Introduction à la programmation VBA sous Microsoft Excel 97

Mathieu Peltier

E-Mail : Mathieu.Peltier@netcourrier.com
Homepage : http://altern.org/peltierm/welcome.htm

Mise à jour du 6/2/2000 – Typographie LaTeX 2_ε

Ce guide est une introduction succincte à la programmation VBA sous Microsoft Excel 97. Loin d'être exhaustif, il a pour ambition de fournir au débutant une première approche de VBA.

Table des matières

1 Présentation générale de VBA **3**
- 1.1 Pour commencer . . . 3
- 1.2 Qu'est-ce que VBA ? . . . 3
- 1.3 Notion de programmation objet . . . 3
- 1.4 Avantages et inconvénients de VBA . . . 4

2 L'environnement de programmation **5**
- 2.1 L'éditeur intégré VBE . . . 5
- 2.2 Notion de projet . . . 5
- 2.3 Les éléments d'un projet . . . 5
- 2.4 Les modules . . . 5
- 2.5 Les boîtes de dialogue . . . 6
- 2.6 Les modules de classe . . . 6
- 2.7 La fenêtre propriétés . . . 7
- 2.8 Obtenir de l'aide . . . 7

3 Variables et constantes **7**
- 3.1 Les types de données . . . 7
- 3.2 Nom des variables . . . 8

3.3 Déclaration des variables 9
3.4 La portée des variables 9
3.5 Les constantes 10

4 Principaux mots-clé du langage VBA 10
4.1 Les opérateurs 10
4.1.1 Opérateurs arithmétiques. 10
4.1.2 Opérateurs de comparaison. 11
4.1.3 Opérateurs logiques. 11
4.2 Structures conditionnelles 11
4.2.1 If Then Else ElseIf 11
4.2.2 Select Case 12
4.3 Boucles 12
4.3.1 Do While Loop 12
4.3.2 Do Until Loop 12
4.3.3 For Step Next 13
4.3.4 For Each In Next 13
4.4 Gestion des programmes 13
4.4.1 Exit 13
4.4.2 End et Stop 14
4.4.3 On Error GoTo 14
4.5 Conversion de types 15

5 Procédures et fonctions 15
5.1 Les procédures 15
5.2 Les fonctions 16
5.3 Passage d'arguments par valeur ou par référence 16
5.4 Les procédures évènements 17

6 Conception de programme 17
6.1 Quelques conseils 17
6.2 Utiliser des références 18
6.3 Débogage de programme 18
6.4 Les macros complémentaires 19

7 Un exemple pratique 19
7.1 Conception de la boîte de dialogue 20
7.2 Programmation des contrôles de la boîte 21
7.3 Améliorations possibles 23
7.4 Création d'une commande spécifique 24

8 Pour aller plus loin 24
8.1 Utiliser l'aide en ligne ! 24
8.2 Quelques références 24

1 Présentation générale de VBA

1.1 Pour commencer

Il est possible de programmer sous Excel en n'écrivant *aucune* ligne de code ! Excel met pour cela à disposition l'enregistrement automatique de macro. La procédure à suivre est la suivante :

- activer la commande **Outils/Macro/Nouvelle macro** ;
- entrer dans la boîte de dialogue le nom de la macro qu'on souhaite enregistrer (le bouton **Arrêter l'enregistrement** devrait apparaître) ;
- réaliser manuellement l'ensemble des commandes, qu'on veut voir enregistrer dans la macro [1] ;
- cliquer sur le bouton **Arrêter l'enregistrement** pour stopper la procédure.

Une fois l'opération réalisée, on dispose d'une nouvelle macro réalisant les actions enregistrées. Les macros créées peuvent être exécutées grâce à la commande **Outils/Macro/Macros...** (ALT+F8).

Ce procédé peut être très utile pour réaliser des macros simples. Il permet également de se familiariser avec la programmation VBA, en étudiant le code VBA obtenu [2]. Cependant, le code obtenu est loin d'être optimisé. De plus, pour des commandes complexes, l'enregistrement échoue dans certains cas.

1.2 Qu'est-ce que VBA ?

VBA est l'acronyme de *Visual Basic for Applications*. Il s'agit du langage de programmation de l'ensemble des applications Microsoft Office (Excel, Word, Access, etc.), introduit à partir de la version 97.

VBA permet d'automatiser certaines tâches rébarbatives ou répétitives, mais encore de réaliser des applications complètes. Il s'agit d'un langage de programmation orienté objet comparable à Visual Basic [3]. Les procédures développées sont également appelées des *macros*.

1.3 Notion de programmation objet

VBA est un langage de programmation orienté objet : on accède ainsi aux différents éléments de l'application (classeur, feuilles de calcul, graphiques, etc.) en utilisant ce qu'on appelle des objets, qui représentent précisément

1. Il est recommandé de ne pas faire d'erreur pendant l'enregistrement : les corrections éventuelles sont en effet également prises en compte.

2. cf. section 2 (page 5) pour savoir comment visualiser le code obtenu.

3. La différence essentielle est que Visual Basic permet de réaliser des programmes exécutables, alors que VBA nécessitera toujours la présence du programme Office, sous lequel l'application a été développée.

ces éléments. Par exemple, pour renommer le classeur *Classeur1*, il suffira d'entrer :

```
WorkBooks("Classeur1").name = nouveau_nom
```

Pour chaque objet est défini un ensemble de ce que l'on appelle des *propriétés* et des *méthodes*, auxquelles on accéde au moyen des instructions :

```
<Objet>.<Propriété> et <Objet>.<Méthode>
```

Les propriétés sont des caractéristiques modifiables ou non de l'objet. Par exemple, *Name* est ici une propriété de l'objet *WorkBooks("Classeur")*. Une propriété peut également être un objet. Par exemple, pour changer le nom de la feuille de calcul *Feuil1* du classeur *Classeur1*, on écrira :

```
Workbooks("Classeur1").Worksheets("Feuil1").Name = nouveau_nom
```

Les méthodes sont les actions rattachées à l'objet en question. Par exemple, la méthode *close* de l'objet *WorkBooks("Classeur1")* sert à fermer le classeur :

```
WorkBooks("Classeur1").close
```

L'ensemble des objets contenus dans un objet particulier forme ce que l'on appelle une *collection*. Consulter l'explorateur d'objets pour connaître la hiérarchie des objets[4].

1.4 Avantages et inconvénients de VBA

Le principal avantage de VBA est sa simplicité. Il permet au non-spécialiste, après un rapide apprentissage, de réaliser des applications complexes, en obtenant en particulier une interface de qualité. De plus, VBA permet de programmer sur toutes les applications d'Office.

Cependant, il faut garder à l'esprit que les programmes développés ne pourront être utilisés sans l'application sous laquelle ils ont été développé. De plus, VBA est relativement lent; les programmes nécessitent donc une machine assez performante pour fonctionner confortablement. Il faut aussi reconnaître que toutes les possibilités du langage[5] ne sont pas documentées.

Enfin, dernière objection et non des moindres, Excel se révèle dans certains cas instable : le programmeur se retrouve parfois dans des situations inextricables, à cause des dysfonctionnements d'Excel. Dans ce cas, une certaine dose de calme est nécessaire !

4. cf. section 2.8 (page 7).
5. en particulier pour des utilisations avancées.

2 L'environnement de programmation

2.1 L'éditeur intégré VBE

On dispose à l'intérieur d'Excel d'un environnement de programmation VBA puissant, nommé *Visual Basic Editor* ou *VBE*. On accède à cet environnement grâce à la commande **Outils/Macro/Visual Basic Editor** (Alt+F11). La barre d'outils attachée à l'éditeur diffère de la barre d'outils d'une feuille de calcul. Pour revenir à l'écran standard, taper à nouveau Alt+F11.

2.2 Notion de projet

Une application sous Excel est appelée *projet*. Un projet est rattaché en fait à chaque classeur, et contient toutes les feuilles du classeur, mais aussi les éventuels modules, les boîtes de dialogue, etc. comme on le verra par la suite. Tout programme VBA un tant soit peu complexe contient en effet plusieurs de ces éléments. La commande **Affichage/Explorateur de projet** fait apparaître une fenêtre permettant d'explorer les projets de tous les classeurs ouverts. Par défaut, le projet associé à un classeur quelconque s'appelle *VBAProject*.

2.3 Les éléments d'un projet

Un projet regroupe, comme on l'a vu, un ensemble d'éléments comprenant des objets Microsoft Excel, qui sont les feuilles de calcul du classeur considéré, des boîtes de dialogue, des modules et des modules de classe[6].

Il est possible d'insérer ces derniers éléments grâce aux commandes **Insertion/Module**, **Insertion/Userform** et **Insertion/Module de classe**.

Noter enfin qu'il est possible d'importer et d'exporter chacun de ces éléments sous la forme de fichiers indépendants. Les commandes à utiliser sont **Fichiers/Exporter un fichier...** et **Fichiers/Importer un fichier...**.

2.4 Les modules

Les *modules* contiennent le code d'une application. Il est possible[7] de structurer les programmes en plusieurs modules. Chaque module contiendra alors le code rattaché à une action particulière.

Excel offre des caractéristiques très intéressantes, qui facilite l'édition du code. Ainsi, le code est affiché avec différentes couleurs permettant de différencier les mots clés[8], les commentaires, etc. De plus, la syntaxe est

6. Comme on le verra par la suite, un module de classe est un module particulier.

7. et vivement recommandé pour des raisons de lisibilité dès que le programme devient un tant soit peu complexe.

8. Ceux-ci sont également formatés automatiquement.

automatiquement vérifiée en arrière plan et les procédures et fonctions (cf. section 5, page 15) sont clairement séparées.

Mais le plus intéressant est certainement la complétion automatique des instructions : par exemple, si l'utilisateur rentre le mot *ActiveWorkBook*[9] suivi d'un point, l'éditeur présente dans une liste déroulante toutes les possibilités disponibles pour compléter la saisie. L'utilisateur n'a plus qu'à sélectionner la ligne désirée en tapant le nombre suffisant de caractères, et à appuyer sur TAB pour valider la saisie. Ce procédé accélère considérablement la saisie, évite bien des erreur de syntaxe et permet également de connaître les propriétés et méthodes associées à un objet donné.

2.5 Les boîtes de dialogue

Les *boîtes de dialogue* ou *userforms* sont des feuilles particulières, destinées à gérer les interactions entre l'utilisateur et le programme.

Lors de la conception de la boîte de dialogue, le programmeur dispose d'un certain nombre d'objets, qu'il peut insérer dans la boîte de dialogue. Ces objets font partie de la boîte à outils que l'on active par la commande **Affichage/Boîte à outils**. Il suffit alors de sélectionner l'élément désiré[10] et de les créer dans la boîte de dialogue.

Il appartient ensuite au programmeur d'écrire les procédures permettant de gérer la boîte de dialogue obtenue, par exemple pour récupérer la valeur d'une saisie réalisée dans une zone de saisie. Le code ad hoc est à placer dans le module de classe[11] associé à la boîte de dialogue, que l'on active par exemple par le bouton ad hoc dans l'explorateur de projet, si la boîte en question est sélectionnée.

2.6 Les modules de classe

Les *modules de classe* sont des modules particuliers où est placé le code VBA rattaché à un objet particulier.

Un exemple typique est le module de classe *ThisWorkbook* : ce module contient toutes les procédures gérant le classeur en cours. Il est par exemple possible d'écrire une procédure événement[12] associée à l'ouverture du classeur. Une telle procédure, dont le nom est *Workbook_open*, devra être écrite dans le module de classe *ThisWorkBook*.

Un autre exemple de module de classe est le module où est stocké le code associé à chaque boîte de dialogue. C'est également dans ce module que seront programmées les procédures qui gèrent la boîte de dialogue[13].

9. Cet objet représente le classeur actif, comme son nom l'indique.
10. Ces éléments sont appelés des *contrôles*.
11. Voir section suivante.
12. cf. section 5.4 (page 17).
13. Ne serait-ce que par exemple les procédures régissant le comportement des boutons de commandes de la boîte de dialogue.

2.7 La fenêtre propriétés

A chaque objet est associé toute une série de propriétés, contenues dans la *fenêtre propriétés*. Pour visualiser ces propriétés, utiliser la commande **Affichage/Fenêtre Propriétés**.

Il est par exemple possible de transformer à sa guise la couleur de fond d'une boîte de dialogue, en modifiant sa propriété *BackColor* [14] ou encore de modifier le type de police grâce à la propriété *Font*.

Il est également souhaitable de renommer les éléments contenus dans le projet, afin de donner des noms plus explicites que les noms par défaut affectés par Excel. Dans le même ordre d'idée, on peut également renommer le projet.

Afin d'être plus efficace, il est conseillé de travailler sous l'éditeur avec la fenêtre propriétés et l'explorateur de projets en permanence ouverts, le reste de l'écran servant à visualiser l'élément sélectionné.

2.8 Obtenir de l'aide

Il y a de nombreuses possibilités pour obtenir de l'aide sous l'environnement VBE. Le compagnon Office peut tout d'abord être interrogé grâce au bouton **Assistant Office** de la barre d'outils standard. Si la réponse n'est pas satisfaisante [15], procéder à la recherche grâce à la commande **?/Sommaire et Index**. En outre, à tout moment la touche F1 permet d'obtenir une aide contextuelle.

Une autre possibilité intéressante pour se retrouver dans la masse des propriétés et méthodes mises à disposition pour chaque objet est d'utiliser l'explorateur d'objets grâce à la commande **Affichage/Explorateur d'objets** (F2). Cet outil permet de chercher une propriété ou une méthode donnée grâce à un index.

Enfin, comme signalé précédemment, l'enregistreur de macro peut être également un bon moyen de déterminer les commandes à utiliser pour une action particulière. Il suffira alors de réaliser l'action manuellement et de voir ensuite comment Excel l'a traduite en VBA.

3 Variables et constantes

3.1 Les types de données

VBA autorise la création de la plupart des types de données classiques. Les principaux types de données disponibles sont les suivants :

– *Integer* : entier court ;

14. Pour cela, cliquer sur la ligne BackColor et utiliser le sélecteur qui apparaît.
15. comme souvent...

- *Long* : entier long ;
- *Single* : réel simple ;
- *Double* : réel en double précision ;
- *Currency* : nombre monétaire ;
- *Date* : date et heure [16] ;
- *String* : chaîne de caractères ;
- *Boolean* : booléen (*True* ou *False*) ;
- *Object* : référence quelconque à un objet ;
- *Variant* : type particulier pouvant être n'importe quel autre type.

Outre ces types élémentaires, il est également possible de créer des tableaux et des types personnalisés. Les deux exemples de déclarations suivantes illustrent le fonctionnement de ces deux types de données :

```
Dim tab[1 to 20] As Integer
Type personne
  nom As String * 20
  adresse As String * 50
  anniversaire As Date
End Type
```

L'instruction *Dim* sert à déclarer la variable tab comme un tableau, comme on le verra à la section suivante.

L'instruction *Type* permet de définir le type personnalisé *personne*. On accède ensuite à chaque champs d'un type personnalisé en spécifiant la désignation du champs à la suite du nom de la variable. On écrira par exemple :

```
Dim zaza As personne
zaza.nom = ''x''
zaza.anniversaire = ''12/02/1979''
```

3.2 Nom des variables

Les noms des variables doivent commencer par un caractère alphabétique et ne pas comporter les caractères suivants :

```
. % & ! # @ $
```

De plus, ils ne peuvent pas excéder 255 caractères. Pour faciliter la lisibilité des programmes, on s'efforcera d'utiliser des minuscules pour les variables utilisées par le programme.

16. Le problème du passage à l'an 2000 est pris en compte !

3.3 Déclaration des variables

Par défaut, il n'est pas nécessaire de déclarer les variables utilisées. On peut donc utiliser n'importe quelle variable sans se préoccuper de sa déclaration préalable. Cependant, ce type de fonctionnement n'est pas satisfaisant.

En effet, d'une part, les variables non déclarées ont en fait toutes le type *variant*, qui peut contenir n'importe quel autre type. Il va de soit que la taille des variables créées est alors loin d'être optimisée, ce qui ralentit l'exécution et consomme de la mémoire inutilement.

D'autre part, la déclaration explicite des variables permet d'éviter certaines erreurs difficiles à détecter, dès que le programme devient un tant soit peu complexe. Il vaut donc mieux opter pour la déclaration explicite des variables : on écrira à cet effet la ligne *Option Explicit* au début de chaque module de code [17].

L'instruction utilisée pour déclarer une variable est *Dim* comme on l'a déjà vu [18] :

```
Dim ma_variable As Integer
```

3.4 La portée des variables

Toutes les variables n'ont pas la même portée. On distingue les variables *locales* et les variables *globales* ou *publiques*.

Les variables locales sont déclarées par l'instruction *Dim* au niveau d'une procédure ou d'une fonction [19]. Elles ne sont accessibles qu'à l'intérieur d'une procédure ou de la fonction à l'intérieur de laquelle elles apparaissent. Leur durée de vie est d'ailleurs limitée à l'existence de la procédure ou de la fonction dans laquelle elles sont définies.

La portée des variables globales s'étend au niveau du module dans lesquelles elles sont déclarées. Les déclarations s'effectuent au début du module, et les variables concernées seront disponibles pour toutes les procédures et fonctions du module.

Il est également possible de définir des variables globales, utilisables dans tous les autres modules, grâce à l'instruction *Public* :

```
Public var_publique As Integer
```

Les variables globales ont une durée de vie beaucoup plus importante que les variables locales : leur contenu est en effet conservé jusqu'à l'initialisation du programme [20].

17. Cocher la case *Déclaration explicite des variables* dans la boîte de dialogue obtenue par la commande **Outils/Options...** sous VBE (onglet *Editeur*).

18. En fait, il existe d'autres possibilités (cf. section 3.4, page 9).

19. cf. section 5, page 15.

20. Cette initialisation n'est pas automatique. Elle a lieu à la première exécution du programme et peut être obtenu grâce à l'instruction *end*.

Des conflits entre les différentes variables peuvent apparaître dans certains programmes. Dans ce cas, VBA applique une règle simple : les variables prises en compte sont celles qui ont le domaine de validité le plus restreint.

3.5 Les constantes

Il se trouve souvent dans un programme des valeurs qui doivent être utilisées, mais qui ne sont pas susceptibles de varier. Dans le but d'améliorer la lisibilité des programmes mais aussi de faciliter la modification ultérieure du programme, on aura tout intérêt à définir des constantes symboliques, dans un module particulier.

L'instruction *Const* sert à déclarer les constantes, qui peuvent être typées ou non :

```
Dim taux_interet = 0,03
Dim nb_max As Integer = 100
```

Pour le nom des constantes, les mêmes règles que dans le cas des variables s'appliquent.

Il est à noter qu'au sein de VBA sont définies un grand nombre de constantes auxquelles on peut accéder, dans n'importe quel programme. Ces constantes ont été implémentées pour faciliter la tache des programmeurs, qui peuvent par exemple utiliser des constantes prédéfinies au nom explicite pour exploiter certaines valeurs retournées par des fonctions préprogrammées. Consulter l'aide en ligne pour obtenir plus d'informations.

4 Principaux mots-clé du langage VBA

VBA comprend un grand nombre d'instructions. Cette section présente quelques mots clés importants et couramment utilisés. La plupart des mots clés présentés ici sont en fait les mêmes que ceux utilisés dans la plupart des langages. Attention, de nombreuses instructions utiles ne sont pas citées dans cette section[21]. Pour plus d'informations, consulter l'aide en ligne à la rubrique Mots clés.

4.1 Les opérateurs

4.1.1 Opérateurs arithmétiques.

Les opérateurs suivants permettent de réaliser les opérations élémentaires :

```
+ - * /
```

21. Il existe en particulier des instructions permettant de gérer les fichiers, les entrées/sorties, etc.

De plus, les opérateurs Mod et \ permettent de déterminer respectivement le reste et le quotient d'une division entière. Enfin & permet de concaténer deux chaînes de caractères [22].

4.1.2 Opérateurs de comparaison.

Les opérateurs suivants permettent de réaliser les comparaisons classiques :

```
= <> < > <= >=
```

Il n'y a donc pas de différence entre l'opérateur d'affectation et l'opérateur de comparaison. Les opérateurs *Like* et *Is* permettent, de plus, respectivement de comparer deux chaînes de caractères et deux variables de type objet. Consulter l'aide en ligne pour plus de précisions.

4.1.3 Opérateurs logiques.

Les principaux opérateurs suivants permettent les opérations classiques :

```
Not And Or Xor
```

4.2 Structures conditionnelles

4.2.1 If Then Else ElseIf

L'instruction *If Then Else* permet d'écrire une structure conditionnelle. La syntaxe est la suivante [23] :

```
If <condition> then
  <instructions>
Else
  <instructions>
End If
```

Lorsque la condition est réalisée, la première suite d'instructions est exécutée. Dans le cas contraire, c'est bien sûr la deuxième suite d 'instructions qui est exécutée. Noter que l'instruction *Else* est facultative.

Dans le cas où il est nécessaire d'imbriquer plusieurs structures conditionnelles, on peut utiliser l'instruction *ElseIf* :

```
If <condition> Then
  <instructions>
ElseIf <condition>
  <instructions>
End If
```

22. On peut également pour cela utiliser l'opérateur +.

23. Lorsque la structure conditionnelle ne présente qu'une instruction, il est également possible de l'écrire sur une seule ligne en omettant l'instruction *End If*. La syntaxe proposée ici a l'avantage d'être plus claire et de permettre le rajout ultérieur d'instructions supplémentaires.

4.2.2 Select Case

L'instruction *Select Case* permet d'écrire une structure conditionnelle dans laquelle une expression doit être comparée à plusieurs valeurs. Elle remplace alors avantageusement l'instruction *ElseIf*. La syntaxe est la suivante :

```
Select Case <variable>

  Case valeur1:
    <instructions>
  Case valeur2:
    <instructions>
  ...
  Case Else:
    <instructions>

End Select
```

Lorsque la variable est égale à une valeur répertoriée, les instructions correspondantes sont exécutées, et l'instruction *Select Case* se termine. La ligne *Case Else* permet d'inclure toutes les occurrences de la variable non répertoriées auparavant. Elle est facultative.

4.3 Boucles

4.3.1 Do While Loop

L'instruction *Do While Loop* permet de réaliser une boucle conditionnelle. La syntaxe est la suivante :

```
Do While <condition>
  <instructions>
Loop
```

La condition est évaluée au début du traitement. Si elle est vraie, les instructions sont exécutées. Ensuite la condition est réévaluée, et ainsi de suite.

4.3.2 Do Until Loop

L'instruction *Do Until Loop* est similaire à *Do While Loop*. Cependant, les instructions contenues dans le corps de l'instruction sont d'abord exécutées, à la suite de quoi la condition est évaluée. La syntaxe est donc la suivante :

```
Do
  <instructions>
Loop Until <condition>
```

4.3.3 For Step Next

L'instruction *For Step Next* est utilisée pour répéter une action selon un nombre d'itérations déterminé. La syntaxe est la suivante :

```
For <variable> = valeur1 To valeur2 Step <pas>
  <instructions>
Next <variable>
```

Step sert à indiquer le pas d'itération. Cette précision est toutefois facultative : par défaut, la valeur du pas utilisé est 1.

4.3.4 For Each In Next

For Each In Next est une variante de l'instruction *For Step Next*. Elle permet de réaliser une itération en parcourant tous les objets contenus dans une collection[24]. La syntaxe est la suivante :

```
For Each <objet> In <collection>
  <instructions>
Next <object>
```

Le code suivant permet par exemple de renommer toutes les feuilles de calcul du classeur actif :

```
Dim i As Integer
Dim feuille As sheet
...
i = 0
For Each feuille In ActiveWorkbook.Sheets
  i = i + 1
  'Cstr (i) permet de convertir l'entier i en une chaîne de caractères
  feuille.Name = "Feuil" + CStr(i)
Next feuille
```

A noter dans le programme ci-dessus la ligne de commentaire introduite par une apostrophe.

4.4 Gestion des programmes

4.4.1 Exit

L'instruction *Exit* permet de sortir de la séquence en cours. Par exemple, pour terminer prématurément une procédure, on écrira :

```
Sub essai()
  ...
  If <condition> Then
    Exit Sub
```

24. cf. section 1.3 (page 3).

```
  End If
  ...
End Sub
```

De même, les commandes *Exit Do* et *Exit Next* permettent de quitter une structure conditionnelle et une boucle *For Next*.

4.4.2 End et Stop

L'instruction *End* permet de mettre fin à une séquence, comme on l'a déjà vu avec les instructions *End Sub*, *End Function* ou *End If*.

L'emploi de *End* de façon isolée est plus intéressant et permet de mettre fin au programme en cours, en éliminant de la mémoire les variables utilisées, les boîtes de dialogue éventuellement chargées, les fichiers ouverts par la commande *Open*, etc. En effet, par défaut, toutes ces informations sont conservées en mémoire à la fin de l'exécution d'un programme.

L'instruction *Stop* stoppera de façon analogue le programme en cours, mais conservera cette fois l'ensemble des informations en cours.

4.4.3 On Error GoTo

L'instruction *On Error Goto* permet de gérer les erreurs qui peuvent apparaître dans un programme. Le programmeur a ainsi la possibilité de prévoir les erreurs possibles, en réagissant en conséquence. La syntaxe est la suivante :

```
Sub essai()
  On Error GoTo <étiquette>
  ...
<étiquette>:
  'traitement de l'erreur
End Sub
```

L'instruction *On Error GoTo 0* permet à tout moment d'invalider le contrôle des erreurs par la séquence *On Error GoTo*. Pour traiter correctement l'erreur, il est possible d'utiliser la propriété *Number* de l'objet *Err*. Celle-ci renvoie le numéro d'erreur[25]. Enfin, l'instruction *Resume* permet de reprendre le cours normal du programme en ignorant la cause de l'erreur et la commande *On Error Resume Next* permet d'ignorer systématiquement les erreurs, en sautant les instructions correspondantes.

Voici une illustration du mécanisme de traitement des erreurs[26] :

```
Sub essai()
  Dim i As Integer
```

25. Consulter l'aide pour connaître l'erreur correspondante.

26. L'instruction *MsgBox* permet d'afficher automatiquement une boîte de dialogue. Consulter l'aide en ligne pour plus d'informations.

```
  On Error GoTo erreur
  'ouverture du fichier 'classeur1.xls'
  Workbooks.Open ("classeur1.xls")
  ...
erreur:
  'message d'information
  i = MsgBox("Impossible d'ouvrir le fichier 'classeur1.xls'!", vbCritical)
  'reprise du programme
  Resume
End Sub
```

4.5 Conversion de types

Il est parfois nécessaire d'opérer des conversions de types. VBA permet d'effectuer ces opérations grâce à une série d'instructions, dont voici quelques exemples :

- *CInt* : conversion en type de données entier ;
- *CStr* : conversion en type de données string ;
- *CBdl* : conversion en type de données double ;
- *CDate* : conversion en type de données date ;
- *CVar* : conversion en type de données variant.

Un exemple d'utilisation d'une telle instruction a été donné à la section 4.3.4 (page 13).

5 Procédures et fonctions

Un programme est généralement composé d'une succession de sous-programmes, réalisant des actions indépendantes. Ces sous-programmes sont appelés *procédures* ou *fonctions*. La différence entre les deux est qu'une fonction renvoie une valeur à l'inverse d'une procédure.

5.1 Les procédures

L'instruction *Sub* permet de déclarer une procédure. Par défaut, une procédure est de type *Public*, c'est-à-dire qu'on peut l'utiliser à partir de tous les modules. En revanche, l'utilisation du mot clé *Private* restreint la validité de la déclaration au module dans lequel la procédure est déclarée.

Voici deux exemples :

```
Sub ma_procédure()
  'déclarations des variables éventuelles
  ...
  'corps d'une procédure publique
```

```
  ...
End Sub


Private Sub ma_procédure()
  'déclarations des variables éventuelles
  ...
  'corps d'une procédure privée
  ...
End Sub
```

5.2 Les fonctions

De façon analogue aux procédures, l'instruction *Function* permet de déclarer une fonction. Les règles de portée qui s'appliquent sont les mêmes que dans le cas des procédures. Le type de la valeur que renvoie la fonction doit être déclaré comme le montre l'exemple suivant :

```
Function ma_fonction() As Boolean
  'déclarations des variables éventuelles
  ...
  'corps d'une fonction publique renvoyant un booléen
  ...
End Function
```

5.3 Passage d'arguments par valeur ou par référence

Il est possible de passer des arguments à une procédure ou une fonction. L'appel au sous-programme considéré peut être réalisé grâce à l'instruction *Call* suivie du nom du sous-programme et de la liste des arguments placés entre parenthèses :

```
Call ma_fonction(23)
```

On distingue deux types de mécanismes en ce qui concerne le passage d'arguments : le passage par *valeur* et le passage par *référence*. Lorsqu'une variable est transmise par référence, la procédure ne peut modifier la variable concernée : elle ne travaille en fait que sur une copie temporaire de la variable en question. A l'inverse, lors d'un passage par valeur, la variable considérée est réellement utilisée par la procédure, qui est donc susceptible de la modifier.

Par défaut, les variables sont transmises par référence. Pour transmettre un argument par valeur, il faut utiliser le mot clé *ByVal*, comme le montre l'exemple suivant :

```
Sub référence(i As Integer)
  'passage par référence
  i = 1
```

```
End Sub

Sub valeur(ByVal i As Integer)
  'passage par valeur
  i = 2
End Sub

'procédure principal
Sub essai()
  Dim i As Integer
  i = 0
  Call valeur(i)
  'i vaut 0, malgré l'appel précédent
  Call référence(i)
  'i vaut 1 maintenant
End Sub
```

5.4 Les procédures évènements

Les procédures évènements sont des procédures particulières. Comme on l'a vu à la section 2.6 (page 6), ce type de procédures est implémenté dans des modules de classe.

Elles servent à définir les actions devant intervenir pour un certain nombre d'évènement préprogrammés et gérés automatiquement par Excel.

L'éditeur permet de déterminer dans chaque module quels sont les évènements qu'on peut gérer pour un objet donné.

A cet effet, il suffit d'utiliser le sélecteur d'objets en haut à gauche de chaque module de classe. Une fois l'objet souhaité sélectionné, on pourra sélectionné l'évènement désiré grâce à la liste déroulante de droite. Excel crée alors directement le corps de la procédure correspondante. Il est à noter que parfois les procédures évènements ont également des arguments. Ces arguments sont utilisés en interne par Excel, comme l'illustre l'exemple suivant :

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
  'si le classeur actif n'est pas protégé, Cancel devient False
  'Excel interromp alors le processus de fermeture du fichier
  Cancel = Not (ActiveWorkbook.Protect)
End Sub
```

6 Conception de programme

6.1 Quelques conseils

Comme pour tous les langages de programmation, il convient de ne pas sous-estimer la difficulté des programmes que l'on développe. On insérera ainsi le maximum de commentaires explicitant le code.

```
'Ceci est un commentaire pertinent
```

En VBA, les commentaires peuvent être saisis n'importe où après un apostrophe : la ligne suivant une apostrophe sera ignorée lors de la compilation. Par défaut, les commentaires apparaissent en vert dans l'éditeur.

De plus, afin de préserver la lisibilité du programme, on aura tout intérêt à indenter rigoureusement le code source et à ne saisir qu 'une instruction par ligne[27]. Dans le cas où une ligne est trop longue, il est possible de la «couper» en deux grâce à un underscore _.

Enfin, il est conseillé de sauvegarder le plus souvent possible son travail, surtout s'il s'agit d'un projet relativement complexe.[28].

6.2 Utiliser des références

De nombreuses bibliothèques sont disponibles dans Excel et permettent de réaliser des opérations particulières. Pour visualiser les bibliothèques auxquelles le projet en cours fait référence, utiliser la commande **Outils/Références...** de l'éditeur.

Certaines bibliothèques contiennent le corps du langage VBA et sont intégrées par défaut au projet en cours. C'est le cas par exemple de la bibliothèque *Visual Basic For Applications*. Les autres sont utilisées pour des utilisations spécifiques et ne doivent être référencées qu'en cas de besoin[29].

Par exemple, la bibliothèque *Microsoft DAO 3.51 Object Library* permet de gérer depuis Excel une base de données Access. L'ajout d'une référence rajoute une nouvelle collection d'objets, qui apportent de nouvelles fonctionnalités à Excel.

6.3 Débogage de programme

L'éditeur VBE offre de nombreuses possibilités pour déboguer un programme. Il est par exemple possible de tester une macro en mode *pas à pas*, d'utiliser des *points d'arrêt*, ou encore des *espions*.

Le mode pas à pas permet d'exécuter le programme instruction par instruction, la touche F8 entraînant l'exécution de l'instruction suivante. Les points arrêts permettent d'interrompre temporairement l'exécution du programme le temps de visualiser par exemple le contenu des variables, grâce à un espion[30].

La touche F5 permet de relancer l'exécution du programme jusqu'au prochain point d'arrêt.

---

27. Il est cependant possible de saisir plusieurs instructions par ligne en les séparant par un double point.

28. Excel se révèle en effet parfois assez instable ; une erreur fatale peut toujours se produire.

29. La référence à une bibliothèque consomme en effet de la mémoire.

30. Il est à noter que, quand l'exécution d'un programme est interrompue, il est possible de visualiser le contenu d'une variable en pointant sur l'expression considérée le pointeur de la souris. Pour les expressions complexes, il est préférable d'utiliser des espions.

Dans la plupart des cas, il est possible de corriger en temps réel les erreurs détectées. On peut alors relancer l'exécution du programme sans être obligé d'initialiser le programme. Avant de tester l'exécution d'un programme, utiliser la commande **Débogage/Compiler <projet>** [31] et corriger les erreurs signalées, pour gagner du temps.

Enfin, la touche ESC permet d'interrompre le programme pendant son exécution.

6.4 Les macros complémentaires

Les macros complémentaires ne sont rien d'autres que des programmes VBA, réalisant des tâches particulières ou fournissant des fonctions supplémentaires. Les macros utilisées dans la session en cours peuvent être visualisées par la commande **Outils/Macros complémentaires...** de l'éditeur. Comme dans le cas des références, il ne faut les activer qu'en cas de besoin. Par exemple, la macro complémentaire *Utilitaire d'analyse–VBA* fournit des fonctions et procédures VBA, qui vous permettent d'utiliser de nombreuses fonctions financières et des outils d'analyse statistique.

L'un des principaux avantages d'une macro complémentaire est qu'elle est pré compilée, ce qui accélère son exécution.

Les macros complémentaires peuvent être réalisaées très facilement, en enregistrant simplement le programme développé en tant que macro complémentaire. Pour utiliser le fichier généré, il faudra ensuite activer la macro grâce à la commande **Outils/Macros complémentaires...**. Le programmeur doit cependant garder à l'esprit un certain nombre de remarques :

- une macro complémentaire ne peut être modifiée. Il faut donc conserver une version sous la forme d'un fichier classique pour pouvoir apporter des modifications à la macro par la suite ;
- le classeur d'une macro complémentaire ne peut être modifié : si l'on veut stocker des informations de manière permanente, il faudra les enregistrer éventuellement dans un autre classeur ; de même, il faut généralement prévoir des boîtes de dialogue pour gérer l'interface utilisateur ;
- la macro complémentaire n'est pas le classeur actif : les références aux feuilles de la macro doivent être précédées par exemple du nom du classeur.

7 Un exemple pratique

On se propose de réaliser une macro permettant la saisie d'informations à l'écran. Plus précisément, on souhaite concevoir une boîte de dialogue dans

31. Il vaut mieux prendre l'habitude de sauvegarder son travail avant de lancer la compilation. Celle-ci peut en effet parfois entraîner des erreurs fatales...

laquelle l'utilisateur rentrera son nom, sa qualité et sa date de naissance. Les informations saisies devront alors être reportées dans une feuille de calcul[32].

7.1 Conception de la boîte de dialogue

La première étape consiste à créer la boîte de dialogue. Pour cela, insérer une nouvelle boîte de dialogue sous l'éditeur en utilisant la commande **Insertion/UserForm**. Faire ensuite apparaître la boîte à outils grâce à la commande **Affichage/boîte à outils**. Créer deux zones d'édition (*TextBox*) pour la saisie du nom et de la date de naissance, en indiquant avec deux zones de texte (*Label*) le nom du champ considéré. Pour la saisie de la qualité, une liste déroulante permettant le choix entre plusieurs champs prédéfinis (Mlle, Mme et M.) est tout indiqué. Créer enfin les deux boutons classiques *OK* et *Annuler*.

Penser à définir correctement les propriétés *Cancel* et *Default* des boutons[33]. Ne pas oublier enfin de donner un titre à la boîte de dialogue (propriété *Caption*) et également de se préoccuper de l'ordre de tabulation[34].

Lors de la conception d'une boîte de dialogue, utiliser la barre d'outils *Userform* et le menu **Format** de l'éditeur pour améliorer sa mise en forme.

La boîte de dialogue finale devrait ressembler à la figure 1.

FIG. 1 – *Un exemple de boîte de dialogue*

Une fois la boîte de dialogue créée, il reste à affecter des noms personnalisés aux différents contrôles utilisés. Cette opération n'est pas obligatoire

32. L'intérêt d'un tel programme est avant tout pédagogique...

33. Ces deux propriétés définissent quel bouton est activé par les touches RETURN et ESC. Affecter ici *Cancel* à *True* pour le bouton Annuler et *Default* à *False* pour le bouton OK. Procéder à l'opération inverse pour le bouton Annuler

34. La commande **Affichage/Ordre de tabulation** permet de définir l'ordre dans lequel l'utilisateur pourra se déplacer parmi les contrôles grâce à la touche TAB.

mais facilitera la programmation par la suite. Pour cela, afficher les propriétés de chaque objet[35] et modifier le champ nom. On choisira par exemple les noms suivants :

- zone de saisie du nom : nom ;
- zone de saisie de la date de naissance : date_naissance ;
- zone déroulante : qualité ;
- bouton OK : bouton_ok ;
- bouton Annuler : bouton_annuler ;
- boîte de dialogue elle-même : boîte_saisie.

Il est inutile d'affecter un nom aux zones de texte, dans la mesure où elles ne seront pas manipulées dans le code par la suite.

7.2 Programmation des contrôles de la boîte

Une fois la boîte créée, il est nécessaire de programmer le comportement des différents contrôles. Il faut pour cela inclure du code dans le module de classe associé à la boîte de dialogue[36].

Il est nécessaire tout d'abord de prévoir, lors de l'initialisation de la boîte de dialogue, le remplissage de la zone déroulante. Ensuite, il faut programmer le comportement des deux boutons créés. Lorsque l'utilisateur appuie sur Annuler, il faut fermer la boîte de dialogue. Un clic sur OK devra provoquer la fermeture de la boîte de dialogue et l'écriture des informations saisies dans une feuille de calcul. Voici le code permettant de réaliser ces traitements :

```
'force la déclaration explicite des variables
Option Explicit

'procédure évènement gérant un clic sur le bouton OK
Private Sub bouton_ok_Click()
    'écriture des saisies dans la feuille de calcul
    'Feuil1 (cellules A1, B1 et C1)
    With Sheets("Feuil1")
        .Range("A1").Value = qualité.Text
        .Range("B1").Value = nom.Text
        .Range("C1").Value = date_naissance.Text
    End With

    'fermeture de la boîte de dialogue
    Hide
End Sub
```

35. Pour cela, faire apparaître la fenêtre propriétés grâce à la commande **Affichage/Fenêtres Propriétés** et sélectionner simplement chaque objet.

36. cf. section 2.6 (page 6)

```
'procédure évènement gérant un clic sur le bouton Annuler
Private Sub bouton_annuler_Click()
    'fermeture de la boîte de dialogue
    Hide
End Sub

'procédure évènement s'exécutant à l'apparition de la boîte de dialogue
Private Sub UserForm_Initialize()
    'remplissage de la zone déroulante
    With qualité
        .AddItem ("Mlle")
        .AddItem ("Mme")
        .AddItem ("M.")

        'valeur par défaut
        'les valeurs d'une liste déroulante sont repérées avec des indices
        'dont le premier commence à 0. Ecrire 1 pour "Mme" et 2 pour "M."
        .ListIndex = 0
    End With
End Sub
```

Pour remplir la liste déroulante, on peut également écrire les trois champs concernés dans une feuille de calcul et affecter les plages concernées à la propriété *RowSource* de la liste déroulante. Si on saisit par exemple les champs dans les cellules *A1*, *A2* et *A3* de la feuille *Feuil2*, on peut affecter directement la valeur *Feuil2!A1:A3* à cette propriété.

A noter que dans le module de classe, il est implicite que les objets cités font référence à la boîte de dialogue. Il n'est donc pas nécessaire de préciser le nom de la boîte de dialogue à chaque fois qu'on manipule un contrôle, en écrivant par exemple :

```
Sheets("Feuil1").Range("A1").Value = Boîte_saisie.qualité.Text
```

Le langage VBA permet de nombreuses écritures implicites de ce type. Par exemple, dans le code ci-dessus, la référence à la feuille *Feuil1* sous-entend qu'il s'agit de la feuille Feuil1 du classeur actif.

Il est enfin nécessaire de créer une procédure servant à afficher la boîte de dialogue de saisie. C'est en quelque sorte le programme principal. Cette procédure n'est pas rattachée à un objet quelconque. On doit donc la programmer dans un module général[37].

Voici la procédure en question :

```
Public Sub saisie()
    'lancement de la boîte de dialogue
    Boîte_saisie.Show
End Sub
```

37. Utiliser la commande **Insertion/Module** pour insérer un nouveau module.

Cette procédure doit être publique pour pouvoir être exécutée par exemple par la commande **Outils/Macro/Macros...**.

7.3 Améliorations possibles

Le programme précédent est satisfaisant. Cependant, il présente quelques imperfections. Par exemple, il serait souhaitable que le programme vérifie que l'utilisateur a bien saisi les informations demandées.

De plus, il serait bon de vérifier que l'utilisateur a bien rentré une date compréhensible par Excel. Voici le code de la nouvelle procédure *bouton_ok_ Click* tenant compte de ces modifications :

```
'procédure évènement gérant un clic sur le bouton OK
Private Sub bouton_ok_Click()
    Dim i As Integer

    If nom.Text <> "" And date_naissance.Text <> "" Then

        'vérifie si la date entrée est valide

        If Not (IsDate(date_naissance.Value)) Then
            'message d'erreur
            'MsgBox sert à afficher un message au moyen d'une boîte
            'de dialogue automatique, consulter l'aide en ligne pour
            'plus de renseignements
            i = MsgBox("La date saisie est incorrecte!", vbCritical)

            'effacement de la saisie et positionnement du focus sur
            'le contrôle date_naissance
            With date_naissance
                .SetFocus
                .Text = ""
            End With

            'fin de la procédure
            Exit Sub

        End If

        'écriture des saisies dans la feuille de calcul
        'Feuil1 (cellules A1, B1 et C1)
        With Sheets("Feuil1")
            .Range("A1").Value = qualité.Text
            .Range("B1").Value = nom.Text
            .Range("C1").Value = date_naissance.Text
        End With

        'fermeture de la boîte de dialogue
        Hide
```

```
    Else
        'message d'erreur
        i = MsgBox("Veuillez entrer les données manquantes!", _
                    vbCritical, "Erreur")
    End If
End Sub
```

7.4 Création d'une commande spécifique

On souhaite maintenant créer une commande spécifique qui déclencherait la procédure de saisie. Il est possible de réaliser cette opération manuellement ou par programmation. Voyons ici la méthode manuelle :

- activer la commande **Affichage/Barres d'outils/Personnaliser...** ;
- créer une nouvelle barre d'outils (utilisez le bouton *Nouvelle...* de l'onglet *Barre d'outils*) ;
- cliquer sur l'onglet *Commandes* et choisir la catégorie *Macros*;
- cliquer sur la commande *Élément de menu personnalisé* et, tout en laissant appuyé le bouton de la souris, déplacer l'élément dans la nouvelle barre d'outils[38] ;
- modifier le nom du nouveau menu introduit et lui affecter la macro *Saisie* en utilisant le menu contextuel du nouveau menu.

8 Pour aller plus loin

8.1 Utiliser l'aide en ligne !

Ce guide n'est qu'une introduction au langage VBA. Le sujet est néanmoins particulièrement vaste et de nombreux aspects du langage n'ont pas été abordé. Pour aller plus loin, consulter l'aide en ligne et utiliser l'explorateur d'objets, pour se retrouver dans la hiérarchie des objets.

8.2 Quelques références

Cette introduction a été rédigée en grande partie grâce à *La Bible Microsoft Excel 97*[39]. Malgré des imperfections, ce livre constitue une bonne référence pour les personnes voulant s'initier à la programmation VBA.

Sur Internet, on pourra consulter avec profit le site de L. Longre à l'adresse suivante :

`http://longre.free.fr/`

38. Il suffit pour cela de se positionner au dessus de la barre d'outils et de relâcher le bouton de la souris.

39. La référence exacte est : *Microsoft Excel 97 – Expertise et Programme* de A. Dahms, B. Van almsick et G. Hagemeister, collection La Bible, Micro Application, 1997.

Ce site est consacré à Excel en général et à la programmation VBA en particulier. Il contient une mine d'informations, propose des exemples à télécharger et recense également un grand nombre de liens sur Excel.

On pourra aussi visiter le site de Philippe Noss, développeur informatique indépendant, qui propose des exemples, des trucs et astuces, des liens, etc., sur Excel. L'adresse est la suivante :

`http://www.cybercable.tm.fr/~pnoss/index.html`

On peut également participer au forum suivant consacré à Excel et à la programmation VBA (en langue française) :

`microsoft.public.excel.fr`.