Programmation structurée en Visual Basic Premiers pas

Licence — Université Lille 1
Pour toutes remarques : Alexandre.Sedoglavic@univ-lille1.fr

Première année DEUST — 2008-09

Ce cours est porte sur la programmation structurée en s'appuyant sur le langage VisualBasic.

VB 1.0 a été introduit en 1991 par MicroSoft en se basant sur le langage Basic (Dartmouth College, New Hampshire, USA, 1964).

Il s'agit d'un langage de *programmation événementielle* dans lequel les programmes sont définis par leurs réactions à différents événements.

Ainsi, il permet la création aisé d'interfaces utilisateur graphique (GUI), l'accès aux bases de données, etc.

La dernière mise à jour est la version VB 6.0 sortie en 1998. À partir de la version 7.0, le langage évolue et est qualifié de Visual Basic.net ; ce cours ne présente pas les spécificités de ce langage.

Types : représentation mémoire

Les types suivants décrivent comment sont stockées — taille et codage — en mémoire les informations correspondantes.

Type	taille en octet	valeurs	
Boolean	2	Vrai ou Faux	
Byte	1	$0 \ldots 255$	
SByte	1	$-128 \ldots 127$	
Char	2	$0 \ldots 65535$	C
Date	8	$01/01/100 \rightarrow 31/12/9999$	
Decimal	16	entier ou décimal	D
Double	8		R
Integer	4	$-2^{16} \ldots 2^{16} - 1$	I

Types : représentation mémoire (suite)

Type	taille en octet	valeurs
UInteger	4	$0 \dots 2^{31} - 1$
Long	8	$-2^{32} \dots 2^{32} - 1$
ULong	8	$0 \dots 2^{64} - 1$
Short	2	$-2^{8} \dots 2^{8} - 1$
UShort	2	$0 \dots 2^{16} - 1$
Single	4	
String	variable	
Structure	variable	suivant les membres
Object	4	pointe sur tous types de données

Typages de constantes

Les constantes peuvent être suivies de lettres — indiquées en dernière colonne des tableaux précédents — permettant de spécifier le type. Par exemple,

- `"a"c` est un charactère (et non pas une chaîne de caractères i.e. une suite de caractères se terminant par 0) ;
- `256S` correspond en fait à 0, etc.

Il est possible d'utiliser les numérotations octales et hexadécimal pour manipuler les constantes.
Par exemple,

- `&H10` représente la constante 16 ;
- `&O10` représente la constante 8.

Identificateurs

Les identificateurs servent à manipuler les objets du langage i.e. manipuler de l'espace mémoire auquel on donne un nom. Cet espace mémoire peut contenir des données ou des instructions manipulant des données (identificateurs de variables/de fonctions).

Un identificateur se doit d'informer sur ce qu'est et à quoi sert l'objet qu'il identifie.

En VB, un identificateur est une suite d'au plus 40 caractères (hors séparateurs, opérateurs, mots clefs) commençant par une lettre.

Il convient de n'utiliser ni de caractère accentué ni de caractère spéciaux (espace, #, ?, etc).

Si l'identificateur est constitué de plusieurs mots, ces derniers commencent par une majuscule.

Mots réservés et commentaires

Les mots réservés de VB sont :

And	As	ByRef	ByVal	Call	Case
Class	Const	Dim	Do	Else	ElseIf
End	Exit	False	For	Function	GoTo
If	Is	Loop	Me	Module	Next
Not	Nothing	Option	Or	Private	Public
Resume	Select	Step	Sub	Then	To
True	Until	While			

Ils ne peuvent pas servir d'identificateur.

Il est possible de commenter son code de la manière suivantes :

```
' l'apostrophe (code ASCII 39) d\'ebute un commentaire
REM on peut la remplacer par le mot clef REM
REM 2*35 ce mot doit d\'ebuter la ligne
2*35 ' alors que l'apostrophe peut se mettre apr\'es du code
```

Déclaration de variables

Schématiquement, une variable correspond à un emplacement en mémoire. Dans le code source, ce dernier est manipulé par l'entremise de l'identificateur de la variable.

En VB, on déclare les variables comme suit :

`Dim` IdentificateurDeLaVariable `As` TypeDeLaVariable

Par exemple, on peut avoir :

```
Dim VariableEntier As Integer
Dim UnReel As Single
REM d\'eclaration multiple
Dim AutreEntier As Integer, AutreReel As Single
Dim NomPrenom As String
```

Une constante est une forme de variable dont l'affectation ne peut se faire qu'à la déclaration et qui ne varie pas. En VB, on déclare (et définit) les constantes comme suit :

`Const` IdentificateurDeVariable `As` TypeDeVariable = Valeur

Par exemple, on peut avoir : `Const Pi As Single = 3.14`

Conversion explicite

La table suivante présente les fonctions de conversion en VB :

fonction	type obtenu	fonction	type obtenu
CBool	Boolean	CObj	Object
CByte	Byte	CSByte	SByte
CChar	Char	CShort	Short
CDate	Date	CSng	Single
CDbl	Double	CStr	String
CDec	Decimal	CUInt	UInteger
CInt	Integer	CULng	ULong
CLng	Long	CUShort	UShort

Plus génériquement, la fonction CType prend comme paramètres une évaluation de variable et un type ; elle retourne le résultat de l'évaluation codée dans le type spécifié.

Pour finir, constatons que les constantes sont considérées comme étant de type `Integer`. Ainsi, le programme

```
Module essai
  Public Sub Main()
      MsgBox (2*16384)
  End Sub
End Module
```

va provoquer une erreur de débordement alors que le code

```
Module essai
  Public Sub Main()
      MsgBox (2*16384&)
  End Sub
End Module
```

ne le fera pas.

Opérateurs arithmétiques

Ces opérateurs sont infixes et agissent sur 2 arguments :

`a opérateur b`

(il existe un + et un − unaire).

opérateur	commentaires	Priorité
^	retourne a à la puissance b	5
*	multiplication adapte le type de retour au besoin	4
/	division réelle conversion automatique des opérandes en rééls	4
\	quotient de la division euclidienne conversion automatique des opérandes en entiers	3
Mod	reste de la division euclidienne même résultat que $a - (a \setminus b) * b$	2
+	additions de 2 nombres ne devrait pas être utilisé pour concaténer des chaînes	1
−	soustraction de a par b	1
<<	décalage des lettres binaires vers la gauche	
>>	décalage des lettres binaires vers la droite	

Il convient de ne pas hésiter à parenthéser les expressions.

Opérateur d'affectation des variables

L'opérateur = permet de donner une valeur aux variables. Ainsi, en utilisant les déclarations précédentes :

```
VariableEntier = 7
UnReel = 4.33
AutreEntier = 5
AutreReel = 0.33
NomPrenom = "Sedoglavic Alexandre"
```

Par ailleurs, on peut coupler certaines opérations et l'affectation avec les opérateurs suivant :

$$\hat{}=,\ /=,\ \backslash=,\ *=,\ -=,\ +=,\ \&=,\ >>=,\ <<=.$$

Ainsi, quand c'est possible l'affectation $A\diamond = B$ se comporte comme $A = A \diamond B$ (ou $\diamond =$ est un des opérateurs ci-dessus).

Opérateurs booléens et de comparaisons

L'action des opérateurs booléens est résumée dans le tableau suivant :

		conjonction	disjonction	ou exclusif	Equivalence	Implication	négation
A	B	A And B	A Or B	A Xor B	A Eqv B	A Imp B	Not A
T	T	T	T	F	T	T	F
T	F	F	T	T	F	F	F
F	T	F	T	T	F	T	T
F	F	F	F	F	T	T	T

Les opérateurs suivants comparent 2 quantités et retourne un booléen :

=	Égalité
<>	Inégalité
<	plus petit
>	plus grand
>=	plus grand ou égal
<=	plus petit ou égal

Autres opérateurs

On dispose des opérateurs suivants :

- & opérateur de concaténation de chaînes de caractères ;
- `Is` égalité de 2 objets ;
- `IsNot` inégalité de 2 objets ;
- `TypeOf() Is` permet de déterminer le type d'un object ;
- A `Like` regexp retourne vrai si la chaîne de caractères *A* vérifie l'expression régulière regexp (voir la page les définissant ci-dessous).

Les *expressions régulières* décrivent des propriétés de construction de chaînes de caractères. Pour ce faire, on utilise en shell les *métacaractères* :

- le point d'interrogation ? correspond à n'importe quel caractère (sauf EOL). L'expression régulière `b?l` représente les chaînes *bal* et *bol* et toutes les autres combinaisons comme *bwl* ;
- le dièse # correspond à n'importe quel chiffre ;
- la paire de crochet `[ ]` permet de spécifier plus restrictivement un ensemble de caractères. L'expression régulière `dupon[dt]` ne représente que les chaînes *dupond* et *dupont*. L'expression `dupon[d-t]` représente les chaînes commençant par *dupon* et se terminant par une lettre comprise entre *d* et *t*. L'expression `dupon[!dt]` représente les chaînes commençant par *dupon* et ne se terminant ni par *d* ni par *t* ;
- $*$ désigne $0, 1$ ou plusieurs caractères quelconques.

Le préfixe \ (antislash) transforme un métacaractère en caractère.

Environnement de développement intégré (ide)

Microsoft Visual Studio est un ensemble de logiciels de développement pour Windows permettant d'engendrer des application Web ASP.NET, des services Web XML, des applications bureautiques, etc.

Les langages VB, Visual C++, Visual C#, etc. utilisent tous cet environnement intégré fournissant un éditeur de texte, un dévermineur, etc.

Cet IDE est disponible à l'url :

`www.microsoft.com/visualstudio/`

Premier programme

```
Module essai
  Public Sub Main()
      MsgBox ("Bonjour le monde")
  End Sub
End Module
```

Voyons comment créer ce programme (nous reviendrons sur la sémantique plus tard).

- lancer l'environnement de développement VB
- Fichier → Nouveau projet → Application console
- taper votre programme
- Générer (phase de compilation)
- Déboguer → démarrer (F5)

Instructions conditionnelles simples

L'exécution du flux d'instructions peut être conditionnée comme suit :

```
If Expression
Then
     instructions
End If
```

Dans cette instruction, `Expression` est une expression qui s'évalue en un booléen ; si le résultat est vrai alors l'instruction `instructions` est exécutée.

Il est possible d'indiquer une alternative comme suit :

```
If Expression
Then
     instructions1
Else
     instructions2
End If
```

Instructions conditionnelles multiples

La première forme de conditionnelle multiple se base sur le `If` :

```
If Expression1
Then
     instructions1
ElseIf Expression2
Then
     instructions2
   ....
Else
     instructionN
End If
```

Le premier jeu d'instructions correspondant à une `Expression` s'évaluant à vrai sera le seul exécuté dans le code ci-dessus.

Comparaisons multiples

L'évaluation d'une expression est comparée à plusieurs suites d'expressions de comparaisons dépendantes d'un seul argument :

```
Select Case TestExpression
    case SuiteExpressionComparative1
            instruction(s)1
    case SuiteExpressionComparative2
            instruction(s)2
        ....
    case SuiteExpressionComparativeN
            instruction(s)n
    case Else instruction ' si rien ne correspond avant
End Select
```

Chaque expression de comparaison est :

- soit une expression ;
- soit du type : expression1 To expression2 ;
- soit du type : <optionnel> Is </optionnel> OperateurDeComparaison expression.

Si TestExpression vérifie une suite d'expressions de comparaison `SuiteExpressionComparative`, l'instruction la suivant est exécutée et on passe au `case` suivant. Si cette elle vérifie plusieurs suites, seule la première instruction associée est exécutée.

Attention, le `Is` utilisé dans cette instruction n'est pas l'opérateur du même nom.

Par exemple, on peut avoir :

```
Dim Salaire,RemboursementMensuel As Decimal
Select Case (RemboursementMensuel/Salaire)*100
    case Is >= 33# MsgBox("ca va pas \^etre possible")
    case 30# To 32# MsgBox("ca va \^etre juste")
    case 20# To 29# MsgBox("c'est OK")
    case Else MsgBox("en chaussette")
End Select
```

Boucle itérative

Cette instruction nécessite un compteur, un intervalle (`s`, `e`) et un `pas` afin d'obtenir :

```
For compteur <opt>As type</opt> = s To e <opt>Step pas</opt>
     instructions
Next <opt>compteur</opt>
```

où

- `type` doit être spécifié si `compteur` n'est pas déclaré précédemment ;
- `compteur` prend toutes les valeurs entre `s` et `e` en étant incrémenté de `pas` à chaque itération.

Nous verrons plus tard d'autres formes de cette instruction (`For Each` énumération des membres d'un tableau, etc).

Itérations à condition

Dans l'itération suivante :

```
While condition
     instruction(s)
End While
```

`instruction(s)` est exécuté tant que `condition` est vrai. Cette expression est testée en début de boucle.

Plus généralement, on peut avoir les boucles

```
Do { While | Until } condition
   instruction(s)
Loop
--- ou ---
Do
   instruction(s)
Loop { While | Until } condition
```

où `While` (resp. `Until`) permet la répétition de la boucle tant que `condition` est vrai (resp. faux).