Chapitre 2

Le langage Visuel Basic

2.1 Déclaration des variables

2.1.1 Types de données

Type de données	Taille	Désignation
Byte	1 octet	Entier de 0 à 255
Boolean	2 octets	True ou False
Integer	2 octets	Entier de -32 768 à 32 767
Long	4 octets	Entier long de -2 147 483 648 à 2 147 483 647
Single	4 octets	Réel en simple précision Valeurs négatives : $-3{,}402823 * 10^{38}$ à $-1{,}401298 * 10^{-45}$ Valeurs positives : $1{,}401298 * 10^{-45}$ à $3{,}402823 * 10^{38}$
Double	8 octets	Réel en double précision
Date	8 octets	Type date du 01/01/100 au 31/12/9999
String		Chaîne de caractères de longueur non fixe pouvant contenir jusqu'à environ 2 milliards de caractères.
String * N		Chaîne de caractères de longueur fixe précisée à l'aide de la constante N pouvant aller jusqu'à environ 65 400 caractères.
Variant		C'est un type « passe par tout » pouvant contenir différents types de valeur : numériques, chaînes, etc.
Object	4 octets	C'est aussi un type passe par tout, mais il permet de déclarer des variables dynamiques pouvant référencer en suite des objets de différentes classes.

2.1.2 Déclaration des variables

La déclaration des variables peut être réalisée à l'aide de l'un des mots clés suivants : Dim, Private, Public, Global, Static. (Dim étant le mot clé le plus utilisé).

Syntaxe de déclaration :

Dim | Private | Public | global | Static V1 **As** Type1**,** V2 **As** Type2**,** …

Exemples :

```
Dim x As Integer, y As Single
Public S1  As String
Private S2 As String * 20
```

La différence entre les différents mots clés est expliquée dans le tableau suivant :

Mot clé	Désignation
Dim	Déclaration d'une variable globale ou locale à une procédure. Une variable globale déclarée avec Dim n'est pas accessible à partir des autres modules du projet (elle est privée).
Private	Déclaration d'une variable globale privée càd inaccessible à partir des autres modules du projet. Private agit comme Dim, seulement on ne peut pas déclarer une variable locale à l'aide de Private.
Public	Déclaration d'une variable globale publique càd *accessible* depuis les autres modules du projet. On ne déclare pas une variable locale avec Public.
Global	Le mot clé Global a le même effet que public, seulement il n'est utilisé que dans les modules (fichiers .bas)
Static	Utilisé dans une procédure ou une fonction pour déclarer une variable locale statique, càd qui garde ses valeurs d'un appel à l'autre.

2.1.3 Déclaration Implicite

On peut utiliser une variable sans déclaration. Elle prend alors le type **Variant**. Dans ce cas on peut lui affecter des valeurs de différents types.

Exemple :

```
V = "chaîne"
V = 20
V = "20" + "30"          '➔ V = "2030"
V = "20" + 30            '➔ V = 50
V = "20" & 30            '➔ V = "2030"
V = "6" * "3"            '➔ V = 18
```

On remarque que les conversions Chaîne → Numérique sont automatiques à condition que l'un des éléments de l'expression soit numérique et l'opérateur utilisé est un opérateur arithmétique. Le symbole + est à la fois un opérateur de concaténation de chaînes et un opérateur d'addition.

On remarque aussi que les conversions inverses (numérique → chaîne) sont automatiques. En effet puisque le symbole & est aussi un opérateur de concaténation de chaînes : l'expression "20" & 30 est considérée comme une concaténation de deux chaînes.

2.1.4 Imposer la déclaration Explicite

Il est parfois préférable d'exiger la déclaration de toutes les variables du programme. Pour cela on utilise l'instruction suivante au début du programme :

```
Option Explicit
```

De cette manière le compilateur signale une erreur pour toute utilisation de variable non déclarée explicitement.

2.1.5 Définition des constantes

```
Const C1 [As type1] = Val1, C2 [As type2] = Val2, …
```

La précision du type des constantes est optionnelle.

Exemple :

Const CX = 20

Const Pi As Single = 3.14, S = "Message:»

Remarque :

Une constante peut aussi être définie privée ou public en faisant précéder le mot clé Const par l'un des mots clés Private ou Public.

Exemple :

Public Const CP = 359

2.1.6 Déclaration des tableaux

Dim NomduTableau ([début1 To] fin1, [début2 To] fin2, …) **As** Type1**,** …

Remarque :

Le mot clé **Dim** est utilisé ici comme représentant de tous les mots clés de déclaration :

Dim | Private | Public | global | Static

Avec :

« $début_i$ » est l'indice du premier élément de la $i^{ème}$ dimension du tableau.

« fin_i » est l'indice du dernier élément de la $i^{ème}$ dimension.

Si on ne précise pas l'indice de début, il prend par défaut la valeur 0.

« $début_i$ » et « fin_i » peuvent prendre des valeurs négatives.

Exemples :

```
Dim Matrice (9, 19) As Integer
```

➔ Matrice d'entiers de 10 lignes (0 à 9) et 20 colonnes (0 à 19)

```
Dim T (1 to 50)
```

➔ Un tableau de variants indexé de 1 à 50

```
Dim TT (1 To 3,  9,  -10 To 10) As Single
```

➔ Déclaration d'un tableau tridimensionnel

Remarque :

L'accès aux éléments d'un tableau T est réalisé par la notation **T (i)**. S'il s'agit d'un tableau multidimensionnel, l'accès est réalisé avec : **T (i, j, ..)**

Exemple :

```
T(1) = 30
TT(1, 5, 0) = 6.5
```

2.1.7 Définition d'un type enregistrement

La définition d'un enregistrement est réalisée à l'aide du mot clé **Type** :

```
Type NonDuTypeEnegistrement
     Champ1  As Type1
```

```
    Champ2  As Type2
    …
    ChampN As TypeN
End Type
```

La déclaration ensuite d'une variable de ce type peut être réalisée comme suit :

```
    Dim E As NonDuTypeEnegistrement
```

Remarques :

1- Si on définit dans un module (fichier .bas) un type enregistrement, il devient accessible dans tous les autres modules (ou feuilles) de l'application.

2- Dans une feuille la définition d'un type enregistrement doit obligatoirement être réalisée d'une manière privée en faisant précéder le mot clé Type par **Private** :

```
Private Type NonDuTypeEnegistrement
    Champ1  As Type1
    Champ2  As Type2
    …
    ChampN As TypeN
End Type
```

Exemple :

```
    Private Type  Personne
            CIN  As String * 12
            Nom As String * 30
            Prenom As String * 20
            DN As Date
    End Type
    Dim P As Personne
```

Remarque :

L'accès à un champ d'enregistrement est réalisé par l'intermédiaire du sélecteur « **.** » :

```
Enregistrement.Champ
```

Exemple :

```
P.DN = #10/03/1980#
```

2.1.8 Définition des types énumérés

La définition d'un type énuméré consiste à définir un ensemble de constantes dont les valeurs commencent par défaut à partir de 0. La valeur de chaque constante est égale à la valeur de la constante précédente + 1.

La définition d'un type énuméré est réalisée à l'aide du mot clé **Enum** :

```
Enum NomDuTypeEnméré
    Constante 1 [ = valeur 1]
    Constante 2 [ = valeur 1]
```

```
…
End Enum
```

Si on ne précise pas de valeur pour une constante **Constante$_i$**, sa valeur est alors égale à 0 si elle est la première constante, sinon elle prend la valeur **Constante$_{i-1}$** + 1.

Exemple :

```
Enum Couleurs
    Black
    Bleue
    Green
    Cyan
    Yellow = 14
    White
End Enum
```

Les constantes auront alors respectivement les valeurs : 0, 1, 2, 3, 14 et 15

Remarque :

Un type énuméré peut être public ou privé.

2.2 Instructions Basic

(1) Le langage Visual

Basic utilise une *instruction par ligne*. Si on veut décomposer une même instruction sur deux lignes, on utilise le blanc souligné (_) à la fin de la première ligne pour indiquer qu'il y a une suite sur la ligne suivante.

Exemple :

```
X = Y + Z + T _
    + 20 – W
```

(2) La *dernière instruction* exécutée en Visual Basic est le mot clé **End**. Là ou on rencontre cette instruction (même dans une procédure), le programme est arrêté :

End

(3) Visual Basic donne la possibilité d'insérer des *commentaires* dans le programme. En effet un commentaire est une suite de caractère qui débute par le symbole quote (') et qui se termine à la fin de la ligne :

Exemple :

```
... ' Ceci est un commentaire
```

2.3 Les Opérateurs

2.3.1 Opérateurs arithmétiques

+	Addition

–	soustraction
*	Multiplication
/	Division réelle
\	Division entière
^	Puissance
Mod	Reste d'une division entière

2.3.2 Opérateurs Logiques

And	Et logique
Or	Ou logique
Xor	Ou bien
Not	Non logique

2.3.3 Opérateurs de relation

<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
<>	Différent
=	Egal

2.3.4 Opérateur d'affectation

On utilise le symbole **=**

2.3.5 Opérateurs de concaténation de chaînes

On utilise le symbole **&**

2.4 Structures de Choix

2.4.1 Instruction If

- Cas N° 1 :

If condition **Then** instruction1 [**Else** instruction2]

- Cas N° 2 :

```
If condition-1 Then
    Instruction-1
[ElseIf condition-2 Then
    Instruction-2
ElseIf ... ]
[Else
    instruction-n]
End If
```

2.4.2 Instruction Select Case

```
Select Case Expression
  Case cas1
        instruction1...
```

```
Case cas2
        instruction2
...
[Case Else
        instruction-n ]
End Select
```

Avec Cas_i pouvant être :

- une valeur
- séquence de valeurs : v1, v2, …
- un intervalle de valeurs : v1 To v2
- une comparaison : Is …

Exemple :

```
Case 1 To 4, 7 To 9, 11, 13, Is > 100
```

2.4.3 Fonction IIF

```
R = IIF(ExpLog, SiVrai, SiFaux)
```

La fonction IIF retourne la valeur du deuxième paramètre « SiVrai » dans le cas ou l'expression logique « ExpLog » est vérifiée sinon elle retourne la valeur du 3ème paramètre « SiFaux ».

2.4.4 Fonction Choose

```
R = Choose (Indice (1 ou 2…), Choix1, Choix2, …)
```

Si « Indice » a la valeur 1, la fonction Choose retourne Choix1,
Sinon, Si « Indice » a la valeur 2, la fonction Choose retourne Choix2, etc...
Exemple :

```
Dim T (1 to 5) as integer
...
X = Choose (i, T(1), T(2), T(3), T(4), T(5))
```

Cette écriture est équivalente à

```
X = T(i)
```

2.4.5 Fonction Switch

R = **Switch**(Condition1, Val1, Condition2, Val2, …)
La fonction **Switch** reçoit une suite de paramètres par paires.
Si la « condition1 » est vérifiée alors la fonction retourne « Val1 »,
Sinon, si la « condition2 » est vérifiée alors la fonction retourne « Val2 »,
etc.

2.5 Structures de boucles

2.5.1 La boucle For

```
For compteur = debut To fin [Step pas]
        instruction1
        instruction2
```

```
        …
        [Exit For]
        …
        instruction-n
Next
```

Le compteur est une variable numérique utilisée comme un compteur de boucle. Il est incrémenté à la fin de chaque itération de la boucle par « pas » (compteur = compteur + pas). Si on ne précise pas le pas d'incrémentation, le compteur est incrémenté par 1. L'instruction « **Exit For** » peut être utilisée pour forcer la sortie de la boucle si une condition particulière est réalisée.

Exemples :

```
Dim S As String
For i = 0 To 4
        S = S & i
Next
```

➔ S = "01234"

```
Dim S As Variant
For i = 0 To 4
        S = S + i
Next
```

➔ S = 10

2.5.2 La boucle While

```
While condition
        instruction1
        instruction2
        …
        instruction-n
Wend
```

Exemple :

La boule suivante détermine la première puissance de 2 supérieure ou égale à 1000

```
X = 1
While X < 1000
        X = X * 2
Wend
```

2.5.3 La boucle « Do While | Until … [Exit Do] … Loop »

- *Cas N° 1 :* Tant qu'une condition est vraie on répète les instructions

```
Do While condition
        instruction1
        …
        [Exit Do]
        …
        instruction-n
Loop
```

- *Cas N° 2 :* Les instructions sont répétées jusqu'à ce qu'une condition soit vraie. Et on commence par le test de la condition.

```
Do Until condition
        instruction1
        …
        [Exit Do]
        …
        instruction-n
Loop
```

2.5.4 La boucle « Do … [Exit Do] … Loop While | Until »

- *Cas N° 1 :* On répète les instructions tant qu'une condition est vraie. Le test est réalisé cette fois-ci à la fin et non au début. Càd la boucle est exécutée au moins une fois.

```
Do
        instruction1
        …
        [Exit Do]
        …
        Instruction-n
Loop While condition
```

- *Cas N° 2 :* Les instructions sont répétées jusqu'à ce qu'une condition soit vraie. De la même manière, le test est réalisé à la fin et non au début. Càd aussi que la boucle est exécutée au moins une fois.

```
Do
        instruction1
        …
        [Exit Do]
        …
        instruction-n
Loop Until condition
```

2.5.5 La boucle For Each

```
For Each element In group
        instruction1
        instruction2
        …
        [Exit For]
        …
        instruction-n
Next
```

Les instructions sont répétées pour chaque élément du groupe. Le groupe peut être un tableau ou une *collection* (voir plus loin dans le cours). La variable élément permet de parcourir la liste des éléments du groupe. A chaque itération elle prend la valeur du prochain élément du groupe.

Exemple :

```
    Dim T (5) As Integer
    Dim S as Integer
For i = 0 To 5
   T(i) = i * 2
Next
For Each X In T
   S = S + X
Next
```

2.6 Fonctions de gestion des chaînes de caractères

Fonction ou opérateur	***Désignation et Exemples***
&, +	Opérateurs de Concaténation. Il est préférable d'utiliser le symbole &.
	S = "Visual " & "Basic" ➔ S = "Visual Basic"
Asc(C)	Retourne le code Ascii du caractère C.
	Code = Asc("A") ➔ Code = 65
Chr(N)	Retourne le caractère de code Ascii N.
	C = Chr(65) ➔ C = "A"

InStr(S1, S2) , ou encore InStr(Pos, S1, S2 [, Maj])	Teste si la chaîne S2 est incluse dans S1 et retourne la position de la première occurrence de S2 dans S1 (à partir de 1). Si aucune, alors elle retourne 0 ou Nulle si l'une des chaînes a la valeur Nulle. Pos indique la position de début de recherche dans la chaîne S1 (pos doit être >=1). Maj est un paramètre optionnel ayant une valeur par défaut égale à 0. Si on veut une recherche sans faire de distinction entre majuscule et minuscule on utilise ce paramètre avec la valeur 1.
	Pos = Instr(3, "AB**CDEFGE**", "e", **1**) ➔ Pos = 5 Pos = Instr(1, "**ABCDEFGE**","e", **0**) ➔ Pos = 0 Pos = Instr("**ABCDEFGE**","W") ➔ Pos = 0
LCase(S)	Convertit une chaîne de caractères S en minuscule
	S1 = "Visual BASIC" S2 = Lcase(S1) ➔ S2 = "visual basic"
Left(S,N)	Extrait la sous chaîne gauche de S de longueur N.
	S1 = "Visual Basic" S2 = Left(S1, 6) ➔ S2 = "Visual"
Len(S)	Détermine la longueur de la chaîne S
	S1 = "Visual BASIC" L = Len(S1) ➔ L = 12
LSet S2 = S1	Aligne les caractères de la chaîne S1 à gauche dans S2 et complète par des espaces à droite suivant la longueur de S2. Si S2 est une chaîne de longueur non fixe ou non déclarée (Variant) alors elle doit obligatoirement avoir été initialisée à une chaîne qui fixe sa longueur. Si ce n'est pas le cas le résultat est une chaîne vide.
	Dim S1 as String * 10, S2 as String LSet S1 = "abc" ➔ S1 = "abc " LSet S2 = "abc" ➔ S2 = "" S2 = "12345678" LSet S2 = "abc" ➔ S2 = "abc "
LTrim(S)	Construit et retourne une copie de la chaîne S sans les espaces à gauche.
	S1 = " Visual Basic " S2 = LTrim(S1) ➔ S2 = "Visual Basic "
Mid(S, Pos) = …	Remplace dans la chaîne S à partir du caractère numéro Pos.
	S = "Bonjour" Mid(S, 4) = "soir" ➔ S = Bonsoir Mid(S, 4) = "**S**" ➔ S = Bon**S**oir
Mid(S, Pos, [N])	Extrait une sous chaîne de longueur N à partir de la position Pos de la chaîne S. Si on ne précise pas N ou si le nombre de caractères de S à partir de Pos est inférieur à N, alors tous les caractères de S à partir de Pos jusqu'à la fin sont extraits.

	S1 = "Bonjour" S2 = Mid(S1, 4, 2) ➔ S2 = "jo" S3 = Mid(S1, 4) ➔ S3 = "jour"
Right(S,N)	Extrait la sous chaîne droite de S de longueur N.
	S1 = "Visual Basic" S2 = Left(S1, 5) ➔ S2 = "Basic"
RSet S2 = S1	Aligne les caractères de la chaîne S1 à droite dans S2 et complète par des espaces à gauche suivant la longueur de S2. Si S2 est une chaîne de longueur non fixe ou non déclarée (Variant) alors elle doit obligatoirement avoir été initialisée à une chaîne qui fixe sa longueur. Si ce n'est pas le cas le résultat est une chaîne vide.
	Dim S1 as String * 10, S2 as String RSet S1 = "abc" ➔ S1 = " abc" LSet S2 = "abc" ➔ S2 = "" S2 = "12345678" LSet S2 = "abc" ➔ S2 = " abc"
RTrim(S)	Construit et retourne une copie de la chaîne S sans les espaces à droite.
	S1 = " Visual Basic " S2 = RTrim(S1) ➔ S2 = " Visual Basic"
Space(N)	Construit une chaîne de N espaces
	S = Space(10) ➔ S = " "
Str(N)	Convertit le nombre N (entier ou réel) en une chaîne de caractères. Si le nombre N est positif, la chaîne renvoyée contient un espace supplémentaire à gauche.
	S = Str(256) ➔ S = " 256"
String(N, C)	Construit une chaîne contenant N fois le caractère C.
	S = String(10, "A") ➔ S = "AAAAAAAAAA"
Trim(S)	Construit et retourne une copie de la chaîne S sans les espaces à gauche et à droite (élimine les espaces à gauche et à droite).
	S1 = " Visual Basic " S2 = Trim(S1)➔ S2 = "Visual Basic"
UCase(S)	Convertit une chaîne de caractères S en majuscule
	S1 = "Visual Basic" S2 = Ucase(S1) ➔ S2 = "VISUAL BASIC"
Val(S)	Retourne la valeur numérique contenue dans la chaîne S. la fonction Val arrête la traduction au premier caractère de la chaîne ne faisant pas partie des nombres. La virgule n'est pas reconnue comme éléments des nombres. Cependant les représentation Octales (&O…) et Hexadécimales (&H…) sont acceptées.

	X1 = Val("20.5") ➔ X1 = 20,5 X2 = Val(" 20.5 ") ➔ X2 = 20,5 X3 = Val("20,5") ➔ X3 = 20 X4 = Val("2abc") ➔ X4 = 2 X5 = Val("abc") ➔ X5 = 0 X6 = Val("&H00FF") ➔ X6 = 255

2.7 Les Tableaux

2.7.1 Tableaux statiques

Nous avons déjà vu comment déclarer un tableau statique. Càd les dimensions ainsi que le nombre d'éléments sont fixé :

Dim NomDuTableau ([D1 To] F1, …, [Dn to] Fn) **As** TypeDesElements

Le mot clé **Dim** pouvant être remplacé par l'un des mot : **Private | Public | global | Static**

L'accès à un élément du tableau est réalisé à l'aide de l'écriture suivante :

NomDuTableau (i_1, i_2, …, i_n)

Exemples :

```
T(1) = 20                 'Pour un tableau d'entiers à une dimension
T(2, 3) = 53.6    'Pour un tableau de réels à deux dimensions
```

Cependant, il est possible en Visual Basic de déclarer des tableaux dynamiques. La réservation de la mémoire nécessaire sera réalisée au fur et à mesure des besoins.

2.7.2 Tableaux dynamiques

L'utilisation d'un tableau dynamique nécessite deux étapes :

1- Pendant une première étape on déclare un tableau sans préciser le nombre de dimensions ou celui des éléments. Cette opération est réalisée de la manière suivante :

Dim NomDuTableau ( ) [as TypeDesElements]

Si on ne précise pas le type des éléments, ils seront alors « Variant »

Cette instruction déclare un tableau, mais elle ne lui réserve aucun espace mémoire.

2- Dans la deuxième étape, on a une idée sur le nombre de dimensions et celui des éléments dont on a besoin. On peut alors faire une allocation suivant nos besoins. Il s'agit d'une opération de redimensionnement réalisée de la manière suivante :

ReDim [**Preserve**] NomDuTableau ([D1 **To**] F1**,** [D2 **To**] F2**...** [Dn **To**] Fn)

Cette fois-ci on précise les dimensions et le nombre d'éléments de chaque dimension.

Le mot clé optionnel **Preserve** est utilisé si on veut conserver les données déjà existantes dans le tableau. Cela signifie qu'on a déjà fait un appel à Redim auparavant.

Les **contraintes d'utilisation du mot clé Preserve** sont les suivantes :

- On ne peut changer que la borne supérieure (l'indice maximal) :

Exemple :

```
Dim T() as Integer
```

```
ReDim T(5)
…
ReDim Preserve T(3)          ➔ correcte
…
ReDim Preserve T(9)          ➔ correcte
…
ReDim Preserve T(1 to 5)     ➔ incorrecte
```

- On ne peut pas modifier le nombre de dimensions :

```
Dim T() as Integer
ReDim T(5,6)
…
ReDim Preserve T(5, 6, 3)    ➔ incorrecte
```

- Dans le cas d'un tableau à plusieurs dimensions, on ne peut modifier que le nombre d'éléments de la dernière dimension

```
Dim T() as Integer
ReDim T(5, 6)
…
ReDim Preserve T(5, 12)      ➔ correcte
…
ReDim Preserve T(7, 15)      ➔ incorrecte
```

Remarques :

1- lorsqu'on termine de se servir d'un tableau dynamique T, on peut libérer l'espace mémoire qui lui a été allouée la dernière fois à l'aide de l'instruction **Erase** :

```
Erase(T)
```

2- Il est toujours possible après un appel à Erase de redimensionner de nouveau le tableau dynamique.

3- Pour connaître l'indice minimal (respectivement maximal) d'un tableau dynamique (ou aussi statique) on utilise la fonction **LBound** (respectivement **UBound**) :

```
Min = LBound(T [, dimension])
Max = UBound(T [, dimension])
```

Le deuxième paramètre n'est utilisé que si le tableau a plus qu'une dimension. Il sera alors utilisé pour déterminer l'indice minimal ou maximal d'une dimension bien précise :

Exemple :

```
Redim T1(5)
Redim T2(4, 6)
X1 = UBound (T1)      ➔ X1 = 5
X2 = UBound (T2, 2)   ➔ X2 = 6
M2 = LBound (T2, 1)   ➔ M2 = 0
```

2.7.3 Variant contenant un Tableau

Il est possible de remplir une variable de type Variant par des éléments d'un tableau. La variable devient alors comme un tableau. Cette opération est réalisée à l'aide de la fonction **Array** :

```
Dim T As Variant
T = Array (V1, V2, …, Vn)
```

V1, V2, … et Vn sont les éléments du tableau. Ils peuvent être de n'importe quel type et de types variés.

Exemples :

```
Dim T1, T2, T3
T1 = Array(10, 20, 30, 40, 50)              ➔ Tableau d'entiers
T2 = Array(20, "abd", 2.5, #12/20/68#)      ➔ Tableau d'éléments de
                                                types variés
T3 = Array(Text1, Text2, Text3)             ➔ Tableau d'éléments de
                                                type TextBox
For i = LBound(T1) To UBound(T1)
  S = S + T1(i)                             ➔ Somme des éléments du tableau
Next

For i = LBound(T3) To UBound(T3)
      Text4.Text = Text4.Text  & T3(i).Text  ➔ Concaténation des
                                                textes
Next
```

2.8 Opérateurs supplémentaires

2.8.1 L'opérateur Like

Utilisé pour comparer une chaîne avec un modèle :

```
If (chaîne Like modèle) then …
```

Le modèle est une chaîne de caractère contenant des symboles génériques tels que « ?, *, #, … » permettant ainsi de décrire un modèle pour un ensemble de chaînes bien déterminé. Par exemple pour décrire une suite de caractères qui commence par une lettre majuscule on utilise le modèle : "[A-Z]*"

Les symboles pouvant être utilisés dans le modèle sont les suivants :

Symbole	désignation
?	Représente un caractère quelconque.
*	Représente 0, 1 ou plusieurs caractères.
#	Signifie un chiffre.
[$C_1C_2…C_n$]	*Signifie l'un des caractères : C_1 ou C_2 … ou C_n.* *Exemple :* Tester si S est une chaîne de 3 caractères qui commence par l'une des lettres : A, B ou C : If (S Like "[ABC]??") then ...

[!$C_1C_2…C_n$]	Signifie tous caractère sauf l'un des caractères : C_1 ou C_2 … ou C_n. Exemple : Tester si S est une chaîne qui ne commence pas par l'une des lettres : X, Y, Z ou T : If (S Like "[!XYZT]*") then ...
[C_1–C_2]	Désigne la plage des caractères compris entre C_1 et C_2 . On peut spécifier plusieurs plages de caractères dans le même groupe entre crochets sans utiliser de délimiteurs entre les plages. Pour représenter l'ensemble des caractères majuscules et minuscules plus les caractères « é è à » on utilise la notation suivante : "[A-Za-Zéèà]" Si on commence l'ensemble par un point d'exclamation : "[!A-Za-Zéèà]", cela signifie tous les caractères n'appartenant pas à l'ensemble "[A-Za-Zéèà]".

Remarque :

Si on veut utiliser l'un des caractères spéciaux (?, #, *, [ ) comme caractère ordinaire dans le masque, il suffit de l'entourer par des crochets (exemple : [*]). Le crochet fermant (]) ne peut pas être entouré de crochets, mais il peut être utilisé à l'extérieur. Pour le cas du point d'exclamation, il n'a de fonction spéciale qu'à l'intérieur des crochets. Si on l'utilise à l'extérieur il est lu comme caractère ordinaire. Le trait d'union (–) peut être utilisé au début d'un ensemble entre crochet ou à la fin pour correspondre à lui-même.

Exemple :

"[-a-z]" : signifie toutes les lettres minuscules ou le trait d'union.

2.8.2 Détermination du type d'une variable

2.8.2.1 L'opérateur TypeOf

L'opérateur TypeOf est utilisé pour déterminer le type d'un objet et non d'une variable ordinaire.

```
If (TypeOf Objet is TypeObjet) then …
```

Exemple 1 :

```
If (TypeOf X is TextBox) then
    X.Text = "ceci est du texte"
End If
```

Exemple 2 :

Compter le nombre de boutons d'une feuille (Form1) :

```
Dim x As Object, cb as Integer
cb = 0
For Each x In Form1.Controls
If TypeOf x Is CommandButton Then cb = cb + 1
Next
```

2.8.2.2 La fonction TypeName

C'est une fonction qui retourne sous forme de chaîne de caractères le type d'un objet ou d'une variable quelconque :

```
S = TypeName (Variable)
```

Exemple :

```
Dim x As Integer
S = TypeName(x)        ➔ "Integer"
```

2.8.3 Fonction Format

Retourne une chaîne de caractères contenant une expression formatée en fonction des instructions contenues dans l'expression de mise en forme.

Syntaxe générale :

```
Résultat = Format (Expression [, "Chaîne de mise en forme"] )
```

Remarque :

Si on n'utilise pas la chaîne de mise en forme, la fonction se comporte comme **str** lorsqu'elle est appliquée à une expression numérique. Seulement elle n'ajoute pas comme str l'espace à gauche et elle convertit e point décimal des réels en une virgule :

Exemple :

S = Format(20.5) ➔ S = "20,5"

La chaîne de mise en forme contient un ensemble de Formateurs permettant de configurer l'expression. Certains de ces formateurs sont les suivants :

dd , mm , yy , hh , mm , ss , \ , < , > , # , . , %

Formateur	***Désignation***
h ou hh	Signifie heure
m ou mm	Signifie Minutes lorsqu'il est utilisé avec une expression de type heure (Time). Avec une date, il signifie mois.
s ou ss	Secondes *Exemple :* S = Format(Time, "h:m:s") ➔ "22:5:8". S = Format(Time, "hh:mm:ss") ➔ "22:05:08".
d, dd, ddd, dddd	Signifient jour (day) : d : donne les jours (< à 10) sur un caractère. Exemple : 4 dd : sur deux caractères en complétant par 0 à gauche. Exemple : 04 ddd : jour de la semaine en abrégé. Exemple : mar (au lieu de mardi) dddd : jour de la semaine complet. Exemple : mardi
m, mm, mmm, mmmm	m et mm : donnent le mois respectivement sur 1 ou 2 caractères en complétant par un 0 à gauche. mmm : donne la désignation du mois en abrégé : avr (pour avril)

	mmmm : donne la désignation complete du mois : avril
yy ou yyyy	Donnent l'année sur 2 ou 4 caractères.
#	Représente un chiffre. *Exemple :* Format (52.442, "####.##") ➔ 52,44 Format (52.442, "0####.##") ➔ 00052,44 en complétant à gauche par des 0. Format(52.4, "##,##.##") ➔ 52,4 Format(52.4, "##,##.#0") ➔ 52,40
La virgule : ,	Là ou en met la virgule au milieu, elle signifie de décomposer les nombre en milliers (3 chiffres par 3 espacés) Format(5852.442, "##,##.##") ➔ 5 852,44
%	Convertie un nombre en pourcentage : *Exemple :* S = Format(0.2, "0.00%") ➔ "20,00%".
<	Convertit une chaîne en minuscule : *Exemple :* S = Format("VISUAL BASIC", "<") ➔ "visual basic"
>	Convertit une chaîne en majuscule : *Exemple :* S = Format ("à bientôt", ">") ➔ "À BIENTÔT"

2.9 Procédures et Fonctions

2.9.1 Les procédures

2.9.1.1 Définition d'une procédure

Syntaxe :

```
Sub NomProcedure ( [P1 As type1, P2 As Type2, …] )
        Instruction-1
        … [Exit Sub]
        Instruction-n
End Sub
```

Remarque :

Sub peut être précédé par l'un des mots clés [**Private | Public**] et aussi par [**Static**].

Private : La procédure est inaccessible depuis les autres modules

Public : La procédure est accessible depuis tous les autres modules de l'application. il s'agit du mode par défaut lorsqu'on ne met rien à gauche de Sub.

Static : Signifie que toutes les variables locales de la procédure sont statiques.

Exemple :

```
Private Static Sub Plus ()
   Dim X As Integer
   X = X + 1
   Text1.Text = X
End Sub
```

➔ La procédure Plus est inaccessible depuis les autres modules. A chaque fois qu'elle est appelée une nouvelle valeur est affichée dans le TextBox (1, 2, 3 …)

2.9.1.2 Appel à une procédure

Une procédure P à n paramètres est appelé à l'aide de l'instruction suivante :

P arg_1, arg_2, … , arg_n

arg1, arg2, … sont les arguments d'appel

si la procédure est sans paramètres. L'appel est réalisé comme suit :

P

On remarque que les parenthèses ne sont pas utilisées pour l'appel des procédures en Visual Basic

2.9.2 Les fonctions

2.9.2.1 Définition d'une fonction

Syntaxe :

```
Function NomFonction ( [P1 As type1, P2 As Type2, …] ) As Type Fonction
        Instruction-1
        …
        NomFonction = Expression
        …       [Exit Function]
        Instruction-n
End Function
```

Remarques :

1- De la même manière que pour les procédure, le mot clé **Function** peut être précédé par l'un des mots clés [**Private | Public**] et aussi par [**Static**].

2- Le type de la fonction est le type de la valeur de l'expression retournée par la fonction.

Exemple :

```
Function Suc (X As Integer) As Integer
        Suc = X + 1
End Sub
```

2.9.2.2 Appel à une fonction

Une fonction F à n paramètres est appelée à l'aide de l'instruction suivante :

X = F(arg$_1$, arg$_2$, … , arg$_n$)

arg1, arg2, … sont les arguments d'appel

si la fonction est sans paramètres. L'appel est réalisé comme suit :

X = F() ou **X = F**

2.9.3 Passage des paramètres

2.9.3.1 Passage par adresse

Afin de sortir de la procédure ou de la fonction avec les changements appliquées aux paramètres il est nécessaire de faire un passage par adresse ou par référence. Le type de passage des paramètres **par défaut** de Visual Basic est **le passage par adresse**. Celui-ci peut être obtenu explicitement par le mot clé **ByRef** qui précède le nom du paramètre :

```
Dim | Function NomPF( …, ByRef  Pi  As  Typei, …)
        Instruction-1
        …
        Instruction-n
End Sub | Function
```

Tout changement du paramètre Pi sera alors récupéré à l'extérieur.

Exemple :

```
Sub Inc(ByRef x As Integer)
   x = x + 1
End Sub
```

⇔

```
Sub Inc(x As Integer)
   x = x + 1
End Sub
```

```
Dim a as Integer
a = 20
Inc a   ➔  a = 21
```

2.9.3.2 Passage par valeur

Lorsqu'on utilise un passage par valeur pour un paramètre d'une fonction ou un procédure, la valeur d'entrée du paramètre est conservée après l'exécution.

Le passage par valeur en Visual Basic doit être spécifié explicitement par l'intermédiaire du mot clé **ByVal**.

```
Dim | Function NomPF( …, ByVal  Pi  As  Typei, …)
        Instruction-1
        …
        Instruction-n
End Sub | Function
```

Exemple :

```
Sub Inc(ByVal x As Integer)
```

```
    x = x + 1
End Sub
```

```
Dim a as Integer
a = 20
Inc a    ➔  a est toujours = 20
```

2.9.4 Paramètres optionnels

Il est possible de réaliser des procédures ou des fonctions pour lesquelles certains paramètres ont des valeurs par défaut. Lors de l'appel, il devient possible d'omettre ces paramètres qui prendront alors leurs valeurs par défaut. Ce type de paramètre peut être réalisé à l'aide du mot clé **Optional** :

```
Sub | Function NomPF( …, Optional  Pi  As  Typei = Valeuri, …)
                Instruction-1
                …
                Instruction-n
End Sub | Function
```

$Valeur_i$ étant la valeur par défaut du paramètre Pi dans le cas où il manque lors de l'appel.

Remarques :

1- On ne peut pas mettre un paramètre optionnel entre deux paramètres non optionnels ou inversement. Lorsqu'on définie un paramètre optionnel, tous les paramètres suivants doivent obligatoirement être optionnels.

2- Il est possible de combiner **Optional** avec l'un des mots clés **ByRef** ou **ByVal**.

Exemple :

```
Function Dupliquer(S As String, Optional N As Integer = 2,_
                                Optional Majuscule As Boolean = False) As String
   Dim R As String
   R = ""
   For i = 1 To N
      R = R & S
   Next
   If Majuscule Then
      Dupliquer = UCase(R)
   Else
      Dupliquer = R
   End If
End Function
Sub Appels()
   S1 = Dupliquer("abCd")              '➔      abCdabCd
```

```
S2 = Dupliquer("abCd", 3)              '➔  abCdabCdabCd
S3 = Dupliquer("abCd", 1, True)  '➔  ABCD
S4 = Dupliquer("abCd", , True)    '➔  ABCDABCD
End Sub
```

2.9.5 Liste d'arguments variables

Les procédures (ou fonctions) avec liste d'arguments variables sont des procédures qui peuvent être appelé à chaque fois avec un nombre de paramètre différents. Le nombre de paramètres n'est alors pas limité. A chaque appel, suivant le besoin on appelle la procédure avec le nombre de paramètres nécessaire. Ce type de procédures peut être réalisé avec le mot clé **ParamArray** à l'aide de la syntaxe suivante :

```
Dim | Function NomPF( …, ParamArray  Pi() )
        Instruction-1
        …
        Instruction-n
End Sub | Function
```

Remarques :

1- Le paramètre Pi est obligatoirement un tableau de Variant.

2- Aucun paramètre ne doit être déclaré après le paramètre Pi.

Exemple :

```
Function Somme(ParamArray t()) As Integer
    Dim S As Integer
    n1 = LBound(t)
    n2 = UBound(t)
    For i = n1 To n2
        S = S + t(i)
    Next
    Somme = S
End Function
```

Les appels à la fonction peuvent être réalisés comme suit :

```
    S1 = Somme(1, 2, 3)                          '➔ S1 = 6
S2 = Somme(10, 20)                          '➔ S2 = 30
S3 = Somme(10, 11, 12, 13, 14, 15, 16, 17, 18)  '➔ S3 = 126
```

2.9.6 La récursivité

La récursivité est réalisée lorsqu'une procédure ou une fonction fait appel à elle même pour répéter un même travail plusieurs fois.

Exemple :

```
Function Fact (n As Integer) As Integer
   If n = 1 Then
      Fact = 1
   Else
      Fact = n * Fact(n - 1)
   End If
End Function
```

2.10 Les Fichiers

2.10.1 Fichiers textes

2.10.1.1 Ouverture en Ecriture

```
      Open Nom_physique for Output as [#] Numéro
```

Crée le fichier si inexistant, sinon elle l'écrase.

Exemple :

```
      Open "test01.txt" for Output as 1
ou    Open "test01.txt" for Output as #1
```

2.10.1.2 Ouverture en Lecture

```
      Open Nom_physique for Input as [#]Numéro
```

Exemple :

```
      Open "c:\autoexec.bat" for Input as 2
ou    Open "c:\autoexec.bat" for Input as #2
```

2.10.1.3 Ouverture en Ajout

```
      Open Nom_physique for Append as [#]Numéro
```

Exemple :

```
      Open "c:\autoexec.bat" for Append as 3
```

10.1.4 Fermeture

```
      Close [#] Numéro1, [#] Numéro2, …
```

Exemple :

```
      Close 1, 2
```

2.10.1.5 Procédures d'Ecriture

2.10.1.5.1 Procédure Print #

Print #Numéro**,** Exp1**,** Exp2**,** … [**;**]

Les expressions sont imprimées dans le fichier séparé par des tabulations.

Si le point virgule n'est pas utilisé, un retour chariot est imprimé à la fin de l'écriture.

Sinon la prochaine écriture sera réalisée sur la même ligne.

Exemple :

```
X=20
Y= "test"
Z=19.5
Print #1, X, Y, Z
Print #1, "texte" ;
Print #1, "Suite de la ligne"
```

➔
```
20              test            19,5
texteSuite de la ligne
```

2.10.1.5.2 Procédure Write #

Write #Numéro**,** Exp1**,** Exp2**,** … [**;**]

Même fonctionnement que le Print, mais sépare les expressions par des virgules, et délimite les chaînes de caractères par des guillemets.

Exemple :

```
X=20
Y= "test"
Z=19.5
Write #1, X,Y,Z
Write #1, "texte" ;
Write #1, "Suite de la ligne"
```

➔
```
20,"test",19.5
"texte","Suite de la ligne"
```

2.10.1.6 Procédures de Lecture

2.10.1.6.1 Procédure Input #

Input #Numéro**,** V1**,** V2**,** …

Suppose que l'écrire à été réalisé par Write. Càd : elle cherche les délimiteurs (, et ").

La lecture des numériques ne pause pas de problème. Mais celle des chaîne n'est pas limitée par des espaces, mais des virgules ou des guillemets.

2.10.1.6.2 Procédure Line Input #

Line Input #Numéro**,** variable

Permet la lecture de toute une ligne sous forme de texte.

2.10.1.6.3 Fonction Input

String **Input** (NombreDeCaractères, [**#**]Numéro)

Lit un nombre de caractères donnés et retourne la chaîne lue

Exemple :

```
Text = Input(1000, 1) ; ' lecture de 1000 caractères à partir du fichier d'identificateur #1
```

2.10.1.7 Test de fin de fichier

EOF (Numéro)

Exemple :

```
While Not (EOF(1)) …
```

10.1.8 Taille d'un fichier

LOF(Numéro)

Exemple : Lecture d'un fichier texte par une seule opération

```
Open "c:\autoexec.bat" for Input as 1
L = LOF(1)
Text = input(L, 1)
```

2.10.2 Fichiers Binaires

2.10.2.1 Ouverture en Ecriture

Open Nom_physique **for Binary Access Write as** Numéro

Exemple :

```
Open "test01.bin" for Binary Access Write as 1
```

N'écrase pas le fichier si déjà existant.

2.10.2.2 Ouverture en Lecture

```
Open Nom_physique for Binary Access Read as Numéro
```

Exemple :

```
Open "test01.bin" for Binary Access Read as 2
```

2.10.2.3 Ouverture en Ajout

```
Open Nom_physique for Binary Access Write as Numéro
Seek #Numéro, LOF(Numéro) + 1
```

2.10.2.4 Ecriture dans un fichier binaire

```
Put #Numéro, [Position] , Variable
```

Exemple :

```
X=20
Put #1, ,X
```

2.10.2.5 Procédure de lecture

```
Get #Numéro, [Position] , Variable
```

2.10.2.6 Recherche directe

```
Seek #Numéro, Position
```

2.10.2.7 Lecture de la position courante

```
Pos = Seek (Numéro)
Ou encore : Pos = Loc(Numéro)
```

2.10.3 Fichiers d'enregistrements

2.10.3.1 Ouverture en Ecriture

```
Open Nom_physique for Random Access Write as Numéro Len = Len(Enreg)
```

2.10.3.2 Ouverture en lecture

```
Open Nom_physique for Random Access Read as Numéro Len = Len(Enreg)
```

2.10.2.3 Ouverture en Ajout

```
Open Nom_physique for Random Access Write as Numéro Len = Len(Enreg)
Seek #Numéro, LOF(Numéro)/Len(Enreg) + 1
```

Remarques :

1- Un fichier d'enregistrements est géré comme un fichier binaire.

2- Un fichier binaire ou d'enregistrements peut être lu bloc par bloc (ou écrit bloc par bloc) à l'aide des tableaux :

Exemple :

```
Dim T(1 to 20) as Integer
Get #1,, T
Put #1,, T
```

Table des Matières

2.1 Déclaration des variables _____ 1
2.1.1 Types de données 1
2.1.2 Déclaration des variables..... 1
2.1.3 Déclaration Implicite..... 2
2.1.4 Imposer la déclaration Explicite..... 2
2.1.5 Définition des constantes 2
2.1.6 Déclaration des tableaux 3
2.1.7 Définition d'un type enregistrement 3
2.1.8 Définition des types énumérés 4

2.2 Instructions Basic _____ 5

2.3 Les Opérateurs _____ 5
2.3.1 Opérateurs arithmétiques..... 5
2.3.2 Opérateurs Logiques 6
2.3.3 Opérateurs de relation 6
2.3.4 Opérateur d'affectation 6
2.3.5 Opérateurs de concaténation de chaînes..... 6

2.4 Structures de Choix_____ 6
2.4.1 Instruction If..... 6
2.4.2 Instruction Select Case 6
2.4.3 Fonction IIF..... 7
2.4.4 Fonction Choose..... 7
2.4.5 Fonction Switch..... 7

2.5 Structures de boucles _____ 7
2.5.1 La boucle For 7
2.5.2 La boucle While 8
2.5.3 La boucle « Do While | Until … [Exit Do] … Loop »..... 9
2.5.4 La boucle « Do … [Exit Do] … Loop While | Until »..... 9
2.5.5 La boucle For Each 10

2.6 Fonctions de gestion des chaînes de caractères_____ 10

2.7 Les Tableaux _____ 13
2.7.1 Tableaux statiques..... 13
2.7.2 Tableaux dynamiques..... 13
2.7.3 Variant contenant un Tableau 15

2.8 Opérateurs supplémentaires _____ 15
2.8.1 L'opérateur Like..... 15
2.8.2 Détermination du type d'une variable..... 16
2.8.3 Fonction Format 17

2.9 Procédures et Fonctions _____ 18
2.9.1 Les procédures..... 18
2.9.2 Les fonctions 19
2.9.3 Passage des paramètres 20
2.9.4 Paramètres optionnels 21
2.9.5 Liste d'arguments variables 22

2.9.6 La récursivité 22

2.10 Les Fichiers 23
2.10.1 Fichiers textes 23
2.10.2 Fichiers Binaires 25
2.10.3 Fichiers d'enregistrements 26