîne *ch_de_remplacement*.

La fonction *StrReverse (ch)* retourne une chaîne contenant des caractères dont l'ordre a été inversé par rapport à la chaîne *ch*.

Les fonctions *UCase$ (ch)* et *LCase$ (ch)* convertissent tous les caractères de la chaîne *ch* respectivement en majuscules (en anglais, *upper-case letters*) et minuscules (en anglais, *lower-case letters*). Les caractères qui ne sont pas des lettres demeurent inchangés.

7.8 Fonctions de conversion

- ***Asc* et *Chr$***

La fonction *Asc (ch)* retourne un nombre entier correspondant au code *ASCII* du premier caractère de la chaîne *ch*. Réciproquement, la fonction *Chr$ (ascii)* retourne le caractère correspondant au code *ASCII* donné (nombre entier *ascii*).

- ***IsNumeric, Val, Str$, Hex$* et *Oct$***

La fonction *IsNumeric (ch)* retourne *True* si la chaîne *ch* peut représenter un nombre numérique.

La fonction *Val (ch)* convertit la chaîne *ch* en une valeur numérique (la lecture de la chaîne s'arrête au premier caractère ne faisant apparemment pas partie d'un nombre). A titre d'exemple, considérons les lignes suivantes

```
Dim ch1 As String, ch2 As String
Dim x As Integer, y As Double, z As Double
ch1 = "  2.35 a"
x = Val (ch1)          ' x = 2
y = Val (ch1)          ' y = 2.35
ch2 = "a12"
z = Val (ch2)          ' z = 0
```

La fonction *Str$ (valeur)* convertit une valeur numérique en chaîne.

Il est possible de convertir des valeurs numériques en chaîne sous une forme hexadécimale (base 16), ou octale (base 8), *via* les fonctions *Hex$* et *Oct$* respectivement.

Visual Basic fournit plusieurs autres fonctions permettant de convertir une chaîne en un autre type de donnée.

8 INTERFACE UTILISATEUR GRAPHIQUE : LES BASES

Les Interfaces Utilisateurs Graphiques (IUG) sont construites à partir de *contrôles*. Un *contrôle* est un objet (un objet s'utilise à travers ses propriétés, ses méthodes et ses événement associés) avec lequel un utilisateur interagit *via* la souris, ou le clavier. Les contrôles Visual Basic sont de deux types :

- les contrôles *intrinsèques*, aussi appelés contrôles *standards*, disponibles par défaut dans la barre de contrôles de l'EDI. Une description rapide de ces contrôles est donnée au § 2.1.
- les contrôles *ActiveX*. Ces contrôles sont mis à disposition dans la barre de contrôles, en ouvrant le menu *Projet* | *Composants*, puis en cliquant l'onglet nommé *Contrôles* et en cochant la case correspondant au contrôle *ActiveX* à insérer. A titre d'exemple, des contrôles *ActiveX* sont disponibles dans le cadre des bases de données, des réseaux, sachant que certains des contrôles sont spécifiques aux différentes versions (*Learning, Professional, Enterprise*) de Visual Basic.

Une description complète des contrôles disponibles dans Visual Basic est accessible dans l'aide en ligne (taper sur la touche F1). Décrivons ici quelques-uns des contrôles les plus utilisés, à travers une description de leurs propriétés, méthodes et événements associés.

8.1 Le contrôle *Label*

Les contrôles *label* (en français, étiquette) servent à afficher du texte à l'écran, en mode création ou exécution. Ce texte est non modifiable directement par l'utilisateur.

Ses propriétés
En dehors de la propriété *Name* (par défaut, *Label1, Label2, …*) qui permet de différencier les objets entre eux, la principale propriété du contrôle *Label* est *Caption* (notons que le texte par défaut correspond à celui mis par défaut dans la propriété *Name*) qui permet l'affichage (si nécessaire) d'un texte.
Comme pour un module feuille, les propriétés *BackColor* et *ForeColor* déterminent respectivement la couleur du fond et celle du texte affiché dans le contrôle *Label*.
La propriété *Font* permet de spécifier la police utilisée pour afficher le texte (nom, style (standard, italique, …), taille, …).
La propriété *Alignment* permet au choix de cadrer le texte affiché à droite, à gauche ou au centre.
La propriété *Enabled* permet d'activer, ou non, le contrôle lors de l'exécution de l'application.
La propriété *Visible* permet, par exemple en réponse à un événement, de cacher (*False*), ou non (*True*), le contrôle.

Ses méthodes
Nous retiendrons seulement la méthode *Move* qui permet le déplacement du *Label*.

Ses événements associés
Parmi les événements liés à la souris, on peut retenir les événements : *MouseDown, MouseUp, MouseMove, Click, DblClick*.

Remarques :
- Les événements *MouseDown* et *MouseUp* se produisent, dans l'ordre, lorsque l'utilisateur enfonce (*MouseDown*), ou relâche (*MouseUp*), un bouton de la souris.
- L'événement *MouseMove*, généré continuellement lorsque le pointeur de la souris se déplace sur des objets. La génération de cet événement s'intercale entre *MouseDown* et *MouseUp*.
- L'événement *Click* se produit lorsque l'utilisateur clique un bouton de la souris puis le relâche sur un objet. Lorsque l'on effectue un clic, la séquence d'événements générés est, dans l'ordre, *MouseDown, MouseUp, Click*.
- L'événement *DblClick* se produit lorsque l'utilisateur appuie sur, et relâche, un bouton de la souris deux fois de suite sur un objet. Lorsque l'on effectue un double clic, la séquence d'événements générés est, dans l'ordre, *MouseDown, MouseUp, Click, DblClick, MouseUp*.

8.2 Le contrôle *TextBox*

Le contrôle *TextBox* (en français, *zone de saisie*) permet, *via* une zone à l'écran, à un utilisateur l'introduction, ou l'affichage, d'informations. Le tableau ci-dessous regroupe les principales propriétés, méthodes et événement associés à ces contrôles.

Propriété, méthode, événement	Description
Propriété	
Enabled	Spécifie si l'utilisateur peut, ou non, agir sur le contrôle.
MaxLength	Spécifie le nombre maximal de caractères pouvant être entrés (la valeur 0, mise par défaut, indique qu'il n'y a pas de limite).
MultiLine	Permet quand elle est *True* d'entrer du texte sur plusieurs lignes.
PasswordChar	Spécifie le caractère affiché à la place des caractères entrés.
Text	Spécifie le texte entré.
Méthode	
SetFocus	Place le *focus* sur le contrôle.

Evénement	
Change	Evénement appelé à chaque fois que le contenu de la zone est modifié (par l'utilisateur ou par le programme).
Click, DblClick, MouseDown, MouseMove, MouseUp	Evénements liés à la souris.
GotFocus, LostFocus	Evénements générés lors de la réception, resp. de la perte, du *focus*.
KeyDown, KeyPress, KeyUp	Evénements liés au clavier (générés dans l'ordre : Touche pressée, appuyée, relachée).

8.3 Le contrôle *CommandButton*

Les contrôles *CommandButton* sont représentés par des *boutons*, appelés aussi " boutons-poussoirs ", qu'un utilisateur peut cliquer pour exécuter une action.
La propriété *Caption* permet d'écrire un texte sur le bouton. La propriété *Enabled* indique si le bouton est actif (*True*), ou non (*False*). Suite à un clic sur un bouton (actif), l'événement *Click* est appelé.

8.4 Les contrôles *ListBox, ComboBox*

Les listes permettent de visualiser une liste de différents items (éléments). Pour réaliser des listes, Visual Basic propose deux types de contrôles : *ListBox* (liste) ou *ComboBox* (liste combinée, ce contrôle permet de combiner une liste avec une zone de saisie).

- **Le contrôle *ListBox***

Quand une liste (*ListBox*) contient plus d'items qu'elle ne peut en afficher, une barre de défilement (en anglais, *scrollbar*) verticale apparaît automatiquement.

Propriété, méthode, événement	**Description**
Propriété	
Columns	(entier) Spécifie si le contrôle a une barre de défilement horizontal, si oui, spécifie le nombre de colonnes. Une valeur égale à 0 indique qu'il n'y a pas de défilement horizontal, une valeur supérieure à 0 spécifie le nombre de colonnes dans lesquelles les items sont listés horizontalement.
Enabled	(booléen) Spécifie si l'utilisateur peut, ou non, agir sur le contrôle.
List	(tableau de *String*) Contient les éléments de la liste, le plus petit indice du tableau est égal à 0, le plus grand indice est égal à *ListCount* - 1.
ListCount	(entier) Contient le nombre d'éléments de la liste.
ListIndex	(entier de -1 à *ListCount* - 1) (pour les listes à sélection simple) Contient l'indice de l'élément actuellement sélectionné. La valeur - 1 signifie qu'aucun élément n'est sélectionné.
MultiSelect	(entier de 0 à 2) Spécifie si l'utilisateur peut sélectionner plus d'un item à la fois. Une valeur égale à : - 0 autorise la sélection d'un seul élément à la fois (sélection simple), - 1 autorise plusieurs choix, chaque clic sélectionne/désélectionne un item (sélection multiple), - 2 autorise un choix étendu d'items – possibilité d'utiliser les touches *shift* et *contrôle* (sélection multiple étendu).
Selected	(tableau de booléens) (pour les listes à sélection multiple) L'élément d'indice *i* est actuellement sélectionné si *Selected* (*i*) est *True*.

SelCount	(entier) (pour les listes à sélection multiple) Indique le nombre d'éléments sélectionnés.
Sorted	(booléen) Permet d'avoir une liste dont les éléments sont triés par ordre alphabétique.
Text	(*String*) Spécifie l'élément sélectionné, correspond à *List* (*ListIndex*).
Méthode	
AddItem	*AddItem item* [, *index*] permet d'ajouter (en fin de liste) l'élément *item* dans la liste. Le paramètre optionnel *index* permet d'insérer l'élément *item* dans la liste à l'indice *index*.
Clear	Enlève tous les éléments de la liste.
RemoveItem	*RemoveItem index* permet d'enlever l'élément d'indice *index* de la liste.
Evénement	
Click	Evénement activé à chaque fois qu'un élément de la liste est sélectionné.

- **Le contrôle *ComboBox***

La liste combinée permet l'affichage d'une liste (à sélection simple) et d'une zone de saisie. Elle permet à l'utilisateur de sélectionner un élément dans une liste, ou d'entrer une donnée dans une zone de saisie.

Il existe trois types de listes combinées, définis par la valeur (entière) de la propriété *Style* :

- 0 (*VbComboDropDown*) (Valeur par défaut.) *Liste déroulante modifiable*. Comprend une zone de saisie avec, à côté, une flèche déroulante (*drop down*) permettant de faire apparaître une liste, dite déroulante. L'utilisateur peut sélectionner une option dans la liste, ou taper ce qui convient dans la zone de texte.
- 1 (*VbComboSimple*) *Liste modifiable simple*. Comprend une zone de texte et une liste non déroulante (toujours visible). L'utilisateur peut sélectionner une option de la liste, ou taper ce qui convient dans la zone de texte. La taille d'une liste modifiable simple inclut les parties texte et liste. Par défaut, une liste modifiable simple est dimensionnée de sorte que la liste ne s'affiche pas. Augmenter la valeur de la propriété *Height* pour afficher une plus grande partie de la liste.
- 2 (*VbComboDrop-DownList*) *Liste déroulante* (sans saisie possible). Ce type de liste permet seulement à l'utilisateur de sélectionner une option dans la liste déroulante (semblable à 0 mais la saisie est interdite).

On retrouve des propriétés et méthodes similaires à celles des contrôles *TextBox*.

8.5 Les contrôles *Frame, CheckBox, OptionButton*

Les contrôles *Frame* (en français, cadre) permettent de regrouper (souvent de manière fonctionnelle) plusieurs contrôles dans un cadre afin d'en faire un *groupe* identifiable. Un tel contrôle est souvent couplé aux cases à option. La propriété *Caption* permet de donner un intitulé au cadre, les propriétés *Enabled* et *Visible* permettent de rendre le cadre respectivement inactif (de même pour les contrôles qu'il contient), et non visible - caché - (de même pour les contrôles qu'il contient).

Les contrôles *CheckBox* (en français, case à cocher) peuvent être sélectionnés, ou non. Ils sont habituellement utilisés pour exprimer des attributs optionnels, par exemple l'attribut *Marié* d'une personne. La propriété *Value* permet de savoir si une case est désélectionnée - non cochée - (0, *vbUnChecked*), ou sélectionnée (1, *vbChecked*), ou indisponible - la case est alors ombrée - (2, *vbGrayed*) (cette valeur ne peut être fixée que par programmation).
L'événement *Click* est appelé lorsqu'une case à cocher est sélectionnée, ou désélectionnée.

Les contrôles *OptionButton* (en français, case à option) fonctionnent sensiblement selon le même principe, mais par *groupe*. Dans un même *groupe*, seule une case à option peut être sélectionnée à la fois, i.e., les cases à option d'un même groupe sont exclusives les unes par rapport aux autres. Les cases à option forment un groupe lorsqu'elles sont dans le même *cadre*.
La propriété *Value* permet de savoir si une case à option est sélectionnée - pointée - (*True*), ou désélectionnée (*False*).
L'événement *Click* est appelé lorsqu'une case à option est sélectionnée, ou désélectionnée.

8.6 Les menus

Visual Basic fournit un moyen simple de créer des menus *via* l'*éditeur de menus* (*Menu Editor*), voir le menu *Outils | Créateur de menus*. L'*éditeur de menus* est, en fait, une manière d'affecter les propriétés d'un menu. Une fois un menu créé, ses propriétés et ses événements associés sont visibles dans les fenêtres *Propriétés* et *Code*.

La boîte de dialogue de l'*éditeur de menus* contient les zones de saisies *Caption* et *Name*, pour indiquer respectivement le nom du menu visible par l'utilisateur (par exemple, pour entrer le menu *Fichier*, entrer *&Fichier*), et le nom de la variable utilisée par le programmeur (par exemple, *mnuFichier*). Il est bien sûr possible de créer :

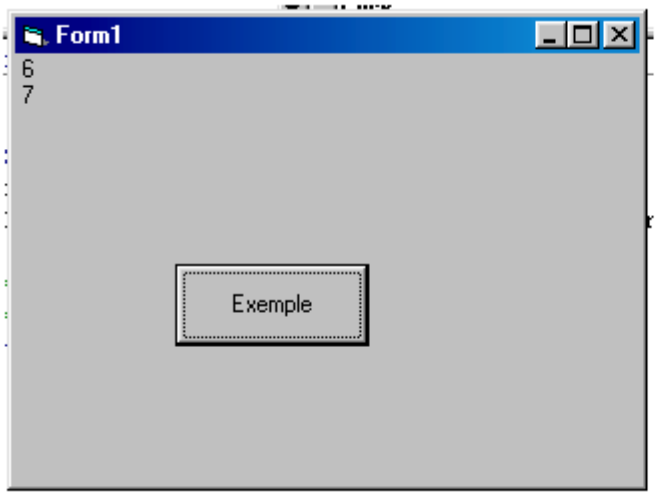
- des menus déroulants (listes déroulantes d'options qui n'apparaissent qu'à la suite d'un clic sur un titre de menu) ;
- des menus imbriqués (cinq niveaux de retrait au maximum).

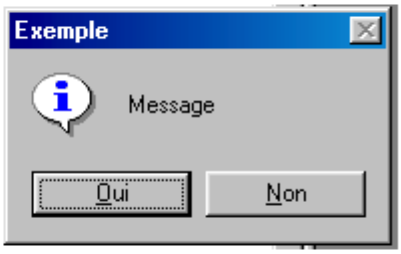
8.7 La fonction *MsgBox*

La fonction *MsgBox* (boîte de message) permet d'afficher, de manière standard sous Windows, une boîte de dialogue fournissant un message à destination de l'utilisateur à propos de l'état d'exécution du programme. La boîte de message *MsgBox* peut, selon sa configuration, afficher un message (un texte), un icône de même que des boutons (ces derniers permettant de fournir des informations au programme).

A titre d'exemple, le programme suivant permet - par un appui sur un bouton de commande - de faire apparaître une boîte de message (voir la figure qui suit). Selon la réponse (appui sur bouton *Yes* ou *No*) de la boîte de message, le contenu de la variable en retour de la fonction *MsgBox* est affiché dans la fenêtre.

```
Option Explicit

Private Sub CmdExemple_Click()
   Dim r As Integer
   r = MsgBox("Message", vbYesNo + vbInformation + vbApplicationModal, _
              "Exemple")
   ' vbYesNo permet l'affichage des boutons Yes et No
   ' vbInformation permet l'affichage de l'icône information
   ' vbApplicationModal indique que la boîte de message est modale (i.e., l'utilisateur
   ' ne peut pas interagir sur une fenêtre tant que la boîte de message n'est pas fermée)
   ' r = 6 (vbYes) si le bouton Yes a été pressé
   ' r = 7 (vbNo) si le bouton No a été pressé
   Print r
End Sub
```


Form1
6
7
Exemple


Exemple
Message
Oui
Non