

FICHE TECHNIQUE N° II

- Chapitre : PRINCIPE DE PROGRAMMATION -

LES VARIABLES

Une variable est une donnée identifiable par un nom unique. Cette donnée va contenir une valeur d'un certain type. Au cours de l'exécution d'un programme, la valeur d'une variable peut être modifiée une ou plusieurs fois. Lors de la déclaration d'une variable, il faut préciser à quelle type elle appartient, c'est à dire que l'on va définir le type de données qu'elle va contenir.

Il faut savoir qu' en Visual Basic, la déclaration des variables est facultative. Cependant, pour les puristes de la programmation, une instruction permet de forcer la déclaration des variables. Pour cela, il faut ajouter dans l'éditeur de code, à la première ligne du module général, l'instruction "option explicit". Dès cet instant toute variable non déclarée sera interprétée par le compilateur comme une erreur de programmation.

En VB une variable se déclare de la manière suivante :

Dim "Nom de la variable" **as** "Type"

Exemple : **Dim** Nom **as** String

Dim Age **as** Integer

Les types de variable :

| Type | Désignation |
|----------|--|
| Integer | Entier |
| Single | Flottant avec précision de 7 chiffres |
| Long | Entier long |
| Double | Flottant avec précision de 16 chiffres |
| String | Chaîne de caractères |
| Byte | Entier |
| Date | Date |
| Currency | Entier très grand |
| Boolean | Booléen |



Il existe également le type "Variant" qui peut contenir des données de tous types. Ce type est très utile lorsque l'on ne connaît pas à l'avance le type des données qui seront contenues dans la variable ou si le type peut varier lors de l'exécution du programme.

Les variables peuvent avoir des portées différentes. Elles sont soit locales (si elles sont déclarées à l'intérieur d'une procédure ou d'une fonction), soit globales (si elles sont déclarées dans le module général en dessous de l'instruction "option explicit" par exemple.

Une variable statique est une variable dont le contenu n'est pas réinitialisée à chaque appel de la fonction ou de la procédure dans laquelle elle est contenue. L'initialisation d'une telle variable est laissée à la charge du programmeur pendant la durée de l'exécution du programme. Ce type de variable est très utile pour les incréments par exemple.

La variable statique se déclare de la manière suivante :

Static "Nom de la variable" **as** "Type"

Exemple : **Static** Compteur **as Integer**



LES CONSTANTES

Une constante est une donnée identifiable par un nom qui est unique et dont la valeur est définie à la déclaration. Cette valeur lui reste affectée pendant toute la durée de vie du programme et n'est pas modifiable lors de l'exécution contrairement à celle de la variable.

En VB la constante se déclare de la manière suivante :

Const "Nom de la constante" = "valeur de la constante"

Ex : **Const** Euro = 6.55957

Les constantes sont soumises au même titre que les variables à la règle des portées. Elles sont soit locales si elles sont déclarées à l'intérieur d'une procédure ou d'une fonction, soit globales si elles sont déclarées dans le module général du programme.

LES TABLEAUX

Les tableaux permettent de traiter des séries de valeurs, des vecteurs ou encore des matrices. Sous VB, il est possible d'utiliser des tableaux unidimensionnels (une dimension) ou multidimensionnelles (plusieurs dimensions). Lors de la déclaration de cette structure de données, il faut spécifier le type de ce tableau autrement dit le type des données que le tableau va contenir. Si les données sont de types différentes, il faudra utiliser le type "Variant" vu précédemment.

Déclaration d'un tableau unidimensionnel :

Dim "Nom du tableau" (Taille du tableau en nombre de cases) **as** "type"

Exemple : Dim Test (49) as Integer

Ici on a créé le tableau unidimensionnel "Test" comportant 50 cases.

Déclaration d'un tableau bidimensionnel :

Dim "Nom du tableau" (Nombre de cases de la première dimension, Nombre de cases de la deuxième dimension) **as** "type"

Exemple : Dim Matrice (2,2) as Integer

Ici on a créé le tableau à deux dimensions "Matrice". Chaque dimension comporte trois cases.

Déclaration d'un tableau tridimensionnel :

Dim "Nom du tableau" (Nombre de cases de la première dimension, Nombre de cases de la deuxième dimension, Nombre de cases de la troisième dimension) **as** "type"

Exemple : Dim Matrice2 (2,2,2) as Integer

Ici on a créé le tableau à trois dimensions "Matrice2". Chaque dimension comporte trois cases.

Les tableaux sont soumis au même titre que les variables et les constantes à la règle des portées. Ils sont soit locaux s'ils sont déclarés à l'intérieur d'une procédure ou d'une fonction, soit globaux s'ils sont déclarés dans le module général.

Remplissage d'un tableau :

Pour remplir un tableau, on affecte une valeur à chacune des cases du tableau à l'aide de l'opérateur "=".

Exemple : Matrice (1,2) = "Laurent"

On a rempli ici case 3 dimension 1

Matrice (2,2) = "DUPONT"

On a rempli ici case 3 dimension 2



LES INSTRUCTIONS

Une instruction est une ligne de code qui peut comporter des éléments syntaxiques, des opérateurs, des fonctions...

| | | |
|-----------|---------------------------|-----------------|
| Exemple : | Moncontrôle.Text = Nom | Instruction n°1 |
| | Debug.print = Date & Time | Instruction n°2 |
| | Compteur = Compteur + 1 | Instruction n°3 |

Un programme est une suite d'instructions. Chaque instruction est contenue dans une procédure ou une fonction ou bien encore dans une boucle ou un test. En Visual Basic, on ne peut écrire qu'une seule instruction par ligne dans l'éditeur de code. Cependant, si l'on souhaite écrire une instruction sur plusieurs lignes, il faut mettre le caractère "_" précédé d'un espace, entre les lignes composantes l'instruction.

Exemple : Textbox.text = "Cette instruction _
 est écrite sur plusieurs _
 lignes. »



LES BOUCLES (STRUCTURES ITERATIVES)

Les boucles sont des structures itératives qui permettent de répéter un certain nombre d'instructions tant que certaines conditions sont remplies. En Visual Basic il existe plusieurs type de boucle : la boucle "For ... Next"(en algorithmie il s'agit de la boucle "Pour »)
la boucle "While ... Wend"(en algorithmie c'est la boucle "Tant que")
la boucle "Do ... Loop"(en algorithmie c'est la boucle "Répéter jusque/tant que")

Dans ce chapitre, on fait appel à la notion de "condition". Elle utilise des opérateurs et peut nécessiter l'utilisation de parenthèses. Exemple de condition : (var > 1) signifie que la valeur de la variable "var" doit être supérieure à 1.

La boucle "For ... Next"(boucle Pour)

```
For "Nom variable"= "valeur minimale"to "valeur maximale »  
    Instruction 1  
    Instruction 2  
    .....  
    Instruction X  
Next
```

Remarques sur cette boucle : Les valeurs minimales et maximales sont soit des nombres ou des variables de type numérique. Il est possible de préciser le pas d'incrémentaion avec l'instruction "Step n"où n est le pas d'incrémentaion. On peut sortir prématurément de la boucle avec l'instruction "Exit For ». Les boucles "For... Next" peuvent être imbriqués.

La boucle "While ... Wend"(Boucle Tant que)

```
While "conditions"  
    Instruction 1  
    Instruction 2  
    .....  
    Instruction X  
Wend
```

Remarques sur cette boucle : La condition fait appel à l'utilisation des opérateurs. Les boucles "While... Wend"peuvent être imbriqués.



La boucle "Do ... Loop" (Boucle Répéter)

Do While/Until "condition"

Instruction 1

Instruction 2

.....

Instruction X

Loop

Remarques sur cette boucle : On utilisera "**Do while**" lorsque l'on exprimera un "répéter tant que" et on utilisera "**Do until**" lorsque l'on exprimera un "répéter jusqu'à". La condition est vérifiée au début de la boucle donc les instructions ne seront exécutées que si les conditions sont remplies. Cette boucle peut être imbriquée. On peut sortir prématurément de la boucle avec l'instruction "**Exit Do**".

Variante de la boucle "Do ... Loop" (Boucle Répéter)

Do

Instruction 1

Instruction 2

.....

Instruction X

Loop While/Until "condition"

Remarques sur cette boucle : On utilisera "**Loop while**" lorsque l'on exprimera un "répéter tant que" et on utilisera "**Loop until**" lorsque l'on exprimera un "répéter jusqu'à". L'avantage de cette variante de la boucle, c'est que comme la condition est vérifiée à la fin, les instructions sont exécutées au moins une fois. Cette boucle peut être imbriquée. On peut sortir prématurément de la boucle avec l'instruction "**Exit Do**".

LES TESTS

Les tests permettent de lancer l'exécution d'une ou plusieurs instructions si une ou plusieurs conditions sont remplies.

En Visual Basic il existe : le test "If ... End if"(Il s'agit du test "Si"en algorithmie)
le test "Select ... Case"(Il s'agit du test "Selon"en algorithmie)

Dans ce chapitre, on fait appel à la notion de "condition". Elle utilise des opérateurs et peut nécessiter l'utilisation de parenthèses. Exemple de condition : (var > 1) signifie que la valeur de la variable "var" doit être supérieure à 1.

Le test "If"(test si) : structure itérative

Test simple: **If** "condition" **then** "instruction" **else** "instruction"

Remarques sur ce test : Lorsque le test ne comporte qu'une seule instruction dans le "**then**" et dans le "**else**", on peut écrire le test sur une seule ligne. Le "**else**" est facultatif.

Test complexe :

```
    If "condition"then  
"instruction 1»  
"instruction 2»  
.....  
"instruction X»  
else  
"instruction 1»  
.....  
"instruction X»  
    End if
```

Remarques sur ce test : Dans ce type de test la balise "End if" est obligatoire. Le "**else**" est là aussi facultatif.



Le test "Select Case" (selon le cas) :

Remarques sur ce test : Ce test compare la valeur d'une variable à différentes valeurs afin de déterminer quelle instruction exécuter. Si aucun cas ne correspond, c'est le cas "Else" qui est employé. Ce cas "Else" est facultatif.

Select Case "variable"

Case valeur 1

Instruction 1

Case valeur 2

Instruction 2

Case valeur 3

Instruction 3

.....

Case valeur n

Instruction n

Case Else

Instruction

End Select



LES PROCEDURES

Une procédure est un sous-programme composé d'instructions. Ce sous programme possède un nom unique et peut être appelé n'importe où dans le programme principal. La procédure permet d'éviter la redondance d'instructions dans le programme principal. La procédure ne renvoie pas de résultat (contrairement à la fonction) mais elle peut accepter des paramètres, on parle alors de procédure paramétrée. Ces paramètres sont des variables qui seront déclarées en même temps que la procédure. Visual Basic est un langage de programmation événementiel qui utilise des procédures événementielles. Ainsi, les instructions placées derrière chaque événement sont contenues dans des procédures événementielles. L'utilisateur peut donc définir ses propres procédures adaptées aux besoins de son programme.

Création d'une procédure non paramétrée :

```
Private/Public Sub "Nom de la procédure"()  
    Instruction 1  
    Instruction 2  
    .....  
    Instruction X  
End Sub
```

Remarques : Avant le mot clé **sub**, on spécifie la portée de la procédure :

Private si elle n'est utilisable que dans la feuille où elle est déclarée.

Public si elle est utilisable dans toutes les feuilles du projet.

Toutes les boucles et tests vus précédemment sont utilisables dans les procédures. Même s'il n'y a pas de paramètres, les () sont obligatoires.

Création d'une procédure paramétrée :

```
Private/Public Sub "Nom de la procédure" ( "Nom paramètre 1" as Type paramètre 1,  
"Nom paramètre 2" as Type paramètre 2, "Nom paramètre X" as Type paramètre X)  
    Instruction 1  
    Instruction 2  
    .....  
    Instruction X  
End Sub
```

Remarques : Le nombre de paramètres utilisables dans une procédure est illimité. Les paramètres sont des variables dont le type est connu. La création d'une procédure paramétrée est nécessaire que si les paramètres sont utilisés dans la procédure.

Appel d'une procédure :

Procédure non paramétrée : **Call** "Nom Procédure"()

Remarque : le **Call** et les parenthèses ne sont pas obligatoires, le nom de la procédure suffit pour l'appel.



Procédure paramétrée : **Call** "Nom Procédure"(valeur paramètre1, valeur paramètre 2, valeur paramètre X)

Remarque : le Call n'est pas obligatoire. Il est obligatoire de renseigner la valeur des paramètres dans les parenthèses lors de l'appel. Il est important que le type des paramètres transmis lors de l'appel correspondent aux types des paramètres déclarés. Ainsi, transmettre un paramètre de type "chaîne de caractères" alors qu'il était déclaré comme "entier" retournera une erreur d'exécution.

LES FONCTIONS

Une fonction est aussi un sous-programme composé d'instructions. Ce sous-programme possède un nom unique et peut être appelé n'importe où dans le programme principal. Elle permet également d'éviter la redondance d'instructions dans le programme mais sa fonction première est d'opérer un traitement dont le résultat est retourné dans le programme principal (ce résultat est stocké dans le nom de la fonction). Elle peut accepter des paramètres, on parle alors de fonction paramétrée. Nous verrons plus loin qu'il existe des fonctions intégrées au langage c'est à dire qu'elles sont inscrites dans le compilateur et que l'utilisateur peut s'en servir sans avoir à les écrire. Exemple : La fonction "Date" retourne la date du jour. Vous l'aurez compris, la différence entre une fonction et une procédure est qu'une fonction retourne un résultat et non la procédure.

Création d'une fonction non paramétrée :

```
Private/Public Function "Nom de la fonction"() as "Type de la fonction »  
    Instruction 1  
    Instruction 2  
    .....  
    Instruction X  
    "Nom de la fonction"= "résultat »  
End Function
```

Remarques : Avant le mot clé **function**, on spécifie la portée de la fonction :

Private si elle n'est utilisable que dans la feuille où elle est déclarée.

Public si elle est utilisable dans toutes les feuilles du projet.

Toutes les boucles et tests vus précédemment sont utilisables dans les fonctions. Même s'il n'y a pas de paramètre, les () sont obligatoires. L'affectation du résultat dans le nom de la fonction est obtenu par l'instruction suivante :

```
"Nom de la fonction"= "résultat"
```

Cette instruction doit être placée dans la fonction même pour que le résultat soit renvoyé au programme principal. Le résultat retourné doit obligatoirement être du type de la fonction sinon une erreur d'exécution sera retournée.

Création d'une fonction paramétrée :

```
Private/Public Function "Nom de la fonction"("Nom paramètre 1"as "Type paramètre  
1, "Nom paramètre 2"as "Type paramètre 2, "Nom paramètre X"as "Type paramètre  
X) as "Type de la fonction"  
    Instruction 1  
    Instruction 2  
    .....  
    Instruction X  
    "Nom de la fonction"= "résultat"  
End Function
```



Remarques : Le nombre de paramètres utilisables dans une fonction est illimité. Les paramètres sont des variables dont le type est connu. La création d'une fonction paramétrée est nécessaire que si les paramètres sont utilisés dans cette fonction.

Appel d'une fonction :

Fonction non paramétrée : "Nom Fonction"

Le nom de la fonction suffit pour l'appel, toutefois, on peut placer un call devant le nom de la fonction si on n'utilise pas directement la fonction dans une instruction.

Fonction paramétrée : "Nom Fonction" (valeur paramètre1, valeur paramètre 2, valeur paramètre X)

Il est obligatoire de renseigner la valeur des paramètres dans les parenthèses lors de l'appel. Il est important que le type des paramètres transmis lors de l'appel correspondent aux types des paramètres déclarés. Ainsi, transmettre un paramètre de type "chaîne de caractères" alors qu'il était déclaré comme "entier" retournera une erreur d'exécution.