Initiation à la programmation en Basic

Table des matières

1. Introduction

1.1. Objectifs du cours

Le but du cours n'est pas d'appendre un langage particulier. Il est de donner des notions générales de programmation utilisables avec différents logiciels et différents langages.
Nous nous y attacherons à apprendre la logique de programmation plutôt que d'étudier une syntaxe bien précise.
Ceci dit, nous sommes bien obligés de choisir un langage pour cet apprentissage. Ce choix entraîne aussi le choix d'un environnement de programmation.
Le langage choisi est dérivé du langage Basic (*Beginner's All-purpose Symbolic Instruction Code*) parce que c'est un langage pour débutant et qu'il permettra par la suite à ceux qui auront suivi ce cours de l'utiliser dans le cadre de l'utilisation des logiciels bureautiques les plus courants (Word, Excel et Access). Nous utiliserons le langage Basic sous trois formes différentes : le QBasic, le QuickBasic et le VisualBasic.
Le QBasic et le QuickBasic seront utilisés dans un premier temps pour se familiariser à la programmation procédurale. Ces environnements de programmations, déjà anciens puisqu'ils datent du milieu des années 80, conviennent cependant très bien pour découvrir un langage de programmation, aborder les notions de variables (types, déclarations, assignations) et la logique de programmation (structures de tests, boucles, fonctions et procédures).
Nous ferons avec ces langages de la programmation procédurale dans un environnement DOS. Ces outils de programmation « QB » fonctionnent en mode invite de commande. Le QBasic tient sur une sur une seule disquette et est interprété. Le QuickBasic est une version plus complète qui permet la création de fichiers exécutables pouvant fonctionner indépendamment de l'environnement de programmation mais toujours en mode « invite de commande ».

Dès que les notions fondamentales de programmation procédurales seront vues, nous passerons à la programmation événementielle avec le Visual Basic. De nouvelles notions apparaîtront alors : objets, propriétés, méthodes, événements …

Ce même langage pourra aussi être utilisé pour l'écriture de script en VbScript dans différents contextes d'exécution :

- des pages Web (lisibles uniquement sur Internet Explorer)
- en utilisation directe sous Windows dans l'environnement de programmation de WSH *Windows Scripting Host*
- ou encore pour programmer certaines fonctions avancées dans les applications de Microsoft Office : VBA *Visual Basic for Applications*

2. Langages interprétés ou compilés

Pour être exécutable un programme doit être compréhensible par l'ordinateur qui ne peut décoder que des 1 et des 0. Ce code, le langage machine, est loin de notre langage de tous les jours. Notre langage parlé est riche, nuancé mais trop ambigu pour l'ordinateur qui ne dispose pas de l'intuition nécessaire pour le comprendre.
Il nous faut donc un langage artificiel utilisable par l'homme et la machine. Les langages de programmation sont nombreux mais il est inutile d'être polyglotte en la matière pour être un expert en programmation. Tous ces langages ont de nombreux points communs, c'est à ceux-ci que nous nous intéresserons. Nous insisterons sur les principes transposables d'un langage de programmation à l'autre en laissant de côté les particularités trop spécifiques aux langages que nous utilisons pour notre apprentissage.
Le langage de programmation ressemble donc à notre langage parlé.
Le code écrit dans ce langage est appelé code source. Il doit être traduit en langage machine pour pouvoir être exécuté par l'ordinateur. Il y a deux manières de réaliser cette traduction : la compilation et l'interprétation.

Compilation Le code source est entièrement traduit pour donner un code binaire, appelé code « objet »[1] Il se présente alors sous la forme d'un fichier dont l'extension est souvent « .obj » Différents fichiers objets obtenus à partir de diverses compilations de fichiers sources (qui n'ont pas nécessairement été écrits dans le même langage) sont alors rassemblés pour former un code exécutable (extension « .exe »)

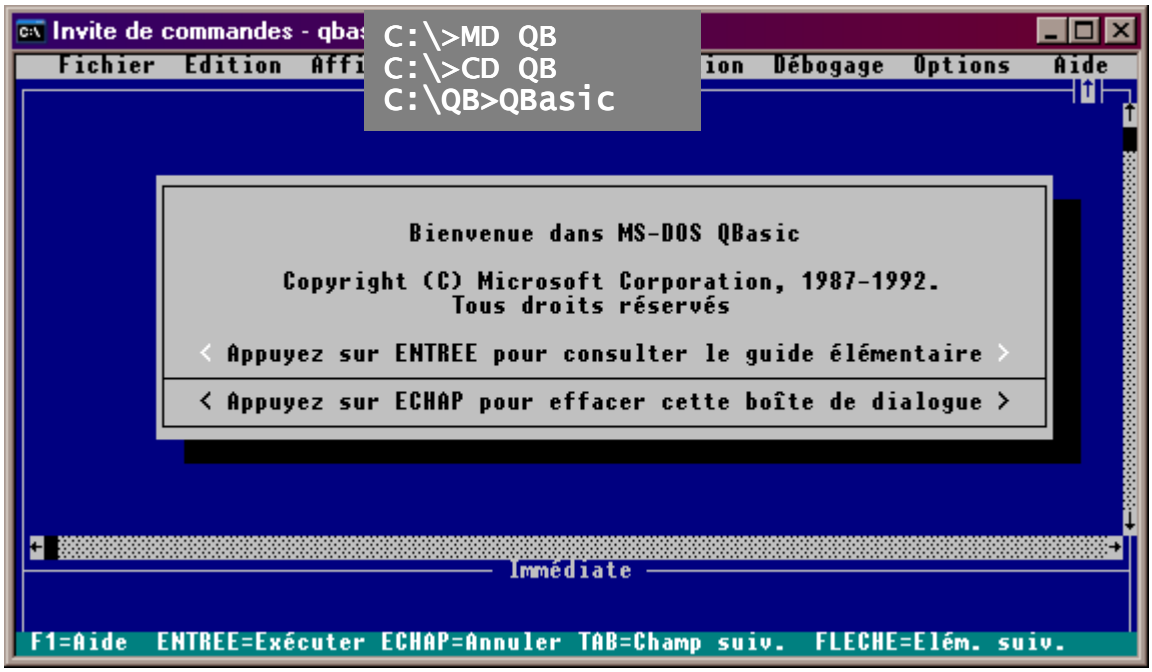
Interprétation Chaque ligne du code source est interprétée puis exécutée immédiatement avant de passer à l'interprétation de la suivante. Cette méthode est plus lente mais elle est aussi plus facile pour le débutant puisque le programme s'exécute instantanément sans devoir passer par les étapes de la compilation. Ces étapes constituent une difficulté qui s'ajoute à celle de l'apprentissage du langage.

Le QBasic est un langage interprété. Le QuickBasic et le VisualBasic peuvent à la fois être interprété et compilés.

[1] Le mot « objet » n'a rien à voir ici avec la programmation orientée objet.

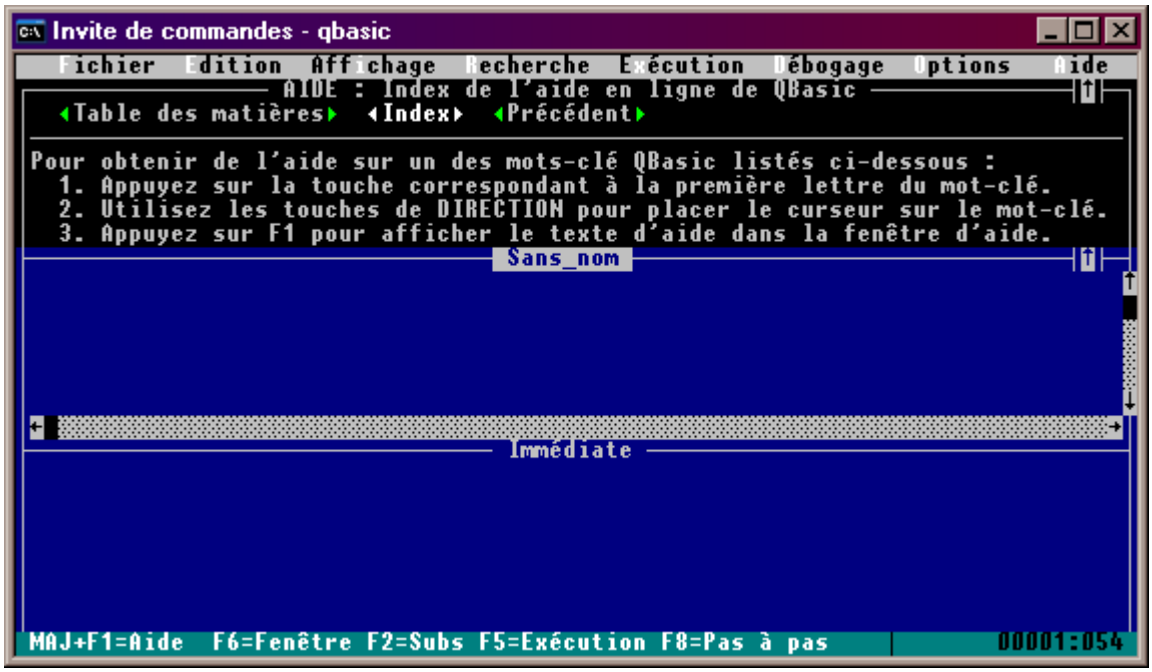
3. Installation de QBasic

Le QBasic tient dans seulement deux fichiers. Vous les trouverez dans les CD-ROM d'installation de Windows 95 dans le répertoire \OTHER\OLDMSDOS ou dans le CDROM d'installation de Windows 98 dans le répertoire \TOOLS\OLDMSDOS.
Il s'agit des fichiers QBASIC.EXE et QBASIC.HLP. Les deux fichiers réunis ne font pas plus de 329 Ko. C'était avant les manigances de l'alliance Windows-Intel pour nous pousser à la consommation en produisant des logiciels toujours plus volumineux et s'assurer ainsi que nos ordinateurs seront toujours plus vite obsolètes !
Copiez ces deux fichiers dans votre répertoire Windows ou WINNT suivant ce que vous avez comme système d'exploitation.
Passez alors en mode invite de commande.
Créez un répertoire où vous enregistrerez vos projets en QBasic.
Placez-vous ensuite dans ce répertoire pour appeler QBasic.



4. L'environnement de programmation QBasic

L'environnement de programmation sert à la fois d'éditeur de programme, d'interpréteur et de debuger (programme de mise au point).
Cet environnement vous donne accès à 3 types de fenêtre.



- La fenêtre d'aide dont le fond est noir se referme en pressant la touche ESC.
- La fenêtre de programmation dont le titre est « sans-nom » tant que le programme qui s'y trouve n'a pas été sauvé sous un autre nom.
- La fenêtre immédiate à partir de laquelle nous testerons des instructions une à une.

Les fenêtres peuvent être redimensionnées en faisant glisser la barre de séparation à l'aide de la souris ou en pressant les touches Alt + ou Alt – pour les agrandir ou les réduire.

5. Concepts fondamentaux de programmation

5.1. Les variables

Les données sont stockées dans des variables. Ces variables sont des emplacements dans la mémoire de l'ordinateur. Elles ont un nom et un type (fixes) elles contiennent une valeur qui elle peut varier.
Le nom d'une variable permet de retrouver son emplacement en mémoire. Il équivaut à son adresse.
Le type de la variable détermine l'espace à lui réserver en mémoire et les traitements qui lui sont appropriés. Une addition entre nombres entiers ne se fait pas avec les mêmes algorithmes que si ces nombres sont en format « virgule flottante ».

5.2. Types de variables

Nous rencontrerons ici trois types de variables: les entiers, les réels et les chaînes de caractères.

	Nom du type	Taille	Min et max
Entier	INTEGER	2 octets	-32 768 à + 32 767
	LONG	4 octets	- 2 147 483 648 à + 2 147 483 647
Réel	SINGLE	4 octets	$\pm 1.40\ 10^{-45}$ à $\pm 3.40\ 10^{+38}$
	DOUBLE	8 octets	$\pm 4.94\ 10^{-324}$ à $\pm 1.79\ 10^{+308}$
Chaîne de caractères	STRING		

Les types de variables diffèrent d'un langage à l'autre. Tous ont des types « entier » et « réel » Par contre, en langage C par exemple il n'y a pas de type chaîne de caractère. D'autres langages tels que le VisualBasic supportent des types supplémentaires comme les types DATE et BOOLEAN.

5.3. Déclaration de variables

Déclarer une variable, c'est lui réserver un emplacement en mémoire auquel on donne un nom et en indiquant son type (autrement dit sa taille et la façon de la traiter).
Voici comment déclarer ces variables en BASIC :

```
DIM NomVariable AS type
```

Exemples :

```
DIM  Age AS INTEGER
DIM  Nom AS STRING
DIM  Candidat AS STRING
DIM  Rue AS STRING, Numero AS INTEGER, Localite AS STRING
```

NB : La déclaration des variables est facultative en QB mais il est pourtant parfois nécessaire de faire la distinction entre les variables de type numérique et celles qui contiennent des chaînes de caractères.
Le QuickBasic permet de faire cette distinction implicitement en ajoutant au nom de la variable un caractère de terminaison $, %, &, ! ou # selon que le type de la variable est une chaîne de caractère($), un entier court (%), un entier long (&), un réel single (!) ou un réel double (#). Ces déclarations implicites sont dépassées et ne sont jamais utilisées dans ces notes. Elles sont cependant mentionnées ici car vous risquez de les rencontrer en recopiant des programmes téléchargés sur d'autres sites.

5.4. Noms des variables

A ce sujet encore, chaque langage a ses propres conventions. D'une manière générale, un nom de variable commence toujours par une lettre et est suivi d'autres lettres et/ou de chiffres. Certains langages autorisent d'autres types de caractères. Le caractère « souligné » est généralement admis mais les autres caractères spéciaux sont soit interdits ou ont des significations très particulières. Le mieux, est donc de ne jamais utiliser de caractères spéciaux (ponctuation, #, $ etc.) ni de caractères accentués.
Les noms d'instructions sont réservés, ils ne sauraient convenir comme nom de variable.
La taille maximum des noms des variables varie aussi d'un langage à l'autre. En QB, cette taille est limitée à 40 caractères c'est bien suffisant.

5.5. Assignations

Assigner une valeur à une variable, c'est écrire cette valeur dans l'emplacement mémoire réservé à cette variable.

```
AGE = 19           'assignation avec une valeur constante
Nom = "Lambert"    'la valeur est ici une chaîne de caractères
Candidat = Nom     'la valeur est issue d'une autre variable
N1 = (N1 + N2)/2   'la valeur est ici une expression
```

5.6. Opérateurs

Opérateur	Description	Exemple	Résultat
+ -	Addition Soustraction	3 + 7 7.5 – 5	10 2.5
* /	Multiplication Division	2.54 * 2 5.2 / 2	5.08 2.6
\ **MOD**	Division entière Reste de la division entière	17 \ 3 17 MOD 3	5 2
^	Elévation à la puissance	2 ^ 10 25 ^ (1/2)	1024 5
+	Concaténation de chaînes	∀Bart∀ + ∀Simpson∀	∀Bart Simpson∀

Exercices

Ex 5a Les noms de ces variables sont-ils valides ?

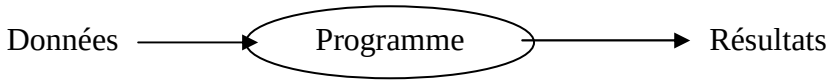
Variables	Oui / Non	Pourquoi ?
Num007		
4000LIEGE		
Tartempion		
Jean-Paul		
Cls		
Nom		
Prénom		
N°		

Ex 5b Déterminez les valeurs de A, B et C après exécution des instructions suivantes :

```
DIM A AS INTEGER, B AS INTEGER, C AS INTEGER
A = 2
B = 3
C = A + B
A = C + 2 * B
B = (C + B) ^ 2
C = C \ 3
```

5.7. Entrées / Sorties

Considérons à présent qu'un programme est une « moulinette » où entrent des données et d'où sortent des résultats.



La première chose à faire pour expérimenter un langage est donc d'apprendre comment entrer une donnée et sortir un résultat à l'écran.

PRINT

Instruction	Résultat
`PRINT "Salut Hervé"`	`Salut Hervé`
`PRINT " 3 + 4 "`	`3 + 4`
`PRINT 3 + 4`	`7`
`PRINT Age + 1`	`20`
`PRINT "Bonjour " + Nom`	`Bonjour Lambert`
`PRINT "Age = " ; Age`	`Age = 19`

NB. Le point-virgule [;] sert de séparateur dans la commande PRINT pour imprimer plusieurs données les unes à la suite des autres. Il se met aussi à la fin de la commande PRINT quand on souhaite que celle-ci ne soit pas suivie d'un retour à la ligne.

INPUT

```
INPUT "Quel est ton nom ? ", Nom
```

Cet exemple suffit pour montrer comment inviter l'utilisateur à entrer une information et comment est précisé le nom de la variable qui va recevoir cette donnée.
NB. Beaucoup de choses pourraient encore être dites sur cette instruction INPUT mais tous ces détails ne nous intéressent pas. Concentrons-nous sur la logique de programmation plutôt que sur les spécificités du QBasic.

Exercices

Ex 5c Ecrivez et testez un petit programme qui vous demande votre nom puis votre prénom. Il vous salue ensuite comme suit :

```
Quel est ton Nom ?  Lambert
Quel est ton prénom ? Gérard
Bonjour Gérard Lambert
```

Ex 5d Après vous avoir demandé votre nom, le programme vous demande votre âge. Il conclut en vous indiquant le nombre d'années qu'il vous reste avant de profiter de votre retraite.

```
Quel est ton Nom ?  Tartempion
Quel est ton âge ? 20
Dans ce cas il te reste 45 ans avant la retraite.
```

Ex 5e Variante pas bien plus compliquée du même exercice (faites la tout de même)

```
Quel est ton Nom ?  Tartempion
Bonjour Tartempion, quel est ton âge ? 20
Dans ce cas il te reste 45 ans avant la retraite.
Courage !
```

6. Structures de programmation

6.1. Programmation structurée

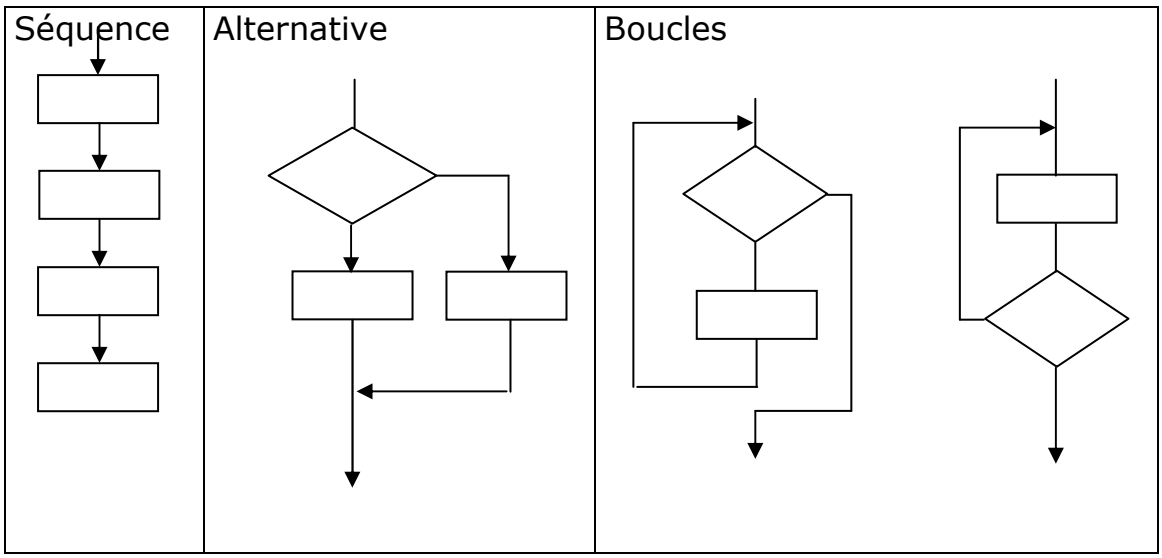
Un programme est un ensemble d'instructions exécutées par l'ordinateur. Les langages tels que le BASIC possèdent depuis longtemps des instructions telles que le GOTO qui permette de faire des sauts d'un endroit à l'autre dans un programme. Grâce à ces sauts combinés avec l'instruction IF on sait demander au programme de changer sa marche à suivre en fonction des valeurs que prennent les variables. On a bidouillé ainsi pendant de nombreuses années en sautant comme bon semblait d'un point à l'autre du programme. C'était le temps de la programmation « spaghetti ».
Les programmes écrits de la sorte pour peu qu'ils ne soient pas abandonnés en cours de route finissent parfois par marcher mais sont difficiles à modifier et à faire évoluer.
Depuis des théoriciens ont démontré que trois structures suffisent pour écrire n'importe quel programme. On est entré dans l'aire de la programmation structurée. Les instructions de type GOTO sont depuis considérées comme des abominations par tout programmeur qui se respecte et les langages structurés tels que le Pascal et le C sont apparus.
Un programme se doit d'être un ensemble structuré d'instructions.

6.2. Les trois structures de base

La programmation structurée repose sur les trois structures suivantes :

La séquence :	un simple enchaînement d'instructions
L'alternative :	choix entre deux séquences
Les boucles :	répétitions d'une même séquence

Voici comment ces structures de programme sont représentées schématiquement dans un organigramme :



6.3. L'alternative

IF . . . THEN . . . ENDIF

Le QBasic comme la plupart des langages de programmation possède une instruction IF pour changer le cours de l'exécution d'un programme en fonction du résultat d'un test de comparaison. La syntaxe varie d'un langage à l'autre, voici comment il conviendrait d'écrire ces structures de décision en QBasic :

Syntaxe	Exemples

IF *condition* THEN
  Séquence d'instructions
ENDIF
	IF n = 0 THEN
	  `Print "Valeur nulle"`
	ENDIF

IF *condition* THEN
  Séquence d'instructions A
ELSE
  Séquence d'instructions B
ENDIF
	IF Age >= 18 THEN
	  PRINT `"Majeur"`
	ELSE
	  PRINT `"Mineur"`
	ENDIF

IF *condition* THEN
  Séquence d'instructions A
ELSEIF condition 2
  Séquence d'instructions B
ELSE
  Séquence d'instructions C
ENDIF
	IF Age < 25 THEN
	  PRINT `"Tarif Jeunes"`
	ELSEIF Age >= 60
	  PRINT `"Tarif Seniors"`
	ELSE
	  PRINT `"Tarif plein"`
	ENDIF

Les conditions sont exprimées au moyen d'opérateurs relationnels

>	plus grand
>=	plus grand ou égal
<	plus petit
<=	plus petit ou égal
=	égal
<>	différent

Les opérateurs logiques tels que AND, OR et XOR rendent possibles les combinaisons de plusieurs conditions.

IF (Sexe = `"M") AND (Age >= 18)THEN` …

SELECT CASE . . .

L'instruction SELECT CASE convient mieux dès qu'il y a plus de trois conditions à tester

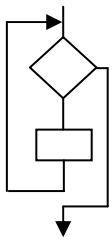
SELECT CASE *expression*
  CASE *liste_de_valeurs_1*
    Séquence d'instruction 1
  CASE *liste_de_valeurs_1*
    Séquence d'instruction 2
  …
  CASE ELSE
    Séquence d'instruction N
END CASE

Les listes de valeurs peuvent se présenter comme suit :

2	= une valeur
12, 15, 21	= plusieurs valeurs
10 TO 20	= une plage de valeurs
IS > 12	= au-delà d'une valeur

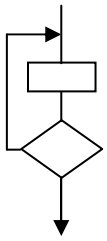
6.4. Les boucles

DO . . . LOOP



Le nombre de répétitions est contrôlé par un test relationnel qui est soit placé au-dessus de la boucle soit en fin de boucle.
Dans le premier cas, puisque le test est fait en premier lieu, la séquence d'instructions pourrait très bien ne jamais être exécutée.
Par contre, la séquence qui précède est toujours exécutée au moins une fois quand le test est mis en fin de boucle.
Les conditions de répétitions sont introduites par WHILE (= tant que …)
ou avec une condition inverse par UNTIL (= jusqu'à ce que …)
Le mot LOOP a ici le sens de « boucler » / « répéter »

DO WHILE *condition* | **LOOP**	**DO UNTIL** *condition* | **LOOP**
DO | **LOOP WHILE** *condition*	**DO** | **LOOP UNTIL** *condition*



FOR . . . NEXT

La boucle FOR . . . NEXT offre une notation plus concise pour les répétitions gérées par un compteur.

```
CLS
FOR i = 0 to 6        'Affiche tous les nombres de 1 à 6
PRINT  i
NEXT i
```

L'instruction NEXT au bas de la boucle incrémente la valeur du compteur. La valeur de celui-ci est ensuite testée pour déterminer si la boucle doit être répétée.
L'incrémentation (le comptage) est par défaut d'une unité à chaque passage à moins qu'un pas différent ne soit précisé en ajoutant une instruction STEP:

```
CLS
FOR i = 0 to 6 STEP 2 'Affiche les nombres pairs de 0 à 6
PRINT  i
NEXT i
```

Plantage d'une boucle

Une cause de « plantage » fréquent est dû au problème de terminaison d'une boucle. Une condition d'arrêt qui n'est jamais vérifiée fera boucler le programme indéfiniment. La seule solution pour en sortir alors sera de presser les touche Ctrl+Break

Exit

L'instruction `EXIT` permet de commander la sortie prématurée d'une boucle quand on s'aperçoit que les répétitions ultérieures deviennent inutiles
Suivant le type de boucle on écrira `EXIT DO` ou `EXIT FOR`

Exercices

Ex 6a Le prix d'entrée d'une attraction est fixé comme suit :

	En semaine	Le week-end
Enfants de moins de 10 ans	Gratuit	Gratuit
Moins de 18 ans et plus de 60 ans	2 Euros	4 Euros
Prix plein	3 Euros	6 Euros

Ecrivez un petit programme qui indique le prix du billet en fonction de l'âge du visiteur et du moment du jour de la visite (demi-tarif en semaine)

Ex 6b Ecrivez un petit programme qui demande un nombre entier puis qui affiche les 10 premiers multiples de ce nombre.

Exemple :

```
Donnez un nombre entier : 12
1 x 12 = 12
2 x 12 = 24
. . .
```

Ex 6c Ecrivez un petit programme qui boucle en vous demandant si vous voulez continuer. Il vous réprimande si vous ne répondez pas par oui ou par non !

```
Ce programme s'achèvera dès que vous le demanderez
Voulez-vous continuer ? oui
Voulez-vous continuer ? bof
Veuillez répondre par oui ou par non !
Voulez-vous continuer ? non
```

Ex 6d La fonction Timer rend le nombre de secondes écoulées depuis minuit en indiquant jusqu'à 8 chiffres derrière la virgule. Si on demande l'affichage de cette valeur dans une boucle à l'aide de la commande PRINT on s'aperçoit que la valeur du timer ne change pourtant que 18 fois par seconde. La boucle étant courte, le programme s'exécute rapidement et l'affichage répète de nombreuses fois la même valeur. Faites le test.

Modifiez ensuite le programme pour qu'il boucle en affichant les valeurs successives de la fonction Timer sans jamais répéter deux fois de suite la même valeur

Ex 6e La fonction RND rend un nombre aléatoire compris entre 0 et 1. Ecrivez un programme qui affiche un nombre aléatoire compris entre 0 et 100.

Ex 6f Cette fois au lieu d'afficher ce nombre, le programme vous demande de le deviner.

```
J'ai choisi un nombre entre 1 et 100
Quel est ce nombre? 36
Trop petit
Quel est ce nombre? 75
Trop grand
. . .
. . .
Quel est ce nombre? 58
Vous avez trouvé !
```

7. Les tableaux et les types structurés

7.1. Tableaux

Nous avons vu ce qu'est une variable. Elle possède un nom et un type. Elle contient une valeur. Un tableau possède lui aussi un nom et un type mais il est destiné à contenir plusieurs valeurs. Chaque valeur est désignée par le nom du tableau suivi d'un ou de plusieurs indices mis entre parenthèses.

La ligne suivante déclare un tableau destiné à contenir les noms d'une vingtaine d'étudiants. Ce tableau comportera en réalité 21 éléments, les indices pouvant aller de 0 à 20.

```
DIM Etudiants(20) AS STRING
```

Les tableaux à plusieurs dimensions sont aussi possibles. Voici par exemple comment on déclare un tableau pour y noter 10 côtes pour 20 étudiants.

```
DIM Cotes( 20, 10) AS INTEGER
```

Chaque élément d'un tableau est du même type que le tableau. Pour enregistrer des informations de types différents, il faudrait recourir à des tableaux parallèles. Les informations correspondantes y seraient repérées par un même indice mais réparties dans les tableaux de différents types.

7.2. Les types structurés

Le Basic permet de définir des structures de données composées de variables de types élémentaires.
Exemple :

```
TYPE Inscription
  Nom    AS STRING
  Prenom AS STRING
  Code   AS INTEGER
END TYPE
```

Ici « Inscription » n'est pas une variable mais un nouveau type de donnée que nous venons de définir pour regrouper une famille d'informations. Il peut servir à rassembler les informations sur un étudiant :

```
DIM Etudiant AS Inscription

Etudiant.Nom    = "Lambert"
Etudiant.Prenom = "Gérard"
```

Nous pouvons même créer un tableau dont le type est notre type personnalisé :

```
DIM Classe(25) AS Inscription

Classe(1).Nom    = "Lambert"
Classe(1).Prenom = "Gérard"
```

Exercices

Ex 7a Ecrire un programme qui vous demande un chiffre de 0 à 9 et vous l'affiche en toutes lettres.

8. Procédures

Ce que nous appelons les « procédures » en QBasic ou en VisualBasic s'appellent dans d'autres contextes des « sous-programmes » ou encore des « sous-routines ». Ce sont des morceaux de programme écrits séparément pour subdiviser les développements en sous-ensembles plus simples. Plus un programme est subdivisé et plus il sera facile à appréhender et à faire évoluer et ceci pour deux raisons :

- Les procédures peuvent être écrites et testées indépendamment du reste du programme dont la complexité devient sans cela plus que proportionnelle à la taille.
- Les procédures sont écrites une fois pour toute puis peuvent êtres réutilisées dans d'autres programmes.

On distingue en QBasic deux types de procédures : les procédures « SUB » et les procédures « FUNCTION » La différence entre les deux est minime : une fonction rend une valeur, une procédure n'en renvoie pas.

8.1. La procédure SUB

Lorsqu'une même suite d'instructions se retrouve en plusieurs endroits du programme, on s'épargne le fait de devoir la réécrire en les plaçant dans une procédure SUB à laquelle on donne un nom. Il suffira d'appeler cette procédure chaque fois que l'exécutions de ces instructions seront nécessaire.
Exemple : Imaginons une procédure SUB appelée Bienvenue qui aurait comme paramètre un nom et qui afficherait un message de bienvenue à l'attention de ce nom. Cette procédure même si elle n'est appelée qu'à un seul endroit du programme aurait le mérite de le découper en entités logiques qui en simplifieraient le programme.
Voici comment cette procédure serait définie et comment ailleurs dans le programme elle serait appelée :

```
SUB Bienvenue( Nom As String)
PRINT ∀Soit le bienvenu∀ + Nom
END SUB
```

L'appel à cette procédure se fait alors avec l'instruction CALL : (*To call* = appeler)

```
CALL Bienvenue(∀Léon∀)
```

Ou encore plus simplement comme ceci (sans l'instruction CALL ni parenthèses)

```
Bienvenue ∀Léon∀
```

Cette dernière forme est plus simple. C'est un peu comme si on avait ajouté au langage une instruction `Bienvenue` qui s'utilise presque comme l'instruction PRINT.

8.2. La procédure FUNCTION

Une fonction est une procédure qui a un rôle supplémentaire : nous retourner une valeur. Imaginer par exemple une fonction que nous appellerons `Maxi` dont le but serait de nous retourner la valeur la plus grande de deux nombres.
Commençons par définir ce que fait la fonction :

```
FUNCTION Maxi( N1 AS LONG, N2 AS LONG) AS LONG
IF N1 > N2 THEN
      Maxi = N1
ELSE
```

```
        Maxi = N2
    END FUNCTION
```

Voici comment elle s'utilise à présent :

```
    MeilleurScore AS LONG
    MeilleurScore = Maxi( Score1, Score2)
```

8.3. Fonctions internes

Un certain nombre de fonctions ont été prévues par les concepteurs du langage BASIC pour faire des tâches qui ont toutes les chances de vous être utiles.

Remarque Certains noms de fonctions son terminés par le caractère spécial '$'. Ce caractère avait une fonction spéciale en QBASIC dont nous n'avons pas encore parlé car il s'agit d'une particularité trop spécifique au Qbasic. Le suffixe $ sert à distinguer les variables de type STRING sans devoir les déclarer explicitement.
Les fonctions dont le nom se termine par le caractère $ sont donc des fonctions qui rendent une chaîne de caractères.
Ces mêmes fonctions existent aussi en Visual Basic mais sans ce $.

Fonctions mathématiques

`ABS(` *n*`)`	valeur absolue de n ABS(-5.7) → 5.7
`CINT(` *n*`)`	conversion du nombre n en entier (arrondi) CINT(-5.7) → -6
`FIX(` *n*`)`	Partie entière tronquée de n FIX(-5.7) → -5
`INT(` *n*`)`	partie entière de n INT(-5.7) → -6
`CLNG(` *n*`)`	conversion du nombre n en LONG
`CSNG(` *n*`)`	conversion du nombre n en SINGLE
`CDBL(` *n*`)`	conversion du nombre n en DOUBLE
`RND`	donne un nombre aléatoire compris entre 0 et 1
`HEX$(` *n*`)`	chaîne de caractère représentant n en hexadécimal

Fonctions pour manipuler les chaînes de caractères

`LEN(` *chaîne*`)`	nombre de caractères de cette *chaîne*
`LEFT$(` *chaîne, n*`)`	*n* premiers caractères de la *chaîne*
`RIGHT$(` *chaîne, n*`)`	*n* derniers caractères de la *chaîne*
`MID$(` *chaîne, p, n*`)`	*n* caractères à partir de la position *p*
`LCASE$(` *chaîne*`)`	met la *chaîne* en minuscules (*lower case*)

`UCASE$( chaîne)`	met la *chaîne* en majuscules (*upper case*)
`ASC( chaîne)`	code ASCII du premier caractère de la *chaîne*
`CHR$( c)`	caractère qui a *c* pour code ASCII
`STRING$( n, c)`	chaîne de *n* caractères ayant *c* pour code ASCII
`STR$( n)`	chaîne de caractères représentant la valeur numérique *n*
`VAL( chaîne)`	valeur numérique si la *chaîne* ne contient que des chiffres
`LTRIM$( chaîne)`	la chaîne débarrassée d'éventuels espaces à gauche
`RTRIM$( chaîne)`	la chaîne débarrassée d'éventuels espaces à droite
`INSTR(début, chaîne, cc)`	recherche la suite de caractère *cc* depuis la position *début* de la *chaîne*

9. Index