

SOMMAIRE

I- INTRODUCTION :	3
II- L'ENVIRONNEMENT DE DEVELOPPEMENT DE VISUAL BASIC :	3
1- ELEMENTS DE L'ENVIRONNEMENT DE DEVELOPPEMENT INTEGRE VISUAL BASIC (IDE) :	4
1- Barre de menus :	4
2- Barres d'outils :	4
3- Boîte à outils :	4
2- LES ETAPES DE CREATION D'UNE APPLICATION EN VISUAL BASIC :	5
3- LES STRUCTURES DE BASE DU VISUAL BASIC :	5
4- LES VARIABLES :	5
1- Types de variables :	6
2- Les déclarations :	7
3- Portée des variables :	7
4- Les constantes :	8
5- LES TABLEAUX :	8
6- LES OPERATEURS :	9
III- LES INSTRUCTIONS DE CONTROLE :	9
1- AFFECTION :	9
2- IF ... THEN ... ELSE :	10
3- GoTo :	10
4- FOR ... NEXT :	10
5- Do ... LOOP :	11
While ... Wend :	12
Select Case :	13
IV- SOUS-PROGRAMMES :	13
1- APPEL INTERNE – GoSUB ... RETURN :	14
2- APPEL EXTERNE – PROCEDURES ET FONCTIONS :	14
V- TRAITEMENT DES CHAINES DE CARACTERES :	17
1- BOITE DE SAISIE :	18
2- MSGBOX :	19
VI- CREATION D'UNE INTERFACE UTILISATEUR :	20
1- CARACTERISTIQUES DES FEUILLES MDI EN MODE EXECUTION :	21
2- CREATEURS DE MENUS :	21
3- ORGANISATION DES FEUILLES FILLES :	22
4- LES MENUS CONTEXTUELS :	22
LES CONTROLES DE VISUAL BASIC :	23
CATEGORIES DE CONTROLES :	23
Les groupes de contrôles :	23

VII- LES FICHIERS.....	32
<u>1- MODES DE TRAITEMENT DES FICHIERS :</u>	33
<u>1) Accès Séquentiel :</u>	33
<u>2) Accès direct ou accès aléatoire :</u>	34
<u>3) Accès binaire :</u>	35
<u>2- LES INSTRUCTIONS DE TRAITEMENT DES FICHIERS :</u>	35
<u>1) Open :</u>	35
<u>2) Line Input :</u>	36
<u>3) Input () :</u>	36
<u>4) Input #:</u>	37
<u>5) Print #:</u>	38
<u>6) Write #:</u>	38
<u>7) Close :</u>	39
<u>8) Freefile :</u>	39
<u>9) Get :</u>	39
<u>10) Put :</u>	40
<u>11) Seek :</u>	41
<u>12) Loc :</u>	42
<u>13) EOF :</u>	43
<u>14) LOF :</u>	43
<u>15) FileLen :</u>	44



Un langage de programmation, c'est un outil qui sert à écrire des programmes répondant à l'attente de l'utilisateur de manière à les exécuter à l'aide d'instructions propre à chaque langage.

Par leur façon de communiquer, on définit deux types de langage :

- Le langage à bas niveau : Langage qui communique directement avec le matériel (Assembleur)
- Le langage à haut niveau : Basic, Pascal, Fortran.

On les différencie aussi par leurs aspects, c'est à dire, le domaine d'utilisation :

- 1- **Aspect scientifique** : C, Pascal, PLM, Fortran ...
- 2- **Gestion** : (SGBD : Dbase, Clipper, Foxpro), Langage COBOL
- 3- **Langage Orienté Objet** : Visual C++, C++, Small Talk.

I- Introduction :

Visual Basic est l'outil le plus facile à utiliser pour créer des applications Microsoft Windows. Visual Basic se compose des deux mots Visual et Basic.

Le mot « Visual » fait référence à la méthode utilisée pour créer l'interface graphique utilisateur (GUI, Graphical User Interface). Au lieu de rédiger de multiples lignes de code pour décrire l'apparence et l'emplacement des éléments d'interface, il vous suffit de glisser – déplacer des objets prédéfinis à l'endroit adéquat sur l'écran.

Le mot « Basic » fait référence au langage BASIC (Beginners All-Purpose Symbolic Instruction Code), langage le plus utilisé par les programmeurs depuis les débuts de l'informatique. Visual Basic constitue une évolution par rapport au langage BASIC initial et comporte aujourd'hui plusieurs centaines d'instructions de fonctions et de mots clés, dont un grand nombre font directement référence à l'interface graphique utilisateur (GUI) de Windows.

II- L'environnement de développement de Visual Basic :

L'environnement de travail de Visual Basic est souvent désigné sous le nom de « environnement de développement intégré » (IDE, Integrated Development Environment), car il intègre de nombreuses fonctions variées telles que la création, la modification, la compilation et le débogage au sein du même environnement.

Dans la plupart des outils de développement traditionnels, chacune de ces fonctions est exécutée par un programme distinct doté de sa propre interface.

1- Eléments de l'environnement de développement intégré Visual Basic (IDE) :**1- Barre de menus :**

Affiche les commandes qui vous permettent d'utiliser Visual Basic. Outre menus habituels, à savoir Fichier, Edition, Affichage, Fenêtre et ? (Aide), des menus permettent d'accéder à des fonctions spécifiques nécessaires à la programmation, notamment les menus Projet, Format et Débogage.

2- Barres d'outils :

Contiennent des boutons constituant des raccourcis vers des éléments de menu fréquemment utilisés. Vous pouvez cliquer sur un bouton de la barre d'outils pour exécuter l'action qu'il représente. Par défaut, la barre d'outils Standard apparaît lorsque vous démarrez Visual Basic.

3- Boîte à outils :

Fournit un ensemble d'outils nécessaires au moment de la création pour disposer les contrôles sur une feuille.

Icône	Nom du contrôle	Nom de la classe	Description
	Zone d'image	PictureBox	Affiche une image, effectue des sorties graphiques ou peut servir de cadre à d'autres contrôles.
	Etiquette	Label	<i>L'étiquette</i> pour afficher du texte, sans permettre une saisie de l'utilisateur.
	Zone de Texte	TextBox	Pour saisir du texte, sur une ou plusieurs lignes.
	Cadre	Frame	Peut regrouper d'autres contrôles (notamment des boutons d'option), ou être utilisé à des fins de décoration.
	Bouton de commande	CommandButton	<i>Le bouton de commande</i> qui permet la saisie de commande de l'utilisateur.
	Case à cocher	Check Box	<i>La case à cocher</i> crée une case que l'utilisateur peut cocher facilement pour indiquer si un état est vrai ou faux, ou pour afficher des choix multiples lorsque l'utilisateur peut en sélectionner plusieurs.
	Bouton d'option	OptionButton	<i>Le bouton d'option</i> utilisé en groupe pour autoriser le choix d'une option ou de plusieurs possibles.
	Zone de liste	ListBox	<i>Zone de liste</i> affiche une liste d'éléments parmi lesquels l'utilisateur peut faire un choix.
	Liste modifiable	ComboBox	<i>Liste Modifiable</i> allie les fonctions d'une zone de texte et d'une zone de liste. Permet à l'utilisateur de taper dans un élément sélectionné ou d'en sélectionner un dans une liste déroulante.
	Liste de répertoires	DirListBox	<i>Liste de répertoires</i> affiche une liste de dossiers et de chemins d'accès parmi lesquels l'utilisateur peut faire un choix.
	Liste de lecteurs	DriveListBox	<i>Liste de lecteurs</i> affiche une liste de lecteurs de disque valides parmi lesquels l'utilisateur peut faire un choix.
	Liste de fichiers	FileListBox	<i>Liste de fichier</i> affiche une liste de fichiers parmi lesquels l'utilisateur peut faire un choix.
	Barres de	HScrollBar et	<i>Barres de défilement horizontal et vertical</i>

	défilement horizontale et vertical	VscrollBar	permettent à un utilisateur d'ajouter des barres de défilement dans des contrôles qui ne les fournissent pas automatiquement. Ces contrôles sont différents des barres de défilement intégrées dans de nombreux contrôles.
	Forme	Shape	<i>Forme</i> Ajoute un rectangle, un carré, une ellipse ou un cercle sur une feuille, un cadre ou une zone d'image.
	Ligne	Line	<i>Ligne</i> Ajoute un segment de droite sur une feuille
	Minuterie	Timer	<i>Minuterie</i> Exécute des événements Timer à des intervalles définis.
	Image	Image	Affiche des images bitmap, des icônes, ou encore des fichiers JPEG ou GIF. Agit comme un bouton de commande lorsque l'utilisateur clique dessus.
	Conteur OLE	OLE	Incorpore des données dans une application Visual Basic.
	Données	Data	Permet de se connecter à une base de données existante et d'afficher les informations qu'elle contient sur vos feuilles.

2- Les étapes de création d'une application en Visual Basic :

La création d'une application en Visual Basic comprend trois étapes :

- Création de l'interface (icônes, fenêtres ...)
- Définition de propriétés (légende de boutons, couleur des fenêtres, taille type d'images ou d'icônes ...)
- L'écriture du code.

3- Les structures de base du Visual Basic :

Un programme en Visual Basic est une suite d'instructions qui peuvent soit s'enchaîner *Séquence d'instructions* soit s'exécuter dans certains cas et pas dans d'autres on parle alors de *Structures alternatives* ou se répéter plusieurs fois *Structures répétitives*.

Comme tout langage de programmation, VB fournit une liste d'instructions qui sont implémentées (propres à lui) et que l'on peut donc utiliser sans les réécrire.

4- Les variables :

Le code d'une application VB se déroule en réponse à des événements. Les instructions sont constituées de *verbes* qui agissent sur des *variables*. Une variable peut être considérée comme une case en mémoire. Elle est identifiée par un *nom*, permet de la désigner, et son contenu est appelé *valeur*.

Le nom d'une variable est choisi par celui qui écrit le programme, en respectant quelques règles :

- Il est constitué de lettres, de chiffres et du caractère ' _ ' (souligné)
- Le premier caractère est obligatoirement une lettre.
- Les lettres majuscules et minuscules sont équivalentes, et les caractères accentués sont acceptés.
- Il ne peut y avoir plus de 40 caractères dans un nom de variable.
- Les mots réservés de Visual Basic (tels que Loop, If, Constant ...) ne peuvent être utilisés.

Exemple :

I
J18
Compteur
Bénéfice_de_l_année_1999
BénéficedeLannée1999

1- Types de variables :

Visual Basic permet l'utilisation de divers types de variables qui peuvent être regroupées dans les catégories suivantes :

Les variables numériques utilisées pour stocker un nombre. Il en existe plusieurs catégories :

Les entiers sur 2 octets (Integer) ou 4 octets (Long) qui peuvent stocker un nombre entier, sans décimale, positif ou négatif. Les premiers sont plus économiques, mais ne peuvent être utilisés pour les grands nombres.

Les nombres à virgule flottante sur 4 octets (single) ou 8 octets (Double) utilisés pour stocker un nombre réel quelconque.

Les unités monétaires qui sont des nombres à virgule fixe ayant 4 décimales (currency).

Les chaînes de caractères (string) utilisées pour stocker des textes, des libellés ... Une seule chaîne de caractère ou une date/heure. Visual Basic effectue automatiquement les conversions selon les besoins. Il s'agit du type de données par défaut, décrit ci-après.

Les types spécifiques (Type) principalement utilisés pour la déclaration des enregistrements d'un fichier.

Suffixe	Type de données	Taille	Plage
%	Integer (entier simple)	2 octets	-32768 à 32767
&	Long (entier long)	4 octets	-2 147 483 648 à 2 147 483 647
!	Single simple précision virgule flottante	8 octets	-3,402823E38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E38 pour les valeurs positives
#	Double (valeur à virgule flottante à double précision)	8 octets	-1,797693134862315E308 à -4,94066E-324 pour les valeurs négatives ; 4,94066E-324 à 1,797693134862315E308 pour les valeurs positifs
@	Currency (entier virgule fixe 4 chiffres décimaux)	8 octets	-922337203685477,5808 à 922337203685477,5807
\$	String	1 octet par caractère	0 à approximativement 65535 octets
Aucun	Variant	Variable	Toute valeur numérique ou chaîne de caractères
Aucun	Défini par l'utilisateur (avec type)	Nombre d'octets requis par les éléments	

Le type de données par défaut est variant : c'est le type de donnée attribuer à toutes les variables non déclarées, il peut contenir des données numériques des chaînes de caractères, des dates ...

2- Les déclarations :

Le langage Basic présente une originalité par rapport à de nombreux autres langages : il autorise l'utilisation de variables sans imposer une déclaration au préalable. Ainsi, la simple utilisation d'une variable dans une ligne de code génère une déclaration implicite. Mais il est également possible de déclarer explicitement les variables avant leur utilisation.

La déclaration explicite d'une variable peut se faire n'importe où dans le code du programme : dans la section des déclarations d'une feuille ou d'un module, ou dans le corps d'une fonction ou d'une procédure. Elle se fait à l'aide de l'instruction **DIM** ou **STATIC** dans une feuille ou un module, ou **GLOBAL** dans la section des déclarations d'un module.

Chaque fois que l'instruction DIM est exécutée, la variable correspondante est réinitialisée à 0 s'il s'agit d'une variable numérique, à la chaîne vide (une chaîne de caractères sans caractère) s'il s'agit d'une variable **String**, ou à la valeur particulière **Empty** s'il s'agit d'une variable **Variant**. Si vous souhaitez qu'une variable garde sa valeur de façon permanente, il faut remplacer le mot **Dim** par **Static**. Dans ce cas, la variable est initialisée aux valeurs précédentes lors du lancement de l'application, et ensuite le seul moyen d'en modifier la valeur est d'utiliser une instruction d'affectation.

```
Sub Maproc ()  
Dim i as integer  
    i = i + 1  
End Sub
```

A chaque appel de la procédure **MaProc**, la variable i est initialisée à 0 par contre si on écrit :

```
Sub Maproc ()  
Static i as integer  
    i = i + 1  
End Sub
```

i est initialisée à 0 lors du lancement de l'application, puis garde sa valeur précédente d'un appel à l'autre, c'est à dire qu'elle est incrémentée d'une unité à chaque appel.

Pour imposer la déclaration explicite des variables, il suffit de placer, dans la section des déclarations de chaque feuille ou module, la ligne suivante : **Option Explicit**.

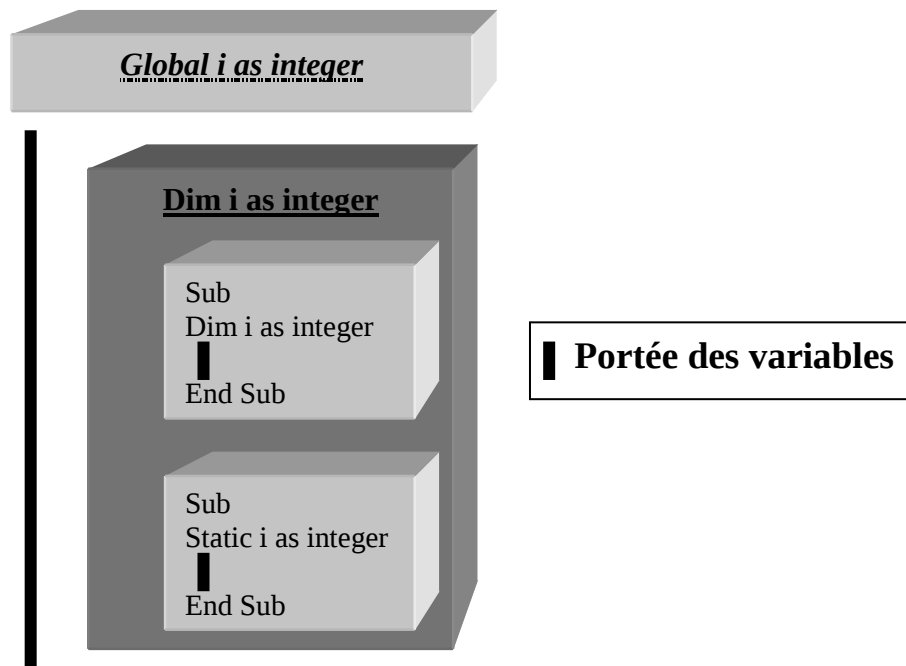
3- Portée des variables :

Si une variable est utilisée sans avoir été préalablement déclarée, elle n'est connue qu'à l'intérieur de la fonction ou de la procédure où elle est initialement utilisée. Si vous faites référence au même nom de variable dans une autre procédure, il s'agit en fait d'une variable différente.

Si la variable est déclarée dans une fonction ou une procédure, avec *Dim* ou *Static*, elle est dite locale à cette fonction ou procédure, et n'est pas connue à l'extérieur.

Si elle est déclarée dans la section des déclarations d'une feuille ou d'un module avec *Dim* elle est connue dans toutes les procédures de la feuille ou du module mais pas dans les autres.

Si enfin elle est déclarée dans la section des déclarations d'un module avec l'instruction *Global* au lieu de *Dim*, elle peut être utilisée dans toutes les feuilles et tous les modules. Il s'agit d'une variable **globale**.



4- Les constantes :

Une constante ressemble à une variable dans la mesure où elle correspond également à une case en mémoire. Elle s'en distingue cependant par le fait que son contenu est défini initialement et ne peut pas être modifié par le programme.

La déclaration d'une constante se fait à l'aide de l'instruction **Const** dans une feuille ou un module, et **Global Const** dans un module, pour que sa portée soit globale dans l'application.

Const Mai = 5

Global Const Pi = 3,14

5- Les tableaux :

On trouve souvent, dans un programme, un type de données qui doit se répéter avec des valeurs différentes. Par exemple, une application de comptabilité peut être amenée à calculer des valeurs pour chacun des mois d'une année. On peut alors utiliser une seule variable et faire appel à un indice pour cliquer chacun des mois, il s'agit d'un tableau. Pour le déclarer, on note :

Dim Montant (1 to 12) As Single

Dim Mois As Integer

Mois = 5

Montant (Mois) = 192.8

L'indice de la variable Montant peut varier entre 1 et 12 (les 12 mois de l'année). la dernière ligne désigne le montant du mois de mai. Si de plus on souhaite travailler sur cinq années, on peut définir un tableau à deux dimensions :

Dim Montant (1985 to 1990, 1 to 12) As Single

Dim Mois As Integer, Année As Integer

Année = 1987

Mois = 5

Montant (Année, Mois) = 192.8

Remarques :

Il peut y avoir jusqu'à 60 dimensions.

Les tableaux peuvent être d'un type de données quelconque.

Visual Basic autorise également la déclaration de tableaux dynamiques. Pour cela, il faut initialement créer le tableaux sans spécifier de dimensions, par exemple :

Dim Année ()

Avant d'utiliser le tableau, il convient de fixer des dimensions à l'aide de l'instruction **ReDim**, par exemple :

ReDim Année (1985 to 1990)

On peut utiliser plusieurs fois cette instruction pour changer les dimensions d'un tableau. Mais, chaque fois, le contenu du tableau est réinitialisé. Pour éviter cela, il faut utiliser le mt clé **Preserve** :

ReDim Preserve Années (1985 to 1993)

Un tableau dynamique ou non, peut être réinitialisé à l'aide de l'instruction **Erase** qui réinitialise les éléments de tableaux statiques et libère la mémoire affectée aux tableaux dynamique.

Les fonctions **LBound** et **UBound** retournent respectivement le plus petit ou le plus grand indice d'un tableau, dans une dimension donnée. Par exemple :

Dim Montant (1985 to 1990, 1 to 12)

Lbound (Montant, 1) 'Retourne 1985

Lbound (Montant, 2) 'Retourne 1

Ubound (Montant, 1) 'Retourne 1990

Ubound (Montant, 2) 'Retourne 12

6- Les opérateurs :

Lorsqu'une expression renferme plusieurs opérations. Chaque partie est évaluée et résolue dans un ordre prédéterminé ; cet ordre est connu sous le nom de priorité des opérateurs. Quant des expressions contiennent des opérateurs de différentes catégories, les opérateurs arithmétiques sont évalués au premier suivi des opérateurs de comparaison puis des opérateurs logiques. Au sein de ces divers catégories, les opérateurs sont évalués dans l'ordre de priorité ci-dessous :

Opérateurs arithmétiques	Opérateurs de comparaison	Opérateurs logiques
Evaluation à une puissance (^)	=	Not
Négation (-)	<>	And
Multiplication et division (*, /)	<	Or
Division d'entier (\)	>	Xor
Arithmétique modulo (mod)	<=	
Concaténation de chaînes (&)	>=	

III- Les instructions de contrôle :

Le code d'un programme Visual Basic est constitué d'instructions qui peuvent être des déclarations ou bine des instructions plus actives.

1- Affectation :

Une des actions les plus couramment effectuées dans un programme est l'affectation de valeur, qui consiste à donner à une variable la valeur d'une autre variable, d'une constante ou d'une fonction. L'affectation s'écrit simplement à l'aide du signe égal « = » : le membre situé à gauche du signe prend la valeur de celui situé à droite ; (on peut également ajouter le

mot clé **Let** devant l'instruction). Les deux membres doivent donc être de même type, c'est-à-dire soit numérique, soit chaîne de caractères.

2- If ... Then ... Else :

Si une condition est vraie alors faire ceci sinon cela. L'instruction IF ... Then ... Else prend deux formes principales. Sur une seule ligne, elle s'écrit :

If condition **Then** instructions [**Else** instructions]

Et sur plusieurs lignes :

If condition 1 **Then**

[bloc-instructions-1]

ElseIf condition 2 **Then**

[bloc-instructions-2]

Else

[bloc-instructions-n]

End If

Exemple :

If a > b then Max = a else Max = b

If Not a <= b then Max = a else Max = b

If i = 1 then

 'Instructions si i vaut 1

ElseIf i = 2 then

 'Instructions si i vaut 2

ElseIf i = 3 then

 'Instructions si i vaut 3

End If

3- GoTo :

A côté de l'affectation et du test, le débranchement inconditionnel faisait partie de la panoplie d'instructions des premiers programmeurs. Le GoTo existe toujours dans Visual Basic, mais on peut la plupart du temps s'en passer grâce aux autres instructions de contrôle qui permettent d'obtenir des programmes plus faciles à lire.

L'instruction GoTo permet de transférer l'exécution du programme en un autre point, identifié par une étiquette (on peut également utiliser des numéros de ligne). Une étiquette est un nom (comme un nom de variable) suivi par « : » la syntaxe est :

GoTo {étiquette-de-ligne | numéro-de-ligne}

Exemple :

GoTo Erreur

 ' instructions

 ' instructions

Erreur : 'Etiquette

 ' instructions

4- For ... Next :

L'instruction For ... Next permet d'exécuter un bloc d'instructions plusieurs fois en utilisant une variable numérique comme compteur. La syntaxe est la suivante :

For compteur = début **To** fin [**Step** pas]

 [bloc-d'instructions]

 [Exit For]

```
    [bloc-d'instructions]
Next [compteur]
```

Exemple :

```
Dim montant (1 to 12) as Single
Dim total as Single
Dim mois as Integer
Total = 0
For mois = 1 To 12
    Total = Total + montant (mois)
Next
```

Le fonctionnement de cet exemple est le suivant :

1. **Instruction For** : donner à Mois la valeur 1
2. **Exécuter les instructions entre For et Next**
3. **Instruction Next** : si Mois est supérieur ou égal à 12, passer à l'instruction suivante (fin de boucle) sinon retourner au point 2.

La valeur ajoutée à chaque boucle est 1 par défaut. On peut la modifier à l'aide du mot clé Step. L'exemple suivant donne le même résultat que le précédent, mais le calcul est effectué en commençant par le dernier mois. La valeur - 1 est ajoutée à chaque itération, ce qui équivaut à dire que 1 est retiré :

```
For mois = 12 To 1 Step - 1
    Total = Total + montant (mois)
Next
Plusieurs boucles peuvent être imbriquées.
```

Exemple :

```
Dim tableau (5, 10) as integer
Dim i = 0 as Integer
For i = 0 to 5
    For j = 0 to 10
        Tableau (i, j) = 1
    Next j
Next i
```

On peut également quitter la boucle avant la fin grâce à **Exit For**. La boucle suivante se termine quand les 12 éléments sont initialisés ou quand le total dépasse la valeur 10000 :

```
For Mois = 1 to 12
    Total = Total + Montant (Mois)
    If Total > 10000 Then Exit For
Next
```

5- Do ... Loop :

L'instruction Do ... Loop permet d'exécuter un bloc d'instructions dans une boucle mais tant qu'une condition est vraie, ou jusqu'à ce qu'une condition devient vraie. Il y a deux façons d'utiliser cette instruction :

```
Do [{While | Until} condition]
    [Bloc-d'instructions]
[Exit Do]
[Bloc-d'instructions]
```

Loop

Ou

Do

[Bloc-d'instructions]

[Exit Do]

[Bloc-d'instructions]

Loop {While | Until} Condition]

Exemple :

Total = 0

Mois = 0

Do While Total < 10000 and mois <= 12

Total = Total + montant (Mois)

Mois = Mois + 1

Loop

Le fonctionnement on est le suivant :

1. Si la condition est fausse, aller après Loop
2. Sinon exécuter les instructions jusqu'à Loop
3. Retourner à 1

On pourrait également écrire :

Total = 0

Mois = 0

Do Until Total >= 10000 or Mois > 12

Total = Total + Montant (Mois)

Mois = Mois + 1

Loop

La seconde forme est un peu différente dans le sens où au moins une boucle est toujours effectuée, puisque la condition n'est testée qu'en fin de boucle. Par exemple :

Do

Total = Total + montant (mois)

Mois = Mois + 1

Loop While Total < 10000 and Mois <= 12

Le fonctionnement est le suivant :

1. Exécuter les instructions
2. Si la condition est vraie retourner en 1, sinon terminer la boucle.

Comme pour l'instruction For, il est possible d'interrompre la boucle en dehors du test à l'aide de Exit Do qui provoque un débranchement immédiat à l'instruction qui suit GloopG.

While ... Wend :

L'instruction While ... Wend permet d'exécuter un bloc d'instructions dans une boucle tant qu'une condition est vraie. Elle peut toujours être remplacée par un bloc Do While ... Loop. La syntaxe est :

While condition

[bloc-d'instructions]

Wend

L'exemple précédent peut s'écrire :

While Total < 10000 and mois <= 12

Total = Total + Montant (Mois)

Mois = Mois + 1

Wend

Select Case :

L'instruction **Select ... Case** permet de sélectionner le bloc d'instructions à exécuter selon la valeur d'une variable de contrôle. La syntaxe est :

```
Select Case expression-testée
    [Case liste-d'expression 1]
        [bloc-d'instructions-1]
    [Case liste-d'expression 2]
        [Bloc-d'instructions-2]
    [Case Else]
        [bloc-d'instructions-n]
```

End Select

L'expression-testée peut être une expression numérique ou une chaîne de caractères. Les liste-d'expressions peuvent être des suites de valeurs séparées par des virgules, des intervalles du type expression to expression, ou un opérateur relationnel introduit par Is. Par exemple :

```
Select Case valeur
    Case 0
        'instructions-1
    Case 1, 5, 6
        'instructions-2
    Case 2 to 4
        'instructions-3
    Case Is > = 7
        'instructions-4
    Case Else
        'instructions-5
```

End Select

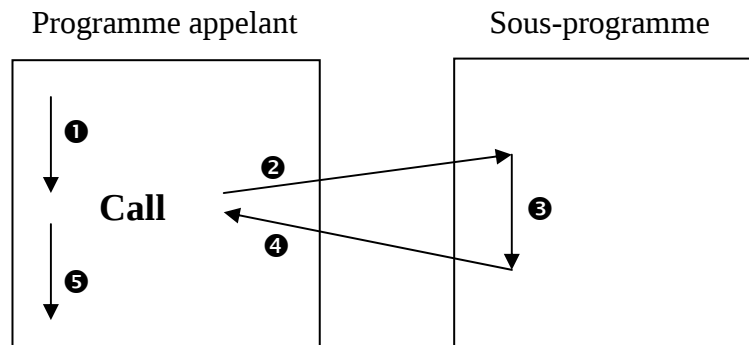
Cela peut se lire de la façon suivante :

1. Si *Valeur* est 0, exécuter les *instructions-1* puis aller à l'instruction qui suit End Select.
2. Si *Valeur* est 1, 5 ou 6, exécuter les *instructions-2* puis aller à l'instruction qui suit End Select.
3. Si *Valeur* est 2 et 4, exécuter les *instructions-3* puis aller à l'instruction qui suit End Select.
4. Si *Valeur* est supérieur ou égale à 7, exécuter les *instructions-4* puis aller à l'instruction qui suit End Select.
5. Dans les autres cas, exécuter les *instructions-5* puis aller à l'instructions qui suit End Select.

Une instruction Select Cas peut souvent remplacer avantageusement une suite IF ... Elseif, tout en rendant la lecture du programme plus aisée.

IV- Sous-programmes :

L'utilisation de ce qui peut être appelé de façon générale des sous-programmes permet de rendre la programmation d'une application plus structurée. Un sous-programme est un bloc d'instructions qui peuvent être exécutées à partir d'autres parties du programme. L'appel du sous-programme (**Call**) provoque un débranchement vers sa première instruction (comme un **GoGo**) mais l'adresse de l'instruction ayant provoqué l'appel est mémorisée. A la fin du sous-programme, le débranchement se fait vers cette adresse, c'est le retour (**return**).



Visual Basic comprend deux types d'appels de sous-programmes : les appels de l'intérieur d'une procédure et les appels d'une procédure ou d'une fonction.

1- Appel interne – GoSub ... Return :

L'instruction **GoSub** ressemble fortement à un **GoTo**. Comme elle, elle provoque un branchement à l'étiquette indiquée. Mais en plus, l'adresse de retour (instruction suivant le **GoSub**) est mémorisée, et la rencontre d'une instruction **Return** conduit au retour à cette instruction. La syntaxe est :

GoSub {étiquette-de-ligne | numéro-de-ligne}

Exemple :

GoSub Calcul

 'Instructions-1

Exit Sub

Calcul :

 'Instructions-2

Return

Lorsque l'instruction **GoSub** est rencontrée, les instructions qui suivent l'étiquette Calcul (Instructions-2) sont exécutées, jusqu'à la rencontre d'un **Return**. Alors les Instructions-1 sont exécutées. On remarquera la présence d'un **Exit Sub** qui évite d'exécuter une seconde fois les instructions d'où il est appelé. Il peut y avoir plusieurs instructions **Return** dans un sous-programme.

2- Appel externe – Procédures et fonctions :

L'instruction **GoSub** est peu utilisée dans Visual Basic. Les sous-programmes sont généralement des procédures (**Sub**) ou des fonctions (**Function**) qui offrent plus de souplesse : l'appel peut se faire n'importe où dans un fichier module à partir d'un autre fichier de l'application – et il est possible de transmettre des paramètres.

Une procédure et une fonction se ressemblent beaucoup. Elles suivent toutes les deux le schéma présenté précédemment. Leur seule différence se trouve dans le fait que la fonction retourne une valeur, contrairement à la procédure.

Pour mettre en œuvre une procédure ou une fonction, il y a deux phases : La déclaration et l'appel. la déclaration se fait selon les syntaxe suivantes :

[Public | Static] [Private] Sub nom-global [(liste-d'arguments)]

 [bloc-d'instructions]

[Exit Sub]

 [bloc-d'instructions]

End Sub

Pour une procédure, et :

[Public | Static] [Private] Function nom-global [(liste-d'arguments)] **[As type]**

 [bloc-d'instructions]

[Exit Sub]
[bloc-d'instructions]
[nom-global = expression]

End Function

Pour une fonction.

Dans les deux cas les mots clés Public, Static et Private Signifient :

Elément	Description
Public	Facultatif. Indique que la procédure Function est accessible à toutes les autres procédures de tous les modules. Si cet élément est utilisé dans un module contenant un élément Option Private, la procédure n'est pas disponible en dehors du projet.
Static	Facultatif. Indique que les variables locales de la procédure Function sont conservées entre les appels. L'attribut Static n'a pas d'effet sur les variables déclarées en dehors de la procédure Function, même si elles sont utilisées dans cette dernière.
Private	Facultatif. Indique que les procédure Function n'est accessible qu'à d'autres procédures du module dans lequel elle a été déclarée.

Dans les deux cas, la liste-d'arguments à la forme suivante :

[Optional] [ByVal | ByRef] [ParamArray] varname [()] [As type] [= defaultvalue]

Elément	Description
Optional	Facultatif. Indique qu'un argument n'est pas obligatoire. S'il est utilisé, les arguments suivants doivent également être facultatifs et être déclarés à aucun argument si le mot clé ParamArray est utilisé.
ByVal	Facultatif. Indique que l'argument est passé par valeur. Moyen permettant de passer à une procédure la valeur d'un argument plutôt que son adresse. La procédure peut de ce fait accéder à une copie de la variable. La valeur réelle de cette dernière n'est donc pas modifiée par la procédure à laquelle elle est passée.
ByRef	Facultatif. Indique que l'argument est passé par référence. ByRef est l'option par défaut dans Visual Basic.
ParamArray	Facultatif. Utilisé uniquement comme dernier argument pour indiquer que le dernier argument est un tableau Optional d'éléments de type Variant. Le mot clé ParamArray, qui permet d'indiquer un nombre quelconque d'arguments, ne peut pas être utilisé avec les mots clés ByVal, ByRef ou Optional.
Varname	Nom de la variable représentant l'argument. respecte les conventions standard d'affectation de noms aux variables.
Type	Facultatif. Type de données de l'argument passé à la procédure
Defaultvalue	Facultatif. Toute constante ou expression constante. Valide uniquement pour les paramètres Optional.

Exemple :

Sub AfficheRésultat (Mois As Integer, Titre As string)

'Instructions

End Sub

La procédure s'appelle AfficheRésultat. Lors de son appel il faut fournir deux paramètres, le premier étant un entier et le second une chaîne de caractères. Les instructions sont exécutées jusqu'à la rencontre du End Sub qui marque la fin physique de la procédure, ou d'une instruction Exit Sub qui provoque un retour. L'appel de cette procédure peut se faire de la façon suivante :

AfficheRésultat 5, « Résultats de Mai »

Ou encore :

AfficheRésultat (5, « Résultats de Mai »)

Les deux syntaxes équivalentes, mais l'utilisation de Call implique l'emploi des paramètres. Une procédure peut ne pas avoir de paramètres :

Sub Enregistre ()

'Instructions

End Sub

L'appel se fait alors de la façon suivante :

Enregistre

Ou :

Call Enregistre

Une fonction se caractérise par le fait qu'elle a une valeur de retour dont le type doit être calculé. Par exemple :

Function Racine (n1 As Double, n2 As Double) As Double

'Instructions

Racine = ...

End Function

Avant de quitter la fonction, ou d'appeler l'instruction **Exit Function**, il faut donc donner une valeur de retour. L'utilisation de la fonction se fait de la façon suivante.

N = Racine (3, 4)

Les parenthèses sont ici obligatoires, même si aucun paramètre n'est défini pour la fonction.

Une procédure ou fonction définie dans un fichier feuille peut être appelée à partir d'une autre procédure ou fonction du même fichier, mais pas à partir des autres fichiers. Tandis que les procédures ou fonctions définies dans un module (fichier .BAS) peuvent être appelées de n'importe où, sauf si le mot Private est utilisé lors de la déclaration, ce qui restreint la portée de la procédure au module où elle est définie.

Exemples :

'Cette fonction définie par l'utilisateur renvoie la racine carrée de l'argument qui lui est transmis.

Function CalculSquareRoot (NumberArg As Double) As Double

If NumberArg < 0 Then ' Evalue l'argument

Exit Function ' Quitte pour revenir à la procédure appelante.

Else

CalculateSquareRoot = Sqr (NumberArg) ' Renvoie la racine carré

EndIf

End Function

Le mot clé **ParamArray** permet à une fonction d'accepter un nombre variable d'arguments. dans la définition suivante, FirstArg est transmis par valeur.

Function CalcSum (By FirstArg as Integer, ParamArray OtherArgs ())

Dim ReturnValue

‘ Si la fonction est appelée sous la forme :

ReturnValue = CalcSum (4, 3, 2, 1)

‘ Les variables locales ont les valeurs suivantes : FirstArg = 4, OtherArgs (1) = 3, OtherArgs (2) = 2, etc., en supposant que la limite inférieure par défaut des tableaux = 1.

Les arguments Optional peuvent être dotés de types et de valeurs par défauts autres que Variant.

‘ Si les arguments d’une fonction sont définis sous la forme :

Function MyFunc(MyStr As String, Optional MyArg1 As Integer = 5, Optional MyArg2 = « Dolly »)

Dim RetVal

‘ La fonction peut être appelée de plusieurs façons :

RetVal = MyFunc(« Bonjour », 2, « à tous ») ‘ Les 3 arguments sont fournis.

RetVal = MyFunc(« Test », , 5) ‘ Les deuxièmes argument est omis.

‘ Arguments un et deux indiqués en utilisant des argument est omis.

RetVal = MyFunc(MyStr := « Bonjour », MyArg1 :=7)

V- Traitement des chaînes de caractères :

Les comparaisons de chaînes de caractères peuvent se faire en différenciant ou non les majuscules de minuscules. Pour cela, il faut utiliser l’instruction option Compare dans la section des déclarations d’une feuille ou d’un module. Pour que les majuscules et les minuscules soient différenciées, on peut utiliser :

Option Compare Binary

Pour ne pas faire de distinction

Option compare Text

De plus, certaines instructions utilisent un paramètre pour changer localement l’option retenue.

Les fonctions suivantes agissent sur des chaînes de caractères.

Fonction	Signification	Exemple
& opérateur	Retourne une chaîne qui est la concaténation de deux autres. On peut également utiliser l’opérateur + pour la concaténation, mais il peut être ambigu dans le cas de variants.	« bonjour » & « messieurs » retourne « bonjour messieurs »
Space\$ Space	Construit une chaîne de caractères qui comprend un nombre donné d’espaces.	Space\$ (20) Retourne une chaîne de 20 espaces
String\$ String	Construit une chaîne de caractères qui comprend un nombre donné d’un caractère spécifique.	String\$ (20, "*") String\$ (20, 42) Retourne une chaîne de 20 caractères "*"
Len	Retourne la longueur d’une chaîne de caractères, ou le nombre d’octets nécessaires pour le stockage d’une variable.	Len ("coucou !") Retourne 8 Dim d as double : print Len(d) Retourne 8
Instr	Retourne la position d’une sous-chaîne de caractères dans une chaîne de caractères, en tenant compte ou non des majuscules / minuscules	Instr ("mon papa est là", "pa") Retourne 5 (1 ^{er} "pa"), et : Instr (6, "mon papa est là", "pa") Retourne 7 (2 ^e "pa")
Left\$, Left	Retourne uen chaîne de caractères constituée	Left\$ ("Jean Pierre Durand", 4)

	des n caractères situés à gauche dans une autre chaîne de caractères.	Retourne "Jean"
Right", Right	Retourne une chaîne de caractères constituée des n caractères situés à droite dans une autre chaîne de caractères.	Right\$ ("Jean Pierre Durand", 6) Retourne "Durand"
Mid\$, Mid	Retourne une chaîne de caractères constituée des n caractères situés à partir d'une position donnée dans une autre chaîne de caractères.	Mid\$ ("Jean Pierre Durand", 6) Retourne "Pierre", et : Mid\$ ("Jean Pierre Durand", 6) Retourne "Pierre Durand"
Mid\$, Mid	Remplace dans une chaîne de caractères n caractères d'une autre chaîne de caractères.	Nom\$ = "Jean Pierre Durand" Mid\$ (Nom\$, 6) = "Michel" Place dans Nom\$ "Jean Michel Durand" et : Mid\$ (Nom\$, 6, 1) = "Michel" Place dans Nom\$ "Jean Pierre Durand"
LTrim\$, LTrim	Retourne une chaîne de caractères d'où sont supprimés tous les espaces situés à gauche d'une chaîne de caractères donnée.	Ltrim\$ ("Bonjour") Retourne "Bonjour"
RTrim\$, RTrim	Retourne une chaîne de caractères d'où sont supprimés tous les espaces situés à droite d'une chaîne de caractères donnée.	RTrim\$ ("Bonjour") Retourne "Bonjour"
Trim\$, Trim	Retourne une chaîne de caractères d'où sont supprimés tous les espaces situés à gauche et à droite d'une chaîne de caractère donnée	Trim\$ ("Bonjour") Retourne "Bonjour"
Lcase\$, Lcase	Convertit en Minuscules tous les caractères alphabétiques d'une chaîne de caractères.	Lcase\$ ("Bonjour") Retourne "Bonjour"
Ucase\$, Ucase	Convertit en majuscules tous les caractères alphabétiques d'une chaîne de caractères.	Ucase\$ ("Bonjour") Retourne "Bonjour"
Lset	Place dans une chaîne de longueur fixe une chaîne de caractères en la cadrant à gauche et en la remplissant à droite avec des espaces.	Dim nom as string * 10 Lset nom = "Coucou" Print ">>" ; Nom ; "<<" Affiche ">> Coucou ...<<" (les "." représentent des espaces)
Rset	Place dans une chaîne de longueur fixe une chaîne de caractères en la cadrant à droite et en la remplissant à gauche avec des espaces,	Dim Nom as String * 10 Rset Nom = "Coucou" Print ">>" ; Nom ; "<<" Affiche ">> ... Coucou <<" (les "." représentent des espaces).
Strcomp	Compare deux chaînes de caractères	Stromp ("André", "Bernard") Retourne - 1

1- Boîte de saisie :

La boîte de saisie affiche un texte et retourne une chaîne de caractères saisie. Deux boutons Ok et Annuler permettent à l'utilisateur de valider sa saisie ou de l'annuler.

Saisie = Inputbox ("nom ? ", "saisir un nom", "ahmed")

Le premier paramètre est le texte affiché, le deuxième le titre de la boîte le troisième la valeur par défaut dans le champ de saisie. Si l'utilisateur sélectionne le bouton Ok, la chaîne retournée est le contenu du champ de saisie de la boîte, sinon elle est vide.

2- MSGBOX :

Affiche un message dans une boîte de dialogue, attend que l'utilisateur clique sur un bouton, puis renvoie une valeur de type Integer qui indique le bouton choisi par l'utilisateur.

MsgBox (prompt[, buttons] [, title] [, helpfile, context])

La syntaxe de la fonction MsgBox comprend les arguments nommés suivants :

Elément	Description
Prompt	Expression de chaîne affichée comme message dans la boîte de dialogue. La longueur maximale de l'argument prompt est d'environ 1 024 caractères selon la largeur des caractères utilisés.
Buttons	Facultatif. Expression numérique qui représente la somme des valeurs indiquant le nombre et le type de boutons à afficher, le style d'icône à utiliser, l'identité du bouton par défaut, ainsi que la modalité du message. Si l'argument buttons est omis, sa valeur par défaut est 0.
Title	Facultatif. Expression de chaîne affichée dans la barre de titre de la boîte de dialogue. Si l'argument title est omis, le nom de l'application est placé dans la barre de titre.
Helpfile	Facultatif. Expression de chaîne indiquant le fichier d'aide à utiliser pour fournir une aide contextuelle sur la boîte de dialogue. Si l'argument helpfile est défini, l'argument context doit l'être également.
Context	Facultatif. Expression indiquant le numéro de contexte attribué par l'auteur de l'aide à la rubrique appropriée. Si l'argument context est défini, l'argument helpfile doit l'être également.

Valeur :

L'argument buttons prend les valeurs suivantes :

Constante	Valeur	Description
VbOKOnly	0	Affiche le bouton OK uniquement
VbOKCancel	1	Affiche les boutons OK et Annuler
VbAbortRetryIgnore	2	Affiche le bouton Abandonner, Réessayer et Ignorer
VbYesNoCancel	3	Affiche les boutons Oui, Non et Annuler
VbYesNo	4	Affiche les boutons Oui et Non
VbRetryCancel	5	Affiche les boutons Réessayer et Annuler
VbCritical	16	Affiche l'icône Message critique
VbQuestion	32	Affiche l'icône Requête d'avertissement
VbExclamation	48	Affiche l'icône Message d'avertissement
VbInformation	64	Affiche l'icône Message d'information
VbDefaultButton1	0	Le premier bouton est le bouton par défaut
VbDefaultButton2	256	Le deuxième bouton est le bouton par défaut
VbDefaultButton3	512	Le troisième bouton est le bouton par défaut
VbDefaultButton4	768	Le quatrième bouton est le bouton par défaut
VbApplicationModal	0	Boîte de dialogue modale. L'utilisateur doit répondre au message affiché dans la zone de message avant de pouvoir continuer de travailler dans l'application en cours.

VbSystemModal	1096	Modal système. Toutes les applications sont interrompues jusqu'à ce que l'utilisateur réponde au message affiché dans la zone de message
---------------	------	--

Valeurs renvoyées :

Constante	Valeur	Bouton
VbOK	1	OK
VbCancel	2	Annuler
VbAbort	3	Abandonner
VbRetry	4	Réessayer
VbIgnore	5	Ignorer
VbYes	6	Oui
VbNo	7	Non

Exemple :

```
Dim Msg, Style, Title, Help, Ctxt, Response, MyString
Msg = « Souhaitez-vous continuer ? » Définit le message
Style = vbYesNo + vbCritical + vbDefaultButton2 ' Définit les boutons
Title = « Démonstration de MsgBox » ' Définit le titre
Help = « DEMO.HLP » ' Définit le fichier d'aide
Ctxt = 1000 ' Définit le contexte de la rubrique
Response = MsgBox (Msg, Style, Title, Help, Ctxt) ' Affiche le message
If Response = vbYes Then ' L'utilisateur a choisi Yes
    MyString = « Yes » ' Effectue une Action
Else ' L'utilisateur a choisi No.
    MyString = « No » ' Effectue uen action
End If
```

VI- Création d'une interface utilisateur :

L'interface est peut-être la partie la plus importante d'une application mais elle est aussi celle que l'on remarque le plus. Pour les utilisateurs, elle représente l'application et certains ne doivent même pas savoir qu'un programme est exécuté en arrière-plan.

On distingue deux principaux styles d'interfaces : l'interface simple document (SDI) et l'interface multidocument (MDI). L'application WordPad fournie avec Microsoft Windows est un exemple d'interface SDI. En effet, cette application ne vous permet d'ouvrir qu'un seul document à la fois ; vous devez fermer un pour en ouvrir un autre. Par contre Microsoft Excel et Microsoft Word sont des interfaces MDI qui permettent d'afficher simultanément plusieurs documents, chaque document s'affichant dans sa propre fenêtre.

L'interface multidocument (MDI) vous permet de créer une application qui gère plusieurs feuilles à l'intérieur d'une seule feuille conteneur. Une application MDI permet à l'utilisateur d'afficher simultanément plusieurs documents, chaque document possédant sa propre fenêtre. Les documents ou fenêtre filles s'affichent dans une fenêtre mère, qui fournit l'espace de travail nécessaire à toutes les fenêtres filles de l'application.

Une application MDI permet à l'utilisateur d'afficher simultanément plusieurs documents, chaque document possédant sa propre fenêtre. Les documents ou fenêtres filles s'affichent dans une fenêtre mère, qui fournit l'espace de travail nécessaire à toutes les fenêtres filles de l'application. Les fenêtres filles sont incluses dans les limites de la zone de la fenêtre mère, lorsque vous réduisez celle-ci toutes les fenêtres de documents sont aussi réduites ; seule l'icône de la fenêtre mère apparaît dans la barre des tâches.

Une feuille fille est une feuille ordinaire dont la propriété MDIChild a la valeur True. Votre application peut inclure plusieurs feuilles filles MDI de types similaires ou différents.

Une feuille MDI standard est semblable à une feuille ordinaire, mais vous ne pouvez pas y placer directement un contrôle, sauf si ce dernier est doté de la propriété Align (tel qu'un contrôle PictureBox) ou s'il n'a pas d'interface visible (tel qu'un contrôle Timer).

1- Caractéristiques des feuilles MDI en mode exécution :

Lorsque vous réduisez une feuille fille, son icône s'affiche dans la feuille MDI et non dans la barre des tâches.

Lorsque vous agrandissez une feuille fille, le libellé est associé à celui de feuille MDI, et il s'affiche dans la barre de titre de cette dernière.

Si vous définissez la propriété AutoShowChildren, vous pouvez afficher automatiquement les feuilles lors de leur chargement (True), ou les charger sans les afficher (false).

Les menus actifs d'une feuille fille s'affichent, le cas échéant, dans la barre de menus de la feuille MDI, et non dans la feuille fille.

2- Créateurs de menus :

Le créateur de menu permet de créer des menus et des barres de menu, d'ajouter de nouvelles commandes aux menus existants, de remplacer les commandes de menu existantes par vos propres commandes, d'ajouter et de supprimer des menus et des barres de menu existants.

Pour afficher le Créateur de menus, c'est à partir du menu Outils, l'option Créateur de menus.

Les deux propriétés de contrôles de menu les plus importantes sont les suivantes :

Name : non utilisé pour désigner un contrôle de menu dans un code.

Caption : Texte qui s'affiche dans le contrôle.

1. Dans la zone de texte **Caption**, entrez le premier titre de menu que vous voulez afficher dans la barre de menus. Insérez une perluète (&) avant la lettre que vous voulez définir comme touche d'accès rapide pour cet élément de menu. La lettre est automatiquement soulignée dans le titre du menu.
2. Dans la zone de texte **Name**, entrez le nom que vous voulez utiliser pour désigner le contrôle de menu dans le code.
3. Cliquez sur les flèches gauches ou droites pour modifier le niveau de retrait du contrôle ; dans le créateur de menu chaque niveau de retrait est précédé de quatre points (...).
4. Sélectionnez **suivant** pour créer un autre contrôle de menu ou cliquez sur **insérer** pour ajouter un contrôle de menu entre les contrôles existants.

5. Pour affecter une touche de raccourci à un élément de menu, sélectionnez une touche de fonction ou une combinaison de touches dans la zone de liste modifiable **Shortcut**.
6. L'option **Index** permet de créer un groupe de contrôles de menu : c'est un ensemble d'éléments d'un même menu qui partagent le même nom et les mêmes procédures d'événements.
7. La propriété **Checked** permet d'insérer une coche ✓ dans un menu pour indiquer à l'utilisateur l'état d'une condition activée/désactivée.
8. La propriété **Enabled** permet de désactiver un menu et vous pouvez l'activer au niveau de votre code. Exemple la commande coller du menu édition est désactivé.
9. La propriété **Visible** permet d'afficher ou de masquer un contrôle de menu. Le fait d'activer la coche permet d'afficher le contrôle de menu par contre si on la désactive le contrôle sera masquer.
10. La propriété **WindowsList** est utilisée au niveau d'un contrôle de menu d'une feuille MDI ou d'une feuille fille MDI pour afficher la liste des feuilles filles ouverte.

3- Organisation des feuilles filles :

Certaines applications affichent dans un menu des actions telles que Mosaïque, Cascade et Réorganiser les icônes, avec la liste des feuilles filles ouvertes. La méthode Arrange permet de réorganiser les feuilles filles dans une feuille MDI. Vous pouvez afficher des feuilles filles en cascade, superposées horizontalement ou sous la forme d'icônes de feuilles filles disposées le long de la partie inférieure de la feuille MDI.

```
Private sub mnucascade_click ()  
    FrmMDI.Arrange Vbcascade      ' Dispose les feuilles filles en cascade  
End Sub  
Private sub mnuTileH_click ()  
    FrmMDI.Arrange VbTileHorizontal ' Dispose les feuilles filles en mosaïque (horizontale)  
End Sub  
Private sub mnuTileV_click ()  
    FrmMDI.Arrange VbTileVertical   ' Dispose les feuilles filles en mosaïque (Verticale)  
End Sub  
Private sub mnuArrange_click ()  
    FrmMDI.Arrange VbArrangeIcons   ' Organisez toutes les icônes des feuilles filles  
End Sub
```

4- Les menus contextuels :

Un menu contextuel est un menu flottant qui s'affiche sur une feuille, indépendamment de la barre de menus. Comme les éléments de ce menu dépendent de l'endroit où le pointeur a été placé lors du clic du bouton droit de la souris, ce type de menu est appelé menu contextuel.

Le code suivant affiche le menu mnufile lorsque l'utilisateur clique sur une feuille avec le bouton droit de la souris. Vous pouvez utiliser l'événement MouseUp ou MouseDown pour détecter le moment où l'utilisateur clique sur le bouton droit de la souris, bien que MouseUp soit l'événement standard :

```
Private Sub Form_MouseUp (Button as Integer, Shift as Integer, X as Single, Y as Single)  
    If Button = 2 then ' Vérifie si le bouton droit de la souris a été actionné.  
        PopupMenu mnufile ' Affiche le menu fichier en tant que menu contextuel  
    End If  
End Sub
```

Les contrôles de Visual Basic :

Catégories de contrôles :

Les trois catégories principales de contrôles de Visual Basic sont les suivantes :

- **Contrôles intrinsèques :** il s'agit, par exemple, des contrôles CommandButton et Frame. Ces contrôles intrinsèques figurent en permanence dans la boîte à outils, à la différence des contrôles ActiveX et des objets insérables qui peuvent être retirés ou ajoutés à la boîte à outils.
- **Contrôles ActiveX :** ils existent sous forme de fichiers séparés dotés de l'extension .ocx. Ces fichiers renferment l'ensemble des contrôles disponibles dans toutes les éditions de Visual Basic (contrôles DBGrid, DBCombo, DBList, etc.) et ceux utilisés seulement dans les éditions Professionnelle et Entreprise (tels que les contrôles listView, Toolbar, Animation et SSTab).
- **Objets insérables :** il peut s'agir d'une feuille de calcul Microsoft Excel contenant la liste des employés de votre société ou d'un objet Calendrier Microsoft Project contenant des informations de planification relatives à un projet. Etant donné que ces objets peuvent être ajoutés dans la boîte à outils, ils sont considérés comme des contrôles.

Les groupes de contrôles :

Un groupe de contrôles est un ensemble de contrôles de même type et portant le même nom. Les contrôles d'un groupe partagent également les mêmes procédures d'événement. un groupe de contrôles comprend au moins un élément et peut comporter autant d'éléments que le permettent les ressources et la mémoire de votre système.

L'ajout de groupes de contrôles consomme moins de ressources que l'ajout de plusieurs contrôles individuels de même type à une feuille au moment de la création. Vous pouvez aussi utiliser des groupes de contrôles si vous souhaitez que plusieurs contrôles utilisent un code commun.

Pour créer un groupe de contrôles, dessinez un contrôle, cliquez sur copier puis sur coller. Visual Basic affiche une boîte de dialogue vous demandant de confirmer la création du groupe de contrôles, choisissez oui pour confirmer l'opération.

Le nouveau contrôle prend la valeur d'index 1. Le premier contrôle que vous avez dessiné possède la valeur 0.

CheckBox :

Un contrôle CheckBox affiche une coche ✓ lorsque la case à cocher correspondante est activée ; la ✓ disparaît lorsque la case à cocher est désactivée. Les cases à cocher offrent à l'utilisateur le choix entre Vrai/Faux ou Oui/Non. Vous pouvez utiliser des groupes de cases à cocher pour afficher plusieurs possibilités dont l'utilisateur peut sélectionner une ou plusieurs. Vous pouvez également définir la valeur d'une case à cocher par programme, à l'aide de la propriété Value.

Les contrôles CheckBox et OptionButton fonctionnent de la manière similaire, à une différence importante près : vous pouvez activer autant de cases à cocher (CheckBox) que vous le souhaitez sur une même feuille, tandis qu'il n'est possible d'activer qu'une seule case d'option (OptionButton) à la fois dans un groupe.

Pour afficher un texte en regard d'une case à cocher, définissez sa propriété **Caption**.

Check1.caption = « Windows 95 »

La propriété **Value** indique l'état de la case : activée, désactivée ou grisée.

Etat	Valeur	Constante
Désactivé	0	VbUnchecked
Activé	1	VbChecked
Grisé	2	VbGrayed

ComboBox :

Un contrôle ComboBox réunit les fonctionnalités d'une zone de texte et celles d'une zone de liste. Ce type de contrôle permet à l'utilisateur de procéder à une sélection soit en tapant du texte (TextBox) dans la partie modifiable, soit en sélectionnant l'un des éléments dans la partie (ListBox).

Styles des listes :

Il existe trois styles de listes modifiables :

Style	Valeur	Constante
Liste modifiable déroulante	0	VbComboDropDown
Liste Modifiable simple	1	VbComboSimple
Zone de liste déroulante	2	VbComboDropDownList

Liste modifiable déroulante :

Cette liste donne la possibilité à l'utilisateur de taper directement du texte ou de cliquer sur la zone afin d'ouvrir une liste de choix.

Liste modifiable simple :

La liste modifiable simple affiche en permanence la liste de tous les choix possibles. Une barre de défilement vertical est automatiquement insérée lorsque le nombre d'éléments est supérieur au nombre d'éléments pouvant être affichés dans la liste.

Zone de liste déroulante :

Une zone de liste déroulante ressemble à une zone de liste normale, elle affiche une liste d'options parmi lesquelles l'utilisateur doit choisir. Toutefois contrairement à la liste modifiable déroulante, on ne peut pas taper du texte dans la zone.

Ajout d'élément dans une liste :

Pour ajouter des éléments dans un contrôle ComboBox, on utilise la méthode **AddItem**.

Combo1.AddItem « Technicien en informatique »

Combo1.AddItem « Technicien en Bureautique »

Pour ajouter un élément à un endroit précis, spécifiez une valeur d'index à la suite de l'élément à ajouter. Par exemple, la ligne suivante insère « Tib » en première position et repositionne les éléments qui suivent :

Combo1.AddItem « Tib » , 0

Vous pouvez également ajouter des éléments à la liste en utilisant la propriété **List** au moment de la création.

Tri d'une liste :

Afin de trier les éléments d'une liste par ordre alphabétique, on affecte la valeur **True** à la propriété **Sorted**.

Combo.Sorted = True

Suppression des éléments d'une liste :

L'instruction **RemoveItem** permet de supprimer des éléments dans une liste modifiable. Cette méthode dispose d'un argument, **index**, qui sert à spécifier l'élément à supprimer :

Combo1.RemoveItem 2

Pour supprimer tous les éléments d'une liste modifiable, utilisez la méthode **Clear**.

Combo1.Clear

ListIndex :

La propriété **ListIndex** définit ou renvoie l'index de l'élément sélectionné dans le contrôle et n'est disponible qu'au moment de l'exécution. la valeur de cette propriété est 0 si le premier élément de la liste est sélectionné, 1 si le deuxième élément est sélectionné, et ainsi de suite. La valeur de la propriété **ListIndex** est - 1 si aucun élément n'est sélectionné.

ListCount :

Pour connaître le nombre d'éléments contenus dans une liste modifiable, on utilise la propriété **ListCount**.

Text1.Text = Combo1.ListCount

CommandButton :

Le contrôle **CommandButton** permet à l'utilisateur de commencer ou d'interrompre une opération, ou encore d'y mettre fin. Lorsque le bouton de commande (**CommandButton**) correspondant est sélectionné, il semble s'enfoncer à l'écran, ce qui explique pourquoi on appelle parfois les boutons de commande « boutons-poussoirs ».

Lorsque l'utilisateur clique sur un bouton, il appelle une commande, écrite dans la procédure d'événement **Click** associée au bouton.

Pour afficher un texte sur un bouton de commande, définissez sa propriété **Caption**

Commande1.Caption = « Ajouter »

La propriété **Caption** permet de créer des touches de raccourci correspondant à vos boutons de commande. Pour ce faire, vous ajoutez un signe **&** devant la lettre qui sera utilisée comme touche d'accès rapide. Par exemple, pour créer une touche de raccourci pour le bouton ajouter, vous ajoutez un signe **&** devant la lettre « **A** ».

Command1.Caption = « &Ajouter »

A l'exécution, la lettre « **A** » sera soulignée et l'utilisateur pourra sélectionner le bouton de commande en appuyant simultanément sur les touche **ALT + A**

L'utilisateur peut toujours cliquer sur un bouton de commande pour choisir . pour permettre à l'utilisateur de le sélectionner en appuyant directement sur la touche **ENTREE**, affectez la valeur **True** à la propriété **Default** du contrôle **CommandButton** correspondant.

Pour permettre à l'utilisateur de le sélectionner en appuyant sur la touche **ECHAP**, affectez la valeur **True** à sa propriété **Cancel**.

Contrôle Data :

Le contrôle Données (**Data**) est utilisé pour créer des applications simples de base de données sans écrire aucun code. De plus, il permet de créer des applications plus complètes qui vous octroient un plus haut niveau de contrôle sur vos données.

Les contrôles Zone de liste dépendante (**DBList**), Liste modifiable dépendante (**DBGrid**) et Microsoft FlexGrid (**MSFlexGrid**) sont tous capables de gérer des jeux d'enregistrements lorsqu'ils sont liés à un contrôle **Data**. Grâce à ces contrôles, il est possible d'afficher et de manipuler simultanément plusieurs enregistrements.

DriveListBox :

La liste de lecteurs (DriveListBox) est une zone de liste déroulante permettant à l'utilisateur de sélectionner un lecteur valide au moment de l'exécution. ce contrôle affiche la liste de tous les lecteurs valides du système de l'utilisateur. Si l'utilisateur sélectionne un nouveau lecteur dans la liste celui-ci apparaît en haut de la zone de liste.

Vous pouvez utiliser du code pour examiner la propriété **Drive** de la liste de lecteurs et déterminer quel est le lecteur sélectionné. Vous pouvez également spécifier le lecteur qui doit apparaître en haut de la zone de liste à l'aide de l'instruction suivante :

Drive1.Drive = « C:\ »

La liste de lecteurs affiche les lecteurs valides disponibles. Le fait de choisir un lecteur dans cette zone de liste ne permet pas automatiquement de changer de lecteur au niveau du système d'exploitation en la spécifiant comme argument dans l'instruction **ChDrive** :

ChDrive Drive1.Drive

DirListBox :

La liste de répertoires (**DirListBox**) affiche l'arborescence du lecteur en cours sur le système de l'utilisateur, en commençant par le dossier de plus haut niveau. Lorsque l'arborescence s'affiche, le nom du dossier en cours apparaît en surbrillance et en retrait par rapport aux dossiers de niveau supérieur dans la hiérarchie, jusqu'à la racine.

La propriété **Path** de la liste de répertoires permet de définir ou de renvoyer le dossier en cours dans la zone de liste (**ListIndex = -1**).

Vous pouvez également affecter la propriété **Drive** de la liste de lecteurs à la propriété **Path** de la liste de répertoires :

Dir1.Path = Drive1.Drive

Une fois cette affectation exécutée, la liste de répertoires affiche tous les dossiers et sous-dossiers disponibles sur ce lecteur. Par défaut, la liste de répertoires affiche également tous les dossiers situés au-dessus du dossier en cours du lecteur affecté à la propriété **Dir1.Path**, ainsi que tous les sous-dossiers situés immédiatement au-dessous.

Pour définir le dossier de travail en cours, utilisez l'instruction **ChDir**. Par exemple, l'instruction suivante définit comme dossier en cours celui qui est affiché dans la liste de répertoires :

ChDir Dir1.Path

FileListBox :

Un contrôle **FileListBox** (Liste de fichiers) permet de répertorier les fichiers contenus dans le dossier spécifié par la propriété **Path** au moment de l'exécution. A l'aide de l'instruction suivante, vous pouvez afficher tous les fichiers se trouvant dans le dossier et le lecteur en cours :

File.Path = Dir1.Path

Visual Basic accepte le caractère générique ?. Par exemple, la spécification, ? ? ?.txt permet d'afficher les fichiers dont le nom de base se compose de trois caractères et qui possèdent l'extension.txt.

Frame :

Un contrôle **Frame** permet de regrouper des contrôles de manière aisément identifiable. On peut aussi utiliser ce contrôle pour subdiviser fonctionnellement une feuille, en vue de séparer, par exemple, des groupes de contrôles **OptionButton**.

Pour regrouper des contrôles, dessinez d'abord le contrôle **Frame**, puis dessinez les contrôles à l'intérieur. Cela vous permet de déplacer simultanément le cadre et les contrôles

qu'il contient. Si vous dessinez un contrôle hors d'un cadre, puis tentez de l'amener à l'intérieur de ce cadre, le contrôle sera superposé et non intégré au cadre. Vous devrez alors déplacer séparément le cadre et les contrôles.

Pour donner un titre au contrôle **Frame** vous pouvez changer au niveau de la propriété **Caption**.

HScrollBar – VScrollBar :

Les barres de défilement sont des contrôles permettant de parcourir rapidement une longue liste d'éléments ou un grand nombre d'informations. Elles peuvent également indiquer la position courante sur une échelle. Vous pouvez utiliser une barre de défilement comme dispositif d'entrée de données ou comme indicateur de vitesse ou de quantité pour régler, par exemple, le volume sonore d'un jeu sur ordinateur ou pour visualiser le temps écoulé lors d'un processus minuté.

Les barres de défilement utilisent les événements **Scroll** et **Change** pour surveiller le mouvement du curseur le long de la barre de défilement.

Événement	Description
Change	Se produit après tout mouvement du curseur de défilement.
Scroll	Se produit lors du déplacement du curseur. Ne se produit pas si l'utilisateur clique sur les flèches de défilement ou sur la barre de défilement.

L'événement **Scroll** permet d'accéder à la valeur de la barre de défilement à mesure que l'utilisateur déplace le curseur. L'événement **Change** se produit une fois qu'il relâche le curseur ou clique sur la barre ou les flèches de défilement.

La propriété **Value** (dont la valeur par défaut est 0) est une valeur spécifiée sous forme d'entier correspondant à la position du curseur dans la barre de défilement. Lorsque la position du curseur de défilement est définie à la valeur minimale, le curseur se place à la position la plus à gauche (pour les barres de défilement horizontal) ou la plus haute (pour les barres de défilement vertical). Lorsque la position du curseur de défilement est définie à sa valeur maximale, le curseur se place à l'extrême droite ou tout en bas de la barre. De même, la valeur correspondant au milieu de la plage place le curseur au milieu de la barre de défilement.

L'utilisateur peut non seulement cliquer avec la souris pour modifier la valeur de la barre de défilement, mais peut également faire glisser le curseur de défilement jusqu'à n'importe quel point de la barre. La valeur résultante dépend de la position du curseur, mais elle se trouve systématiquement dans la plage comprise entre les valeurs des propriétés **Min** et **Max** définies par l'utilisateur.

La propriété **LargeChange** spécifie la valeur de la modification qui survient lorsque l'utilisateur clique dans la barre de défilement tandis que la propriété **SmallChange** spécifie la valeur de la modification qui survient lorsque l'utilisateur clique sur les flèches situées aux extrémités de la barre de défilement. La propriété **Value** de la barre de défilement augmente ou diminue selon les valeurs affectées aux propriétés **LargeChange** et **SmallChange**. Vous pouvez positionner le curseur de défilement au moment de l'exécution en affectant à la propriété **Value** une valeur comprise entre 0 et 32767 inclus.

Image :

Ce contrôle permet d'afficher une image. Ce contrôle affiche des graphiques dans les formats suivants : image bitmap, icône, métafichier, ainsi que des métafichiers étendus et des fichiers JPEG ou GIF.

Les images peuvent être chargées dans le contrôle image au moment de la création en sélectionnant la propriété **Picture** dans la fenêtre propriétés du contrôle, ou au moment de l'exécution en utilisant la propriété **Picture** et la méthode **LoadPicture**.

Image1.Picture = LoadPicture (« C : \Windows \ Winlogo.bmp »)

Quand une image est chargée dans le contrôle Image, celui-ci est redimensionné à la taille de l'image, quelle que soit sa taille lorsqu'il a été dessiné sur la feuille.

Pour effacer le graphique dans le contrôle Image, utilisez la méthode **LoadPicture** sans spécifier de nom de fichier. Par exemple :

Image1.Picture = LoadPicture

La propriété **Stretch** détermine si l'image s'agrandit automatiquement lorsque le contrôle Image est redimensionné au moment de la création. Si la valeur **True** est affectée à la propriété **Stretch**, l'image chargée dans le contrôle Image par l'intermédiaire de sa propriété **Picture** est agrandie automatiquement.

Label :

Un contrôle **Label** est un contrôle graphique qui permet d'afficher du texte que l'utilisateur ne peut pas modifier directement. Il peut être utilisé aussi pour afficher des informations au moment de l'exécution en réponse à un événement ou à un processus de votre application.

Pour modifier le texte afficher dans un contrôle Label, utilisez la propriété **Caption**. Au moment de la création, vous pouvez définir cette propriété en la sélectionnant dans la fenêtre Propriétés du contrôle. Ou bien au moment de l'exécution en écrivant :

Label1.Caption = « Type de Résolution

Par défaut, lorsque le texte de la propriété **Caption** excède la largeur du contrôle, il revient automatiquement à la ligne, et il est tronqué s'il dépasse la hauteur du contrôle.

Pour que le contrôle soit automatiquement redimensionné en fonction de son contenu, affectez la valeur **True** à la propriété **AutoSize**. Le contrôle s'étend en longueur afin de s'adapter au contenu complet de la propriété **Caption**. Pour que le contenu revienne automatiquement à la ligne et s'étende en hauteur, affectez la valeur **True** à la propriété **WordWrap**.

Line :

Un contrôle **Line** est un contrôle graphique affiché sous la forme d'une ligne horizontale, verticale ou diagonale.

Vous pouvez utiliser un contrôle **Line** au moment de la création afin de tracer des lignes sur des feuilles. Vous pouvez contrôler la position, la longueur, la couleur et le style des contrôles.

La couleur et le style d'un segment sont définis à l'aide des propriétés **BorderStyle** et **BorderColor**.

La propriété **BorderStyle**, propose six styles de ligne :

- **Transparent (0 – Transparent)**
- **Continu (1 – Solid)**
- **Tiret (2 – Dash)**
- **Point (3 – Dot)**
- **Tiret – Point (4 – Dash-Dot)**
- **Tiret – Point – Point (5 – Dash-Dot-Dot)**
- **Intérieur plein (6 – Inside Solid)**

La couleur est déterminée par la propriété **BorderColor** dans la fenêtre Propriétés du contrôle Line.

ListBox :

Un contrôle **ListBox** affiche une liste dans laquelle l'utilisateur peut sélectionner un ou plusieurs éléments. Si le nombre d'éléments est trop grand pour tenir dans la zone affichée, une barre de défilement lui est automatiquement ajoutée.

Pour ajouter des éléments dans une zone de liste, utilisez la méthode **AddItem**, dont la syntaxe est la suivante :

List1.AddItem « Maroc »

Pour ajouter un élément à un endroit précis, spécifiez une valeur d'index à la suite de l'élément à ajouter .Ex :

List1.AddItem « Algérie », 0

La valeur 0, spécifie le premier élément de la liste.

Vous pouvez spécifier que les éléments d'une liste soient triés par ordre alphabétique en affectant la valeur **True** à la propriété **Sorted**. Le tri s'effectue sans tenir compte des majuscules et minuscules.

Pour supprimer des éléments d'une zone de liste, utilisez la méthode **RemoveItem** en spécifiant l'index de l'élément à supprimer :

List1.RemoveItem 0

Pour supprimer tous les éléments d'une liste, utilisez la méthode **Clear** :

List1.Clear

La manière la plus simple pour connaître la valeur de l'élément sélectionné dans une liste consiste à utiliser la propriété **Text**. La propriété **Text** correspond toujours à un élément de la liste que l'utilisateur sélectionne au moment de l'exécution.

Si vous souhaitez connaître la position de l'élément sélectionné dans une liste, utilisez la propriété **ListIndex**. Cette propriété définit ou renvoie l'index de l'élément sélectionné dans le contrôle et n'est disponible qu'au moment de l'exécution. Si aucun élément n'est sélectionné, la propriété **ListIndex** a la valeur -1. La propriété **ListIndex** du premier élément de la liste a la valeur 0.

Pour renvoyer le nombre d'éléments contenus dans une liste, utilisez la propriété **ListCount**.

Text1.Text = List1.ListCount

La propriété **Columns** vous permet de spécifier le nombre de colonnes d'une zone de liste. Cette propriété peut prendre les valeurs suivantes :

Valeur	Description
0	Zone de liste à une colonne avec défilement vertical
1	Zone de liste à une colonne avec défilement horizontal
> 1	Zone de liste multicolonne avec défilement horizontal

Vous pouvez permettre aux utilisateurs de sélectionner plusieurs éléments dans une liste en utilisant la propriété **MultiSelect**, qui peut prendre les valeurs suivantes :

Valeur	Type de Sélection	Description
0	None	Zone de liste standard
1	Simple	Un clic ou l'appui sur ESPACE permet de sélectionner ou de désélectionner des éléments supplémentaires dans la liste
2	Extended	La combinaison MAJ + clic ou MAJ + une touche de direction permet d'étendre la sélection afin d'y

		inclure tous les éléments compris entre l'élément sélectionné et celui sélectionné précédemment. CTRL + clic permet de sélectionner un élément dans la liste.
--	--	---

Option Button :

Un contrôle **OptionButton** affiche une option qui peut être activée ou désactivée. Dans un groupe de boutons d'option, l'utilisateur ne peut effectuer qu'une seule sélection.

Bien que les contrôles **OptionButton** et **CheckBox** semblent fonctionner de manière similaire, il existe entre eux une différence importante : lorsqu'un utilisateur sélectionne une case d'option, les autres cases du même groupe sont automatiquement désactivées, ce qui n'est pas le cas des cases à cocher, dont l'utilisateur peut sélectionner un nombre illimité.

Pour changer le label d'un bouton d'option vous pouvez utiliser la propriété **Caption** :

Option1.Caption = « Pennyum »

La propriété **Value** indique si le bouton est sélectionné. Si tel est le cas, la valeur devient **True**.

OptOui.Value = True

PictureBox :

Un contrôle **PictureBox** peut afficher un élément graphique issu d'une image bitmap, d'une icône ou d'un métafichier, ainsi que d'un métafichier étendu ou de fichiers JPEG ou GIF.

L'élément graphique est découpé si le contrôle n'est pas assez grand pour afficher l'image complète.

Le contrôle **PictureBox** est similaire au contrôle **Image** dans le sens qu'ils peuvent être utilisés pour afficher des graphiques dans l'application et qu'ils gèrent tous deux les mêmes formats graphiques. Toutefois, le contrôle **PictureBox** contient des fonctionnalités qui n'existent pas dans le contrôle **Image**, par exemple, la possibilité de jouer le rôle de conteneur pour les autres contrôles et de gérer des méthodes graphiques.

Un contrôle **PictureBox** peut être utilisé pour regrouper des contrôles **OptionButton** et pour afficher les données de sortie des méthodes graphiques et les textes écrits au moyen de la méthodes **Print**.

Pour que la zone d'image d'un contrôle **PictureBox** se redimensionne automatiquement de manière à pouvoir afficher l'ensemble du graphisme, affectez la valeur **True** à sa propriété **AutoSize**.

Il est possible de charger des images dans le contrôle **PictureBox** au moment de la création, en sélectionnant la propriété **Picture** dans la fenêtre Propriétés du contrôle, ou au moment de l'exécution en utilisant la propriété **Picture** et la méthode **LoadPicture**.

Picture1.Picture = LoadPicture (« C:\Windows\Winlogo.bmp »)

PictureBox est Data sont les seuls contrôles Visual Basic standard que vous pouvez placer dans la zone interne d'une feuille MDI. Vous pouvez utiliser un contrôle **PictureBox** pour regrouper des contrôles en haut ou en bas de la zone interne afin de créer une barre d'outils ou une barre d'état.

Comme les feuilles, les zones d'image peuvent être utilisées pour afficher les données en sortie des méthodes graphiques telles que **Circle**, **Line** et **Point**. Par exemple, vous pouvez utiliser la méthode **Circle** pour dessiner un cercle dans une zone d'image en affectant la valeur **True** à la propriété **AutoRedraw** du contrôle.

Picture1.AutoRedraw = True

Picture1.Circle (1200, 1000), 750

Le fait d'affecter la valeur **True** à la propriété **AutoRedraw** permet aux données en sortie de ces méthodes d'être dessinées dans le contrôle et automatiquement redessinées quand le contrôle **PictureBox** est redimensionné ou affiché de nouveau après avoir été masqué par un autre objet.

Vous pouvez utiliser le contrôle **PictureBox** pour afficher du texte en sortie en utilisant la méthode **Print** et en affectant la valeur **True** à la propriété **AutoRedraw**. Par exemple :

Picture1.Print « Bonjour !CasaBlanca »

Shape :

Le contrôle **Shape** est un contrôle graphique qui s'affiche sous forme de rectangle, de carré, d'ellipse, de cercle, de rectangle arrondi ou de carré arrondi.

Vous pouvez définir le style, la couleur, le style de remplissage, la couleur et le style de bordure de toutes les formes que vous dessinez sur une feuille.

La propriété **Style** du contrôle **Shape** vous propose six formes prédéfinies.

Forme	Style	Constante
Rectangle	0	VbShapeRectangle
Carré	1	VbShapeSquare
Ovale	2	VbShapOval
Cercle	3	VbShapCircle
Rectangle à coins arrondis	4	VbShapeRoundedRectangle
Carré à coins arrondis	5	VbShapeRoundedSquare

Vous pouvez utiliser les propriétés **FillStyle** et **BorderStyle** pour définir le style de remplissage et de bordure de toutes les formes que vous dessinez sur une feuille.

La propriété **FillStyle**, comme la propriété **Style**, vous propose une variété de motifs de style de remplissage prédéfinis : Plein, transparent, ligne horizontale, ligne verticale, Diagonale montante, Diagonale descendante, Croix et Croix diagonale.

La propriété **BorderStyle** vous propose une variété de style de bordure prédéfinis : Transparent, Continu, Tiret, Point, Tiret-Point, Tiret-Point-Point et intérieur plein.

Les propriétés **BackColor** et **FillColor** vous permettent d'ajouter des couleurs à une forme et sa bordure.

TextBox :

Un contrôle **TextBox**, parfois appelé champ d'édition ou contrôle d'édition, affiche des informations entrées au moment de la création, tapées par l'utilisateur ou affectées au contrôle au moment de l'exécution.

Le texte entré dans le contrôle **TextBox** est contenu dans la propriété **Text**. Par défaut, vous pouvez entrer jusqu'à 2048 caractères dans une zones de texte. Si vous affectez la valeur **true** à la propriété **MultiLine** du contrôle, vous pouvez entrer jusqu'à 32 Ko de texte.

Lorsque la taille du texte excède les limites du contrôle, vous pouvez affecter la valeur **True** à la propriété **multiline** du contrôle pour gérer automatiquement le renvoi du texte à la ligne et ajouter des barres de défilement horizontal, vertical, ou les deux, en définissant la propriété **ScrollBars**.

Quand vous affectez la valeur **true** à la propriété **MultiLine**, vous pouvez aussi ajuster l'alignement du texte sur aligner à gauche (**Left Justify**), centrer (**center**) ou ligner à droite (**Right Justify**). Par défaut, le texte est aligné à gauche. I la valeur **False** est affectée à la propriété **MultiLine**, la définition de la propriété **Alignment** n'a aucun effet.

La propriété **PasswordChar** spécifie le caractère qui apparaît dans la zone de texte. Par exemple, si vous souhaitez que la zone de texte pour mot de passe contienne des astérisques, affectez la valeur* à cette propriété dans la fenêtre Propriétés. Quel que soit le caractère que l'utilisateur tape dans la zone de texte, un astérisque apparaît.

La propriété **MaxLength** vous permet de déterminer le nombre de caractères pouvant être tapés dans la zone de texte. Si ce nombre est dépassé, un bip sonore est émis et la zone n'accepte plus aucun caractère.

Vous pouvez utiliser la propriété **Locked** pour empêcher les utilisateurs de modifier le contenu d'une zone de texte. Affectez la valeur **True** à cette propriété pour permettre aux utilisateurs de faire défiler et de mettre en surbrillance le contenu d'une zone de texte sans pour autant autoriser les modifications.

Timer :

Les contrôles Minuterie (**Timer**), indépendants de l'utilisateur, réagissent au passage du temps. Vous pouvez les programmer pour déclencher certaines actions à intervalles réguliers. Une réponse type de la minuterie consiste à vérifier l'horloge système pour voir si le moment est venu d'effectuer une tâche donnée. Les minuteries s'avèrent également utiles pour d'autres types de traitement d'arrière-plan.

Un contrôle **Timer** possède deux propriétés principales.

Propriété	Valeur
Enabled	Si vous souhaitez que la minuterie se déclenche dès le chargement de la feuille, affectez la valeur True à cette propriété. Dans le cas contraire, laissez cette propriété définie sur la valeur False. Vous pouvez choisir qu'un événement extérieur (le clic d'un bouton de commande, par exemple) déclenche le fonctionnement de la minuterie.
Interval	Nombre de millisecondes entre les événements de la minuterie

La propriété **Enabled** de la minuterie est différente de la propriété **Enabled** des autres objets. Dans la plupart des objets, cette propriété détermine si l'objet peut répondre à un événement provoqué par l'utilisateur. dans le contrôle **Timer**, le fait d'affecter la valeur **False** à la propriété **Enabled** interrompt le fonctionnement de la minuterie.

La propriété **Interval** détermine plus la fréquence que la durée. La longueur de l'intervalle dépend du degré de précision que vous recherchez.

L'intervalle peut être compris entre 0 et 64767 inclus, ce qui signifie que même l'intervalle le plus long sera de l'ordre d'une minute (64,8 secondes environ).

L'exemple suivant permet d'afficher au niveau d'une étiquette l'heure système en utilisant un contrôle **Timer** dont l'intervalle est 1000 (1 seconde) et la propriété **Enabled** est **True**.

```
Private Sub Timer1_Timer ()  
    Lbltime.Caption = Time  
End Sub
```

VII- Les fichiers

Un fichier est constitué d'un ensemble de données, normalement situées sur disque ou disquette, et identifiées par un nom de fichier. Un fichier n'est autre qu'une suite d'octets connexes enregistrés sur un disque. Lorsque votre application accède à un fichier, elle doit imaginer ce que représente ces octets (des caractères, des enregistrements de données, des entiers, des chaînes, etc.)

Une application doit traiter un fichier en trois phases :

- Ouverture du fichier. Le système réserve une place en mémoire pour permettre l'accès au fichier par l'application. L'ouverture peut se faire selon plusieurs modes, et elle peut donner lieu à la création du fichier s'il n'existait pas. Elle est réalisée à l'aide de l'instruction **Open**.
- Traitement du fichier. Ce peut être des lectures ou des écritures de données. Il est également possible de se positionner dans le fichier à un endroit quelconque.
- Fermeture du fichier, qui indique au système qu'il peut libérer la mémoire réservée pour le fichier. La fermeture se fait avec l'instruction **Close**.

1- Modes de traitement des fichiers :

Visual Basic permet de voir le contenu d'un fichier de trois façons différentes correspondant à trois modes d'accès.

- Le mode d'accès séquentiel est utilisé pour la lecture et l'écriture de fichiers textes dans des blocs continus.
- Le mode d'accès direct où le fichier est constitué d'enregistrements ayant chacun une même longueur. Chaque enregistrement peut lui-même être constitué de plusieurs champs, qu'il appartient à l'application de traiter correctement.
- Le mode d'accès binaire où le fichier est simplement constitué d'une suite d'octets sans liens logiques.

L'accès séquentiel a été conçu pour les fichiers texte sans mise en forme. Chaque caractère du fichier est censé représenter un caractère de texte ou une séquence de mise en forme du texte. Les données sont stockées sous forme de caractères ANSI. Un fichier ouvert pour accès direct est censé être constitué d'un ensemble d'enregistrements de même longueur. Grâce aux types définis par l'utilisateur, vous pouvez créer des enregistrements constitués de nombreux champs pouvant même posséder chacun un type de données différent. Les données sont stockées sous forme d'informations binaires.

L'accès binaire vous permet de stocker des données dans un ordre quelconque au sein de fichiers. Ce mode d'accès est semblable à l'accès direct, mais s'en différencie par le fait qu'il n'y a aucune estimation du type de données ou de la longueur des enregistrements. Toutefois, il est nécessaire de connaître avec précision la manière dont les données ont été écrites dans le fichier pour les extraire correctement.

Il faut bien comprendre qu'il s'agit de modes d'accès, qui ne sont pas enregistrés dans le fichier. Un même fichier peut être traité dans les trois modes, avec des résultats qui peuvent évidemment être souhaités.

1) Accès Séquentiel :

Le mode d'accès séquentiel est en réalité le nom commun donné à trois modes d'accès, spécifiés lors de l'ouverture du fichier avec **Open**, qui précisent le type d'accès effectué :

Input indique que l'accès au fichier doit se faire en lecture seule.

Output indique son accès en écriture. Si le fichier est ouvert dans ce mode, il est réinitialisé à vide et son précédent est perdu.

Append indique son accès en écriture. Le contenu précédent est conservé, et l'écriture dans le fichier ajoute de nouveaux enregistrements.

Exemple:

Open « MYFILLE.TXT » For Input As 1

Open « MYFILLE.TXT » For Output As 1

Open « MYFILLE.TXT » For Append As 1

Si le fichier est ouvert en lecture, on peut en lire les enregistrements successifs à l'aide de l'instruction **line input** qui lit la prochaine ligne (initialement la première), puis positionne le fichier à la ligne suivante. La fonction **EOF** peut être utilisée pour savoir si la lecture a atteint la fin du fichier. L'accès séquentiel autorise également d'autres instructions de lecture: **Input #** et **Write**.

Dim varline

Open « FICHTEST » For Input As #1 'Ouvre le fichier.

Do While Not EOF(1) 'Effectue la boucle jusqu'à la fin du fichier

LineInput #, Varline 'Lit une ligne et l'affecte à la variable varline

Txtnom.text = varline 'Affiche le résultat dans la zone de texte.

Loop

Close # 1 ' Ferme le fichier.

Pour écrire dans un fichier en mode séquentiel, ouvert en **Output**, ou en **Append**, on utilise l'instruction **Print #**, qui écrit le texte donné, et ajoute les caractères de fin de lignes;

Print # 1, thebox. text

ou bien l'instruction **Write #**

Dim varnom as string, varnum as integer

Varnom = « Bonjour »

Varnom = 1230

Write # 1 varnom, varnum

Visual Basic écrit donc les caractères suivants (signe de ponctuation compris) dans le fichier: « Bonjour », 1230.

2) Accès direct ou accès aléatoire :

Un fichier ouvert en mode d'accès est vu comme une suite d'enregistrements de longueur fixe. La structure d'un enregistrement est souvent définie à l'aide d'une directive **Type** qui permet d'en spécifier les champs. Par exemple:

Type enrclient

Nom As String* 25

Numéro As Integer

Rue As String * 20

Montant As Currency

End Type

Par la suite il est nécessaire de déclarer une variable d'enregistrement, soit avec **Dim** pour une utilisation en local avec **Global** au niveau d'un module pour une utilisation globale.

Dim Client As EnreClient

Global Client as EnrClient

L'ouverture du fichier se fait toujours avec l'instruction **Open**, dans laquelle le mode est **Random**. De plus, il faut préciser la longueur de l'enregistrement, avec le mot clé **Len**. Par exemple:

Open « Clients. Dat » For Random As 1 Len = Len (Client)

Contrairement aux fichiers à accès séquentiel, il est possible de lire et d'écrire à la fois des enregistrements dans un fichier à accès direct, à moins que l'accès ait été restreint lors de l'ouverture avec le mot clé **Access**.

La lecture d'un enregistrement se fait avec l'instruction **Get**, et l'écriture avec l'instruction **Put**. Il est également possible de se positionner à un enregistrement donné avec l'instruction **Seek**. Par exemple:

Get #1, Numenr, Client

'Saisie de l'enregistrement en modification

Put #1 Numenr, Client

3) Accès binaire :

Accéder à un fichier en mode binaire se révèle lorsque les deux autres modes ne se prêtent pas à sa structure. Le mode d'accès binaire considère le fichier comme une suite d'octets, sans structure particulière. Celle-ci peut être, si nécessaire, ajoutée dans la logique du programme.

L'ouverture se fait toujours avec l'instruction **Open**:

Open « fichier For Binary As 1

Comme pour le mode d'accès direct, les instructions **Get** et **Put** permettent de lire ou d'écrire des données dans le fichier. Le numéro d'enregistrement est ici remplacé par un numéro d'octet dans le fichier, le premier octet ayant le numéro 1. **Seek** permet de modifier la position courante dans le fichier. Par exemple:

Dim Tampon as string* 16

Get1,32, Tampon

Lit 16 octets (taille de Tampon) à partir du 32^{ème} octet du fichier.

On peut également utiliser la fonction **Input\$** qui retourne une chaîne de caractères contenant un nombre spécifié de caractères du fichier, à partir de la position courante. L'exemple:

Dim Tampon As String

Seek 1,32

Tampon = Input\$ (16,1)

Est équivalent au précédent.

2- Les instructions de traitement des fichiers :

1) Open :

Permet d'exécuter une opération d'Entrée / Sortie (E/S) sur un fichier.

Open pathname **For** mode [**Access** access] **As** [#] filenumber [**Len** = reclength]

La syntaxe de l'instruction **Open** comprend les éléments suivants:

Elément	Description
pathname	Expression de chaîne indiquant un nom de fichier qui peut comprendre un nom de répertoire ou de dossier et un nom de lecteur.
mode	Mot clé indiquant le mode d'ouverture du fichier: Append, Binary, Input, Output, ou Random. S'il n'est pas indiqué, le fichier est ouvert en mode Random.
Access	Facultatif. Mot clé indiquant les opérations autorisées sur le fichier ouvert: Read, Write ou Read Write
filenumber	Numéro de fichier valide compris entre 1 et 511 inclus. Utilisez la fonction FreeFile pour obtenir le prochain numéro de fichier disponible
reclength	Facultatif. Nombre inférieur ou égal à 32767 (octets) . Pour les fichiers ouverts en mode Random, cette valeur représente la longueur de l'enregistrement. Pour les fichiers séquentiels, elle représente le nombre de caractères contenus dans la zone tampon.

Remarques :

Avant de pouvoir exécuter une opération d'Entrée/Sortie sur un fichier, vous devez l'ouvrir. L'instruction Open permet d'associer une zone tampon d'Entrée/Sortie au fichier et de déterminer le mode d'accès à ce fichier dans cette zone.

Si le fichier indiqué par l'argument pathname n'existe pas, il est créé au moment où un fichier est ouvert en mode Append, Binary, Output ou Random.

Si le fichier, a déjà été ouvert par un autre processus et si le type indiqué n'est pas autorisé, l'instruction Open échoue et une erreur se produit.

La clause Len n'est pas prise en compte si l'argument mode a la valeur Binary.

Important : En mode Binary, Input et Random, vous n'êtes pas obligé de fermer un fichier avant de l'ouvrir sous un autre numéro de fichier. En mode Append et Output, vous devez fermer un fichier avant de l'ouvrir sous un autre numéro de fichier.

Ouvre le fichier FICHTEST en mode lecture séquentielle.

Open « FICHTEST » For Input As #1

Ouvre le fichier en mode Random (aléatoire). Ce fichier contient des enregistrements de type Record défini par l'utilisateur.

'Désigne le type défini par l'utilisateur.

Type Record

ID As Integer

Name As String * 20

End Type

Dim MyRecord As Record 'Déclare la variable.

Open « FICHTEST » For Random As #1 Len = Len (MyRecord)

2) Line Input :

Lit une ligne unique à partir d'un fichier séquentiel ouvert et l'attribue à une variable de type string.

Line Input # filenumber, varname

La syntaxe de l'instruction Line Input # comprend les éléments suivants:

filenumber Tout numéro de fichier valide

varname Nom de variable de type Variant ou String valide

Remarques :

Les données lues dans un fichier à l'aide de l'instruction Line Input # sont généralement écrites à l'aide de l'instruction **Print #**.

L'instruction **Line Input #** lit à un les caractères d'un fichier jusqu'au premier retour chariot (**Chr(13)**) ou retour chariot - saut de ligne (**Chr(13)**) + (**Chr(10)**) rencontré.

Les séquences retour chariot - saut de ligne ne sont pas prises en compte (elles ne sont pas ajoutées à la chaîne de caractères).

Dim TextLine

Open « FICHTEST » For Input As #1 'Ouvre le fichier.

Do While Not EOF (1) 'Effectue la boucle jusqu'à la fin du fichier.

Line Input #1, TextLine 'Lit la ligne dans la variable.

Debug. Print TextLine 'Affiche dans la fenêtre Exécution.

Loop

Close #1 'Ferme le fichier.

3) Input () :

Renvoie une valeur de type String contenant les caractères lus dans un fichier ouvert en mode Input ou Binary.

Input (number, [#] filename)

number Toute expression numérique valide indiquant le nombre de caractère à renvoyer .

filename Tout numéro de fichier valide.

Remarques :

Les données lues avec l'instruction **Input #** sont généralement écrites dans un fichier à l'aide de l'instruction **write #**. Utilisez cette instruction uniquement avec des fichiers ouverts en mode Input ou Binary.

Contrairement à l'instruction Input #, la fonction Input renvoie tous les caractères lus, y compris les virgules, les retours chariot, les sauts de ligne, les points d'interrogation et les espaces à gauche.

Dans le cas de fichiers ouverts en mode Binary, une erreur se produit si vous tentez de lire le fichiers à l'aide de la fonction Input jusqu'à ce que la fonction EOF renvoie la valeur True. Utilisez les fonctions LOF et Loc au lieu de la fonction EOF pour la lecture de fichiers binaires à l'aide de l'instruction Input, ou utilisez l'instruction Get avec la fonction EOF.

MyChar = Input (1, #1) 'Lit un caractère.

4) Input # :

lit des données dans un fichier séquentiel ouvert et les attribue à des variables.

Input # filename, varlist

La syntaxe de l'instruction Input # comprend les éléments suivants :

Filename Tout numéro de fichier valide.

Varlist Liste, délimitée par des virgules, de variables auxquelles sont attribuées les valeurs lues dans le fichier.

Remarques :

Les données lues avec l'instruction **Input #** sont généralement écrites dans une fichier à l'aide de l'instruction **Write #**. Utilisez cette instruction uniquement avec des fichiers ouverts en mode Input ou Binary.

Lorsqu'elles sont lues, les chaînes standard ou les données numériques sont effectuées aux variables sans être modifiées.

Les guillemets double (" ") inclus dans les données d'entrée ne sont pris en compte.

Les éléments de données d'un fichier doivent apparaître dans le même ordre que les variables de l'argument varlist et doivent correspondre à des variables possédant le même type de données. Si une variable est numérique et si les données ne le sont pas, la valeur zéro est attribuée à la variable.

Si la fin du fichier est atteinte pendant l'entrée d'un élément de données, cela met fin à l'entrée et une erreur se produit.

Note :

Pour être sûr que les données d'un fichier sont correctement lues et attribuées aux variables à l'aide de l'instruction Input #, utilisez l'instruction Write # au lieu de l'instruction Print # pour écrire les données dans les fichiers. L'instruction Write # garantit une délimitation correcte de chaque champ de données.

Dim MyString, MyNumber

Open « FICHTEST » For Input As # 1

Do While Not EOF(1)

 'Lit les données dans deux variables.

Input # 1, MyString, MyNumber

Loop

Close # 1 'Ferme le fichier.

5) Print # :

Ecrit des données mises en forme pour l'affichage dans un fichier séquentiel.

Print #filename, [out putlist]

La syntaxe de l'instruction Print # comprend les éléments suivants :

Element	Description
filename	Tout numéro de fichier valide
outputlist	Facultatif. Expression ou liste d'expression à imprimer.

Remarques:

Les données écrites à l'aide de l'instruction Print # sont généralement lues dans un fichier avec les instructions Line Input # ou Input.

Si vous omettez l'argument outputlist et si vous indiquez uniquement un séparateur de liste après l'argument filename, une ligne vierge est imprimée dans le fichier. Les différentes expressions peuvent être séparées indifféremment par un espace ou un point-virgule. Ex:

Open « FICHTEST » For Output As # 1 ouvre le fichier en écriture

Print # 1, « ceci est un test » Ecrit le texte dans le fichier.

Print # 1, 'Insère une ligne vierge dans le fichier

Print # 1, « Zone 1 »; Tab; « Zone 2 » sépare la chaîne en deux.

Print # 1, « Bonjour »; " "; « à tous » sépare les chaînes de caractères avec des espaces.

Print # 1, Spc (5); "5 espaces à gauche" Ecrit cinq espaces à gauche.

Print # 1, Tab (10); "Bonjour" Ecrit le mot à partir de la colonne 10.

6) Write # :

Ecrit des données dans un fichier séquentiel.

Write # filename, [outputlist]

Elément	Description
filename	Tout numéro de fichier valide
outputlist	Facultatif. Une ou plusieurs expressions numériques ou expressions de chaîne délimitées par des séparateurs virgules, devant être écrites dans un fichier.

Remarques :

Les données écrites à l'aide de l'instruction Write # sont généralement dans un fichier avec l'instruction Input #.

Si vous omettez l'argument outputlist et si vous insérez un séparateur virgule après l'argument filename, une ligne vierge est imprimée dans le fichier. Vous pouvez séparer plusieurs expressions par un espace, un point-virgule ou une virgule. Un espace ou un point virgule ont le même effet.

Contrairement à l'instruction Print #, l'instruction Write # insère des virgules entre les éléments et des guillemets doubles de part et d'autre des chaînes de caractères au moment de leur écriture dans le fichier. Vous n'avez donc pas à placer des séparateurs explicites dans la liste. L'instruction Write # insère un caractère de passage la ligne, c'est à dire un retour chariot-saut de ligne (chr(13)+chr(10)), après l'écriture dans le fichier du dernier caractère contenu dans l'argument outputlist.

Exemple :

‘Ouvre le fichier en écriture.

Open « FICHTEST » For Output As # 1

Write #1, « Bonjour » à tous », 234 ‘écrit des données séparées par des virgules.

Write #1 Ecrit une ligne vierge.

7) Close :

Termine les opérations d’Entrée / Sortie (E/S) dans un fichier ouvert à l’aide de l’instruction Open.

Close [filenumberlist]

L’argument facultatif filenumberlist représente ou un plusieurs numéros de fichiers qui utilisent la syntaxe suivante, où filenumber peut être tout numéro de fichier valide:

[[#] filenmber], [[#] filenmber].....

Remarques :

Si vous omettez l’argument filenumberlist, tous les fichiers actifs ouverts à l’aide de l’instruction Open sont fermés.

Lorsque l’instruction Close est exécutée, l’association d’un fichier avec son numéro de fichier est désactivée.

Close # 1.

8) Freefile :

Renvoie une valeur de type Integer représentant le prochain numéro de fichier pouvant être utilisée par l’instruction Open.

FreeFile [(rangnumber)]

L’argument facultatif rangnumber est une valeur de type Variant qui indique la plage dans laquelle doit se trouver le prochain numéro de fichier disponible renvoyé par la fonction. indiquez 0 (valeur par défaut) pour renvoyer un numéro de fichier compris entre 1 et 255 inclus. Indiquez 1 pour renvoyez un numéro de fichier compris entre 256 et 511.

Remarques :

Utilisez la fonction FreeFile pour fournir un numéro de fichier encore non utilisé.

Cet exemple utilise la fonction FreeFile pour renvoyez le prochain numéro de fichier disponible. Cinq fichiers sont ouverts à l’intérieur de la boucle et des données sont écrites dans chacun d’eux.

Dim FileNumber

File Number = FreeFile ‘Lit le numéro de fichier inutilisé

Open « TEST » For Output As # File Number

9) Get :

Lit les données d’un fichier disque ouvert et les place dans une variable.

Get [#] filenumber, [recnumber], varname

La syntaxe de l’instruction Get comprend les éléments suivants :

Elément	Description
Filenumber	Tout numéro de fichier valide.
Recnumber	Facultatif. Donnée de type Variant (Long). Numéro de l’enregistrement (fichiers ouverts en mode Random) ou de l’octet (fichiers ouverts en mode Binary) par lequel la lecture débute.

Varname	Nom de variable valide destiné à recevoir les données lues.
---------	---

Remarques :

Les données lues avec l'instruction Get sont généralement placées dans un fichiers à l'aide de l'instruction Put.

Le premier enregistrement ou octet d'un fichier occupe la position 1, le deuxième enregistrement ou octet la position 2, et ainsi de suite. Si l'argument recnumber est omis, la lecture commence à partir de l'enregistrement ou de l'octet suivant la dernière instruction Get ou Put (ou à partir de celui désigné par la dernière fonction Seek). Vous devez toujours inclure des séparateurs virgules. Exemples :

Get # 4, FileBuffer

Les règles suivantes s'appliquent aux fichiers ouverts en mode Random :

- Si la longueur des données lues est inférieure à la longueur indiquée dans la clause Len de l'instruction Open, l'instruction Get lit les enregistrements suivants conformément aux limites de longueur des enregistrements. L'espace compris entre la fin d'un enregistrement et le début de l'enregistrement suivant est rempli par le contenu de la zone tampon du fichier. Comme il est impossible de déterminer avec certitude le volume des données de remplissage, il est généralement préférable que la longueur des enregistrements corresponde à la longueur des données lues.
- Si la variable lue est une chaîne de longueur variable, l'instruction Get lit d'abord un descripteur de deux octets contenant la longueur de la chaîne, puis les données qui sont placées dans la variable. Par conséquent, la longueur d'enregistrement indiquée par la clause Len dans l'instruction Open doit être supérieur d'au moins deux octets à la longueur réelle de la chaîne.

‘Désigne un type défini par l'utilisateur.

Type Record

ID As Integer

Name As String* 20

End Type

Dim MyRecord As Record, Position ‘Déclare les variables.

‘Ouvre l'exemple de fichier en accès aléatoire.

Open « FICHTEST » For Random As # 1 Len = (MyRecord)

Position 3 'Définit le numéro d'enregistrement

Get # 1, Position, MyRecord ‘Lit le troisième enregistrement.

Close # 1 'Ferme le fichier.

10) Put :

Écrit les données d'une variable dans un fichier disque.

Put [#]filename, [recnumber], varname

La syntaxe de l'instruction Put comprend les éléments suivants :

Elément	Description
Filenumber	Tout numéro de fichier valide.
Recnumber	Facultatif. Donnée de type Variant (Long). Numéro de l'enregistrement (fichiers ouverts en mode Random) ou de l'octet(fichiers ouverts en mode Binary) qui marque le début de l'écriture.

Varname Nom de la variable contenant les données à écrire sur le disque.

Remarque :

Les données écrites à l'aide de l'instruction Put sont généralement lues dans un fichier avec l'instruction Get.

Le premier enregistrement ou octet d'un fichier occupe la position 1, le deuxième enregistrement ou octet la position 2, et ainsi de suite. Si l'enregistrement recnumber est omis, l'écriture commence à partir de l'enregistrement ou de l'octet suivant la dernière instruction Get ou Put (ou à partir de celui désigné par la dernière fonction Seek). Vous devez inclure des séparateurs virgules, Exemple :

Put # 4, FileBuffer

Les règles suivantes s'appliquent aux fichiers ouverts en mode Random :

- Si la longueur des données écrites est inférieure à la longueur indiquée dans la clause Len de l'instruction Open, l'instruction Put poursuit l'écriture des enregistrements suivants conformément aux limites de longueur des enregistrements. L'espace compris entre la fin d'un enregistrement et le début de l'enregistrement suivant est rempli par le contenu de la zone tampon du fichier. Comme il est impossible de déterminer avec certitude le volume des données de remplissage, il est généralement préférable que la longueur des enregistrements corresponde à la longueur des données écrites. Une erreur se produit si la longueur des données écrites est supérieure à celle indiquée dans la clause Len de l'instruction Open.
- Si la variable écrite est une chaîne de longueur variable, l'instruction Put écrit d'abord un descripteur de 2 octets contenant la longueur de la chaîne, puis la variable. La longueur d'enregistrement indiquée par la clause Len dans l'instruction Open doit être supérieure d'au moins 2 octets à la longueur réelle de la chaîne.

'Désigne un type défini par l'utilisateur.

Type Record

ID As Integer

Name As String* 20

End Type

Dim MyRecord As Record, RecordNumber 'Déclare les variables.

'Ouvre le fichier en accès aléatoire.

Open « FICHTEST » For Random As # 1 Len = Len (MyRecord)

For RecordNumber = 1 To 5 'Effectue la boucle 5 fois

MyRecord.ID = RecordNumber 'Définit l'identificateur.

MyRecord.Name = »Mon Nom « & RecordNumber 'Crée une chaîne

Put # 1, RecordNumber, MyRecord 'Ecrit l'enregistrement dans le fichier.

Next RecordNumber

Close # 1 'Ferme le fichier.

11) Seek :

Renvoie une valeur de type Long indiquant la position de lecture / écriture courante dans un fichier ouvert à l'aide de l'instruction Open.

Seek(Filenumber)

L'argument filenumber est une valeur de type Integer contenant un numéro de fichier valide.

Remarques :

La fonction Seek renvoie une valeur comprise entre 1 et 2 147 483 647, inclus.

Le tableau suivant indique les valeurs renvoyées pour chaque mode d'accès au fichier.

Mode	Valeur renvoyée
Random	Numéro de l'enregistrement suivant lu ou écrit.
Binary Output Append Input	Position de l'octet à partir duquel l'opération suivante doit avoir lieu. Le premier octet d'un fichier occupe la position 1, le deuxième octet la position 2, et ainsi de suite.

'Désigne un type défini par l'utilisateur.

Type Record

ID As Integer

Name As String * 20

End Type

Dans le cas de fichiers ouverts en mode d'accès aléatoire, la fonction Seek renvoie le numéro de l'enregistrement suivant.

Dim MyRecord As Record 'Déclare la variable.

Open « FICHTEST » For Random As #1 Len = Len(MyRecord)

'Effectue la boucle jusqu'à la fin du fichier.

Do While Not EOF(1)

Get # 1, MyRecord 'Lit l'enregistrement suivant.

'Affiche le numéro d'enregistrement dans la zone de texte pos

Textpos.text = Seek(1)

Loop

Close # 1 'Ferme le fichier.

Dans le cas de fichiers ouverts en d'autres modes que le mode Random, la fonction Seek renvoie la position de l'octet à partir duquel l'opération suivante doit s'effectuer. nous supposons que FICHTEST est un fichier contenant quelques lignes de texte.

Dim MyChar

Open « FICHTEST » For Input As # 1 'Ouvre le fichier en lecture.

Do While Not EOF(1) 'Effectue la boucle jusqu'à la fin du fichier.

MyChar = Input (1, #1) 'Lit le caractère suivant.

Textpos.text = Seek (1) 'Affiche la position de l'octet dans la zone de texte.

Loop

Close # 1 'Ferme le fichier.

12) Loc :

Renvoie une valeur de type Long indiquant la position de lecture / écriture courante dans un fichier ouvert.

Loc(Filenumber)

L(argument filenumber peut être tout numéro de fichier valide de type integer.

Remarques :

Le tableau suivant décrit la valeur renvoyée en fonction du mode d'accès au fichier :

Mode	Valeur renvoyée
Random	Numéro du dernier enregistrement lu ou écrit dans le fichier.
Sequential	Position courante de l'octet dans le fichier divisé par 128. Les informations

renvoyées par la fonction Loc pour les fichiers séquentiels ne sont toutefois ni utilisées, ni obligatoires.

Binary Position du dernier octet lu ou écrit.

Dim MyLocation, MyLine

Open « FICHTEST » For Binary As #1 'Ouvre le fichier qui vient d'être créé.

Do While MyLocation < LOF(1) 'Effectue la boucle jusqu'à la fin du fichier.

MyLine = MyLine & Input (1, #1) 'Lit le caractère dans la variable.

MyLocation = Loc(1) 'Lit la position en cours dans le fichier.

Loop

Close #1 'Ferme le fichier.

13) EOF :

renvoie une valeur de type Integer contenant la valeur Boolean True lorsque la fin d'un fichier ouvert en mode Random ou Input séquentiel est atteinte.

EOF (filenumber)

L'argument filenumber est une valeur de type Integer contenant n'importe quel numéro de fichier valide.

Remarques :

Utilisez la fonction EOF pour éviter de générer une erreur lorsque vous tentez d'obtenir des données au-delà de la fin d'un fichier.

La fonction EOF renvoie la valeur False tant que la fin du fichier n'est pas atteinte. Avec des fichiers ouverts en mode Random ou Binary, une erreur se produit si vous tentez de lire le fichier à l'aide de la fonction Input jusqu'à ce que la fonction EOF renvoie la valeur True. Utilisez les fonctions LOF et Loc au lieu de la fonction EOF pour la lecture de fichiers binaires avec la fonction Input, ou utilisez l'instruction Get avec la fonction EOF. Pour des fichiers ouverts en mode Output, la fonction EOF renvoie toujours la valeur True.

Dim InputData

'Ouvre le fichier afin d'y entrer des données.

Open « MONFICH » For Input As # 1

Do While Not EOF(1) 'Vérifie si la fin du fichier est atteinte.

Line Input #1, InputData 'Lit les lignes de données.

Loop

Close #1 'Ferme le fichier.

14) LOF :

renvoie une valeur de type Long représentant la taille, exprimé en octets, d'un fichier ouvert à l'aide de l'instruction Open.

LOF(filenumber)

L'argument filenumber peut représenter toute valeur de type Integer contenant un numéro de fichier valide.

Note :

Utilisez la fonction FileLen pour obtenir la longueur d'un fichier qui n'est pas ouvert.

Dim FileLength

Open « FICHTEST » For Input As #1 'Ouvre le fichier.

FileLength = LOF(1) 'Lit la taille du fichier.

Close #1 'Ferme le fichier.

15) FileLen :

Renvoie une valeur de type Long indiquant la longueur en octets d'un fichier.

FileLen (pathname)

L'argument pathname est une expression de chaîne définissant un fichier. L'argument pathname peut préciser le répertoire ou dossier, et le lecteur.

Remarques :

Si le fichier indiqué est ouvert lors de l'appel de la fonction FileLen, la valeur renvoyée représente la taille du fichier juste avant son ouverture.

Note :

Pour connaître la longueur d'un fichier ouvert, utilisez la fonction LOF.

Dim MySize

'Renvoie la longueur du fichier (en octets).

MySize = Filelen (« FICHTEST »)