Chapitre 11

Quelques notions de langage BASIC

Nous avons donc choisi de programmer un microproceseur, le 8088, en utilisant un système informatique complet que constitue un micro-ordinateur PC muni du système d'exploitation MS DOS.

On commence à programmer de nos jours en un langage dit de haut niveau et non tout de go en langage machine. Ceci était vrai même sur les premiers micro-ordinateurs, tout au moins ceux qui ont connu un succès commercial. Le langage utilisé était le **BASIC** (pour *Beginner's All-purpose Symbolic Instruction Code*), conçu au *Darmouth College* en 1965 pour initier les étudiants à la programmation.

Nous allons donc commencer par étudier quelques notions de BASIC, à savoir ce qui est indispensable pour notre propos, programmer en langage machine. Nous l'illustrerons par l'interpréteur BASIC appelé *QBasic* livré avec MS DOS 5.0, mais bien entendu tout autre interpréteur convient pour ce chapitre.

11.1 Le BASIC comme petite calculette

11.1.1 Mise en œuvre du QBasic

Il suffit de MS DOS et de l'application Microsoft QBasic.

Écrire « qbasic » à l'apparition du prompteur MS DOS puis appuyer sur la touche retour. On se retrouve dans un environnement intégré (EDI pour *Environment Application Interface*) tel qu'il apparaît à la figure 11.1.

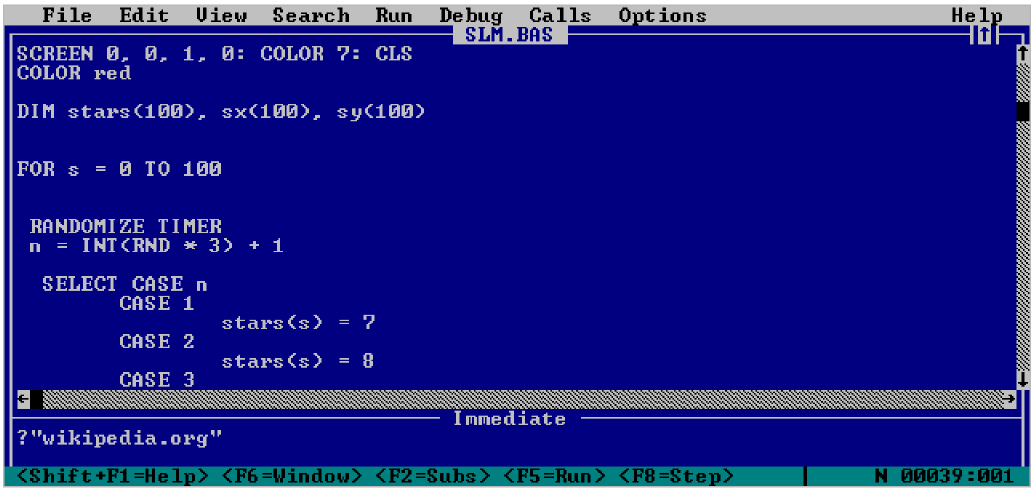

Fig. 11.1 – Éditeur intégré du Qbasic

En haut de la fenêtre se trouve la barre de menu. À l'arrière plan se trouvent deux fenêtres (nous y reviendrons). Tout en bas se trouve une barre avec les raccourcis des commandes les plus utilisées ainsi que les coordonnées du curseur. Enfin, une fenêtre au centre de l'écran vous souhaite la bienvenue. L'option « guide » vous amène dans l'aide en ligne de QBasic.

Appuyez sur la touche ESC du clavier. Vous remarquez qu'il y a deux fenêtres : une grande et une petite. La fenêtre « Immédiate » (la petite), ne permet d'exécuter qu'une instruction à la fois. La fenêtre « Programme » (la grande), est celle qui permet d'écrire de vrais programmmes.

Dans les premiers temps, nous nous servirons seulement de la fenêtre immédiate. Si vous avez déjà utilisé des versions antérieures de Basic, vous verrez quelques différences en QBasic : la numérotation des lignes (10, 20, 30, etc) n'est plus nécessaire, bien que cette opération soit encore supportée ; la fenêtre « immédiate » correspond à l'ancien mode immédiat, c'est-à-dire qu'une commande tapée dans cette fenêtre est immédiatement exécutée après avoir appuyé sur la toucke « Entrée ». La fenêtre programme représente le mode programme et permet de rédiger un programme et de le lancer ultérieurement.

11.1.2 Premières opérations

Commençons par utiliser le BASIC en tant que calculette.

Notre première opération.- Écrivons :

```
print 2 + 2
```

dans la fenêtre « immédiate » puis appuyons sur la touche « retour ». On obtient le résultat voulu sur l'cran habituel de MS-DOS :

```
C:\>qbasic
 4

Appuyez sur une touche pour continuer
```

Si on appuie sur une touche, on revient l'diteur intgr du QBasic.

<u>Affichage des entiers</u>.- On pourra tester d'autres additions ou simplement l'affichage d'un entier. On remarquera que l'on peut utiliser un espace tous les trois chiffres (ou lorsque l'on veut) mais pas la virgule anglo-saxonne :

```
print 2 340 + 25
```

qui donne 2 365.

<u>Dépassement de capacité</u>.- Lorsque l'entier devient trop grand, une valeur approchée est affichée comme nombre décimal sous forme scientifique :

```
print 12345678901234567890
```

donne :

```
1.2345678901234567D+19
```

ce qui signifie $1.2345678901234567 \times 10^{19}$, soit 1 234 567 890 123 4567 000.

<u>Autres opérations sur les entiers</u>.- On peut de même faire effectuer des multiplications :

```
print 2*2
```

ce qui donne bien :

```
4
```

en remarquant que le signe de multiplication est '*' (qui se trouve bien sur le clavier) et non '$\times$' (qui ne se trouve pas sur le clavier) ou la lettre 'x'.

Il n'y a pas grand chose à dire sur la soustraction :

```
print 5 - 6
-2
```

La division correspond à une division entière exacte :

```
print 5/6
0
```

complétée par une opération *modulo* qui donne le reste dans la division euclidienne :

```
print 5 mod 6
5
```

L'exponentiation est également implémentée, avec le signe '^' (qui se trouve sur le clavier) :

```
print 2 ^ 3
8
```

<u>Calcul complexe</u>.- On peut aussi combiner ces opérations élémentaires pour évaluer une expression arithmétique plus complexe, en utilisant au besoin des couples de parenthèses :

```
print (1+2)*5
15
```

11.1.3 Le BASIC orienté texte

Un langage de programmation tel que le BASIC permet d'effectuer des calculs, comme nous venons de le voir, mais également de manipuler du texte, appelé **chaîne de caractères** en informatique (*string* en anglais).

11.1.3.1 Affichage d'un texte

Pour afficher une chaîne de caractères, il suffit de faire suivre la commande `print` de la chaîne de caractères que l'on veut voir afficher écrite, entre guillemets verticaux (signe qui se trouve sur le clavier) :

```
print "Bonjour"
Bonjour
```

En fait toutes les chaînes de caractères ne sont pas permises (elles ne doivent pas contenir de guillemets verticaux entre autre) mais le détail ne nous intéresse pas pour l'instant.

11.1.3.2 Mise en forme

Effacer l'écran.- L'instruction CLS (pour l'anglais *CLear Screen*) efface l'écran.

Affichage sur la même ligne.- Écrivez dans la fenêtre « programme » :

```
CLS
PRINT "7 + 9 = "
PRINT 7 + 9
```

puis apuyez sur la touche « F5 ». Nous basculons vers l'écran de MS-DOS et on voit écrit :

```
7 + 9 =
16
Appuyez sur une touche pour continuer
```

Que s'est-il passé ? QBasic a tout d'abord effacé l'écran (CLS). Puis a écrit « 7 + 9 = », puisque ce texte était placé entre guillemets verticaux. Enfin il a effectué l'opération 7 + 9 et a affiché le résultat, c'est-à-dire 16.

Cette présentation ne nous satisfait pas nécessairement car le résultat 16 se trouve à la ligne et qu'on aimerait bien qu'il se trouve à la suite. On peut, pour cela, utiliser un point virgule. Écrivez :

```
CLS
PRINT "7 + 9 = ";
PRINT 7 + 9
```

Cette fois, en faisant exécuter le programme, on lit :

```
7 + 9 = 16
Appuyez sur une touche pour continuer
```

Le point virgule force l'affichage sur la même ligne.

Dès lors, la loi du moindre effort l'emporte, et l'on peut arriver au même résultat en écrivant un PRINT de moins :

```
CLS
PRINT "7 + 9 ="; 7 + 9
```

Le résultat sera le même, c'est-à-dire :

```
7 + 9 = 16
Appuyez sur une touche pour continuer
```

Tabulation.- Utilisons une virgule à la place d'un point virgule :

```
CLS
PRINT "7 + 9 = ", 7 + 9
```

Nous obtenons comme résultat :

```
7 + 9 =        16
```

c'est-à-dire qu'une tabulation a été posée entre « 7 + 9 = » et 16. La valeur d'une tabulation par défaut est de 8 caractères.

Une abréviation.- Au lieu d'écrire PRINT, qui représente quand même 5 caractères, on peut remplacer ce mot clé par un point d'interrogation « ? » :

```
? "QBasic est un langage surprenant."
```

afichera la phrase à l'écran.

Plusieurs instructions sur la même ligne.- Le symbole deux points « : » permet d'écrire plusieurs instructions sur la même ligne. Au lieu de :

```
CLS
PRINT "7 + 9 = ";
PRINT 7 + 9
```

on peut écrire :

```
CLS : PRINT "7 + 9 = "; : PRINT 7 + 9
```

Le résultat sera identique :

```
7 + 9 = 16
Appuyez sur une touche pour continuer
```

11.2 Variables – types – ordre de lecture

11.2.1 Variables

Qu'est ce qu'une variable ? Pour simplifier, disons qu'une variable est un emplacement que l'on réserve dans la mémoire vive de l'ordinateur pour y stocker des données.

Une variable a un nom, qui est une chaîne de caractères de moins de 40 caractères, qui ne comporte aucun caractère accentué, qui commence par une lettre et qui n'est pas un mot réservé. Le caractère $ ne peut être utilisé que dans des cas précis que nous allons étudier.

11.2.1.1 Variables numériques

Une variable numérique contient un nombre.

Initialisation d'une variable.- Pour initialiser à 5 la variable de nom « Chiffre », il faut procéder ainsi :

```
LET Chiffre = 5
```

C'est en fait une survivance des anciennes versions de BASIC. On peut en fait écrire plus simplement :

```
Chiffre = 5
```

On peut vérifier qu'on a bien initialisé la variable :

```
CLS
Chiffre = 5
PRINT Chiffre
```

dont le résultat sera :

```
5
Appuyez sur une touche pour continuer
```

Utilisation des variables.- On peut bien entendu effectuer des calculs avec les variables :

```
CLS
Chiffre = 5
PRINT Chiffre * NouvelleVariable
```

dont le résultat sera :

```
0
Appuyez sur une touche pour continuer
```

puisque la variable `NouvelleVariable` n'a pas été initialisée et qui prend alors la valeur nulle.

Par contre :

```
CLS
Chiffre = 5
NouvelleVariable = 3
Resultat = Chiffre * NouvelleVariable + 1
PRINT Resultat
```

donnera :

```
16
Appuyez sur une touche pour continuer
```

11.2.1.2 Variables chaîne de caractères

Dénomination des variables.- Si on écrit :

```
Langage = "QBasic est un langage surprenant"
```

QBasic signale une erreur « Types incompatibles ». Il existe en effet deux types de variables : les variables numériques et les variables chaîne de caractères. QBasic doit pouvoir faire la différence entre ces types. Dans la ligne de code ci-dessus, on a essayé d'affecter une chaîne de caractères à une variable de type numérique, d'où l'erreur « Types incompatibles ».

Le nom d'une variable chaîne de caractères doit se terminer par le caractère $. Le programme :

```
CLS
Langage$ = "QBasic est un langage surprenant"
PRINT Langage$
```

affiche bien :

```
QBasic est un langage surprenant
Appuyez sur une touche pour continuer
```

Concaténation.- La concaténation, c'est-à-dire l'ajout d'une chaîne de caractères à la suite d'une autre, s'effectue en BASIC en utilisant le symbole d'addition :

```
CLS
Partie1$ = "Un exemple de "
Partie2$ = "concatenation"
Phrase$ = Partie1$ + Partie2$
PRINT Phrase$
```

donnera :

```
Un exemple de concatenation
Appuyez sur une touche pour continuer
```

11.2.2 Type des variables

Nous avons vu que les variables sont au moins de deux types : les variables numériques et les variables chaîne de caractères. En fait il y a cinq types de variables, le type étant spécifié par le dernier caractère du nom de la variable (comme '$' dans le cas d'une variable chaîne de caractères) :

Type	Symbole	Contenu	Taille
INTEGER	%	Entier compris entre -32 768 et 32 767	2 octets
LONG	&	Entier compris entre -2 147 483 648 et 2 147 483 647	4 octets
SINGLE	!	Nombre décimal à virgule flottante avec 7 chiffres significatifs	4 octets
DOUBLE	#	Nombre décimal à virgule flottante avec 15 chiffres significatifs	4 octets 8 octets
STRING	$	Chaîne de caractères jusqu'à 32 767 caractères	X caractères

Par défaut, si on ne précise pas le type, une variable est du type SINGLE.

11.2.3 Commentaires

Plus on avance en programmation, plus les programmes deviennent longs, plus ils deviennent complexes et moins on s'y retrouve. C'est pourquoi il existe les commentaires. Dans un commentaire, on peut crire ce que lon veut.

Il existe deux façons d'écrire des commentaires :

– en utilisant REM (pour l'anglais *REMark* :

```
REM Ceci est un commentaire, et je peux ecrire ce que je veux
```

– REM est présent depuis les premières versions de BASIC et a été conservé mais n'est pratiquement plus utilisé. On lui préfère l'apostrophe, ce qui permet d'écrire des commentaires en fin de ligne :

```
CLS                 'Cette instruction efface l'ecran
PRINT "7 + 9 = ";   'Celle-ci affiche 7 + 9 =
PRINT 7 + 9         'Quand a celle-la, elle effectue
                    'l'operation 7 + 9 et affiche le resultat
```

11.2.4 Constantes

Une **constante**, comme son nom l'indique et l'analogue d'une variable mais dont la valeur ne varie pas, une fois qu'elle a été initialisée.

Exemple.- Considérons un programme qui ait besoin d'utiliser à plusieurs reprises le nombre π. Au lieu d'écrire à chaque fois une approximation de π telle que 3.14159265458, on peut définir une constante PI ne variera pas et que l'on pourra substituer. Pour cela, on écrit :

```
CONST PI = 3.14159265458
```

La déclaration d'une variable ou d'une constante doit toujours se faire avant sa première référence. Pour calculer l'aire d'un disque, on doit donc définir PI avant son premier appel :

```
CONST PI = 3.14159265458  Dfinition de PI
CLS
PRINT "L'aire d'un disque de rayon de 5 est "; PI * 5 ^ 2
```

Le résultat sera :

```
L'aire d'un disque de rayon 5 est 78.5398163645
```

11.2.5 Lecture des données au clavier

11.2.5.1 Lecture de données numériques

Nous venons de voir comment calculer l'aire d'un disque en BASIC. Mais comment faire pour calculer l'aire de plusieurs disques sans avoir à modifier le programme ? On fait appel à l'instruction de lecture au clavier INPUT. La syntaxe de cette instruction pour lire un nombre est :

```
INPUT variable
```

Exemple.- Une amélioration du programme précédent est donc :

```
CONST PI = 3.14159265458
CLS
PRINT "Entrez le rayon du disque "
INPUT Rayon 'On stocke la valeur du rayon dans la variable Rayon
PRINT "L'aire d'un disque de rayon " ; Rayon; " est " ; PI * Rayon ^ 2
'On effectue le calcul et on affiche le resultat
```

Le résultat doit ressembler à :

```
Entrez le rayon du disque
? _
```

Un point d'interrogation est affiché. L'ordinateur attend une réponse. Entrez n'importe quel nombre, par exemple 5. Le nombre 5 est alors stocké dans la variable `Rayon`. On effectue ensuite le calcul et on affiche le résultat.

11.2.5.2 Lecture des données chaînes de caractères

Exemple.- Rappelons-nous que dans ce cas la variable doit marquer explicitement le fait qu'il s'agit d'une chaîne de caractères :

```
CLS
PRINT "Entrez votre nom "
INPUT Nom$              'Ne pas oublier le $
PRINT "Bonjour " ; Nom$
```

Le programme va vous demander votre nom puis l'afficher. Les règles de ponctuation s'appliquent avec l'instruction INPUT. On peut donc un point virgule à la fin de la seconde ligne :

```
CLS
PRINT "Entrez votre nom ";
INPUT Nom$
PRINT "Bonjour " ; Nom$
```

Le point d'interrogation se trouve maintenant sur la même ligne. Avec une virgule ça aurait été pareil, sauf qu'il y aurait eu une tabulation.

11.2.5.3 Remplacement de PRINT par INPUT

Afin de simplifier l'ensemble, on peut afficher un texte avec l'instruction INPUT, ce qui n'était pas possible dans les versions antérieures de BASIC :

```
CLS
INPUT "Entrez votre nom "; Nom$ 'On supprime l'instruction PRINT
PRINT "Bonjour " ; Nom$
```

Ici, le rôle du point virgule est différent. Avec un point virgule, voici l'affichage :

```
Entrez votre nom?_
```

Avec une virgule, on aurait eu :

```
Entrez votre nom_
```

Le point d'interrogation ne s'affiche plus avec la virgule, ce qui dans certains cas améliore l'esthétique.

11.2.6 Lectures des données placées après le programme

Introduction.- Il est rare que l'on conçoive un programme pour un seul jeu de données. En général un programme attend des données et agit en fonction de celles-ci. Le plus simple, pour transmettre ces données est le mode interactif avec questions/réponses : nous avons vu l'instruction INPUT du BASIC qui permet ce mode. Une autre façon de faire est de placer les données, surtout si elles sont en nombre important, dans un fichier.

Au tout début de l'informatique le mode interactif et les fichiers n'existaient pas. On plaçait le programme (avec un jeu d'interrupteurs puis des cartes perforées) puis les données (de la même façon) dans les registres (d'entrée et d'instructions dans une première étape, avec une seule sorte de registres dans une seconde étape). Deux sessions se distinguaient par des jeux de cartes de données différentes mais le même jeu de cartes de programme.

BASIC reflète encore cette façon de faire.

L'instruction DATA.- On peut terminer un programme BASIC par un certain nombres de lignes commençant par le mot clé DATA (*données* en anglais) suivi d'un certain nombre de constantes séparées par des virgules.

L'instruction READ.- L'instruction BASIC :

```
READ var
```

où `var` est une variable, permet d'accéder aux données placées de la façon ci-dessus. Le premier appel récupère la première donnée, le deuxième la deuxième et ainsi de suite.

Exemple.- L'exemple suivant permet d'additionner deux entiers placés en données et d'en afficher le résultat :

```
READ A
READ B
C = A + B
PRINT A;" + ";B;" = ";C
DATA 11,23
```

qui affiche bien `11 + 23 = 34`.

11.3 Les structures de contrôle

Nous avons vu tout ce qu'il faut connaître sur les **instructions primitives** : les opérations élémentaires que l'on peut effectuer sur les entités (addition, soustraction, multiplication, division, exponentiation sur les entiers), comment entrer un entier (et même une chaîne de caractères, bien que cela nous intéressera moins) et comment afficher le résultat (entier pour ce qui nous intéresse, éventuellement précédé d'une chaîne de caractères pour bien expliquer de quoi il ressort).

Nous avons vu également que l'on peut combiner les opérations élémentaires pour obtenir une **expression arithmétique** en utilisant si besoin est des couples de parenthèses.

Nous avons maintenant besoin des **structures de contrôle** pour combiner les instructions élémentaires afin de réaliser des calculs plus complexes. Nous avons déjà vu le séquencement. Voyons en d'autres maintenant.

11.3.1 Le séquencement

Nous avons vu comment faire exécuter une instruction simple en BASIC. On peut désirer faire exécuter deux ou plusieurs telles instructions les unes après les autres. On parle alors de **séquencement**. Nous avons vu galement comment faire : il suffit d'écrire une instruction par ligne ou de les séparer par des points-virgules.

11.3.2 La rupture de séquence inconditionnelle

La **rupture de séquence inconditionnelle**, ou **saut inconditionnel**, spécifie que l'on veut aller à une ligne, ce qui s'exprime par l'instruction `GOTO` suivie du numéro de ligne désiré.

Imaginez que vous vouliez écrire un grand nombre de fois 'Patrick' sur l'écran de votre ordinateur. On pourrait utiliser le séquencement et écrire un grand nombre de fois `PRINT "Patrick"`. Mais on peut faire mieux avec une boucle.

Utilisation de RUN.- Nous pouvons écrire le court programme suivant :

```
PRINT "Patrick"
RUN
```

Pour arrêter le défilement il suffit d'appuyez sur CONTROL + PAUSE. RUN signifie « courir » en anglais mais, dans ce contexte, RUN donne l'ordre d'exécuter le programme depuis le tout début.

Utilisation de GOTO.- Pour utiliser l'instruction GOTO (« aller à » en anglais), il faut définir une tiquette. Par exemple, pour une étiquette appelée DEBUT, on écrit DEBUT suivi d'un double points ' :'. Lorque l'ordinateur arrivera à l'instruction GOTO suivi de cette étiquette, il ira à l'instruction précédée de l'étiquette spécifiée :

```
DEBUT:
PRINT "Patrick"
GOTO DEBUT
```

Là aussi, pour arrêter, appuyez sur CONTROL + PAUSE.

11.3.3 L'itération contrôlée

Premier exemple.- Nous voudrions un programme qui permette de saisir un entier naturel puis de calculer la somme des carrés des entiers depuis un jusqu'à cet entier.

```
INPUT "ENTRER UN ENTIER : ";N
S = 0
FOR I = 1 TO N
  S = S + I*I
NEXT I
PRINT "SOMME = ";S
```

Ce qui donne :

```
ENTRER UN ENTIER : 2
SOMME = 5
```

La troisisème ligne du programme introduit une nouvelle instruction, appelée **itération contrôlée**. On utilise pour cela une variable entière non déjà utilisée par ailleurs, ici `I`, appelée **variable de contrôle**. Les lignes suivantes, jusqu'à ce qu'on rencontre `NEXT` suivi de cette variable, seront exécutées plusieurs fois : plus exactement une fois pour chaque valeur de `I` comprise dans l'intervalle spécifié, ici depuis 1 jusqu'à `N`.

<u>Second exemple</u>.- On peut également effectuer un décompte. Observez le programme suivant qui réalise un compte à rebours de 10 000 à 0 :

```
FOR I = 10000 TO 0 STEP -1 'Compte a rebours de 10 000 a 0 de 1 en 1
  PRINT I
NEXT I
```

On a fait appel à un nouveau mot-clef : STEP. En anglais, STEP signifie PAS. En effet, dans une boucle FOR, le PAS par défaut est de 1. Pour effectuer un compte à rebours, il nous faut aller de -1 en -1.

<u>Troisième exemple</u>.- Attention ! On pourrait penser qu'une boucle FOR s'exécute, quoiqu'il arrive, le nombre de fois spécifié. Essayez donc ce petit exemple :

```
CLS
FOR i = 1 TO 10000
  i = 10000
  PRINT "Bonjour !"
NEXT i
```

On pourrait penser que le programme va écrire 10 000 fois 'Bonjour !'. En fait l'itération ne s'arrête pas au bout de 10 000 fois, mais lorsque la variable de contrôle `i` est égale à 10 000. Comme on a affecté la valeur 10 000 à `i` dans le corps de la boucle, la boucle ne s'exécute qu'une fois !

11.3.4 Les itérations non contrôlées

11.3.4.1 La boucle FAIRE

<u>Syntaxe</u>.- La boucle `DO instruction LOOP` effectue, une fois de plus, indéfiniment l'instruction `instruction` :

```
DO                'Tete de la boucle
  PRINT "Patrick" 'Instruction(s)
LOOP              'Queue de la boucle
```

Vous pouvez arrêter le défilement avec CONTROL + PAUSE.

Remarquez que, pour le corps de la boucle, on a observé un retrait de deux caractères par rapport aux autres instructions. On parle d'**indentation**. Ce n'est pas syntaxiquement obligatoire mais bien utile pour distinguer chaque bloc du programme.

<u>Utilisation de UNTIL</u>.- Une autre façon de stopper la boucle est de la compléter par une condition. UNTIL signifie « jusqu'à » en anglais :

```
DO
  INPUT "Quel age avez-vous ?", Age
LOOP UNTIL Age > 18
PRINT "Vous etes majeur"
```

Cette boucle se répète jusqu'à ce que l'utilisateur attribue à la variable `Age` une valeur supérieure à 18 par l'intermédiaire de l'instruction INPUT.

Utilisation de WHILE.- WHILE signifie « tant que » en anglais :

```
DO
  INPUT "Quel age avez-vous ?", Age
LOOP WHILE Age < 18
PRINT "Vous etes majeur"
```

La boucle se répète tant que l'utilisateur attribue à la variable `Age` valeur inférieure à 18 par l'intermédiaire de l'instruction INPUT.

Ces deux exemples sont équivalents. C'est à vous d'utiliser UNTIL ou WHILE selon la situation.

Emplacement de la condition.- La condition imposée peut se positionner soit derrière DO, soit derrière LOOP.

```
Age = 35                            'On initialise preaalablement
DO UNTIL Age > 18
  INPUT "Quel age avez-vous ? , Age
LOOP
PRINT "Vous etes majeur"
```

La variable de contrôle est testée en début de boucle. Ici `Age` étant supérieur à 18, la boucle est stoppée dès le début et les instructions ne sont pas exécutées. L'ordinateur passe directement aux instructions situées après la boucle.

11.3.4.2 La boucle TANT QUE

La boucle `WHILE instruction WEND` a un comportement identique à la boucle FAIRE mais la condition n'est positionnable qu'en tête de boucle :

```
WHILE Age < 18
  INPUT "Quel age avez-vous ?", Age
WEND
PRINT "Vous etes majeur"
```

11.3.5 L'instruction EXIT

L'instruction EXIT permet de sortir d'une boucle POUR ou d'une boucle FAIRE (mais pas d'une boucle TANT QUE). Pour une boucle FAIRE on utilise EXIT DO :

```
DO
  PRINT "Nous allons utiliser l'instruction EXIT"
  EXIT DO                                       'On sort de la boucle
  PRINT "Ces lignes ne seront pas executees car on sort de la boucle"
LOOP
```

et EXIT FOR pour une boucle POUR :

```
FOR i = 1 to 10000
  PRINT "Nous utilisons l'instruction EXIT FOR"
  EXIT FOR                                      'On sort de la boucle
  PRINT "Ces lignes ne seront pas ex\'ecut\'ees car on sort de la boucle"
NEXT i
```

11.3.6 Les tests

L'instruction ON GOTO est associée d'une part à une variable numérique et, d'autre part, à une série d'étiquettes :

```
ON Rep GOTO LabelUn, LabelDeux, LabelTrois
```

L'ordinateur va alors tester la valeur de la variable `Rep`. Si elle vaut 1, l'ordinateur va aller à la première étiquette spécifiée. Si elle vaut 2, il va à la deuxième, et ainsi de suite :

```
INPUT "Entrez un chiffre entre 1 et 3 : ", Rep
ON Rep GOTO LabelUn, LabelDeux, LabelTrois
END

LabelUn:
PRINT "Vous avez ecris 1"
END

LabelDeux:
PRINT "Vous avez ecris 2"
END

LabelTrois:
PRINT "Vous avez ecris 3"
END
```

Si la valeur renvoyée par l'utilisateur n'est pas incluse dans l'intervalle (0 ou 4 par exemple), aucun branchement n'est effectué.

Par contre, si l'utilisateur donne une valeur comprise dans l'intervalle, mais non entière, le branchement est effectué avec l'arrondi. Par exemple, si on donne 1.4, l'arrondi est 1 et on va à l'étiquette `LabelUn`.

L'instruction END signifie la fin d'un programme. Lorsque l'ordinateur la rencontre, le programme s'arrête. Elle peut être placée n'importe où dans le code.

11.3.7 Le test

L'utilisation brute de cette instruction `GOTO` peut sembler incongrue mais en fait elle est souvent associée à l'instruction de **test**. Cette instruction est de la forme :

```
IF expression_logique THEN instruction
```

Le programme suivant utilise cette nouvelle structure de contrôle :

```
10 INPUT "ENTRER UN ENTIER : ";N
20 IF N = 98 THEN PRINT "GAGNE"
RUN
```

```
ENTRER UN ENTIER : 10
RUN
ENTRER UN ENTIER : 98
GAGNE
```

11.4 Aide à la programmation

Nous avons vu ce qu'il est nécessaire pour programmer tout ce qu'on veut. Il existe cependant des outils fort utiles pour faciliter la programmation.

11.4.1 Programmation modulaire

Imaginez que nous avions à exécuter le même code plusieurs fois de suites, à des endroits différents du programme. On peut évidemment tout simplement écrire autant de fois que nécessaire le bout de code correspondant. On peut aussi écrire un **sous-programme** qui sera appelé lorqu'on a besoin.

Syntaxe.- Un sous-programme est un morceau de code précédé d'une étiquette et se terminant par l'instruction RETURN.

On utilise l'instruction GOSUB suivie de cette étiquette pour appeler le sous programme. Lorque le sous-programme rencontre l'instruction RETURN, il renvoie à l'instruction suivant le GOSUB qui l'a appelé.

Exemple.- Écrivons un programme demandant un rayon et affichant l'aire du disque de ce rayon. On fait appel à un sous-programme pour le calcul :

```
CONST PI = 3.14159265458
CLS
PRINT "Ce programme utilise l'emploi de GOSUB et RETURN"
PRINT "pour calculer l'aire d'un disque"
PRINT
INPUT "Entrez le rayon du disque : ", Rayon
GOSUB CalculAire      'Appel du sous-programme
PRINT "L'aire de ce disque est de "; Aire
END

CalculAire:               'Etiquette du sous-programme
Aire = PI * Rayon ^ 2
RETURN                    'Fin du sous-programme, on retourne au GOSUB
```

Remarque.- Comme pour GOTO, GOSUB peut aussi se transformer en ON GOSUB. Ainsi, on pourra tester la valeur d'une variable et aller au sous-programme approprié.

Procédures.- Pour créer une procédure, on peut commencer de deux façons :

- Aller dans le menu « Édition » de l'éditeur intégré et séléctionner « Nouvelle SUB » puis écrire dans l'emplacement « Nom » le nom voulu pour la procédure.
- Se placer à la fin du programme et écrire :
  `SUB Aire`

Il faut maintenant lui dire ce que fait la procédure :

```
SUB Message(Nom$)
PRINT "Bonjour ";Nom$
END SUB
```

L'écriture de la procédure est terminée. Nous allons la tester. Il nous faut revenir dans le corps du programme. Pour cela, allez dans le menu « Afichage » cliquez sur « SUBs » uu appuyez sur la touche F2 (qui est un raccourci clavier). Une fenêtre s'affiche, indiquant toutes les procédures et fonctions du programme. Sélectionnez votre programme (normalement « Sans_nom »).

Pour l'instant, aucune ligne de code ne doit normalement se trouver.

En théorie, on doit ajouter en début de programme la ligne suivante, pour dire QBascic qu'on va utiliser une procédure :

```
DECLARE SUB Message(Nom$)
```

mais elle doit normalement s'écrire d'elle même lors de la sauvegarde du programme.

Pour tester notre procédure, nous allons écrire :

```
Message "Arthur"
```

et 'Bonjour Arthur' est affiché.

Fonctions.- Une fonction est comme une procédure mais elle renvoie une valeur. On n'écrit plus SUB mais FUNCTION :

```
INPUT "Entrez le rayon du disque : ", Rayon
PRINT "L'aire du disque est "; Aire(Rayon)  'Appel de la fonction.
                                            'Notez les parentheses
END
FUNCTION Aire (Rayon)
Aire = 3.14159265458 * Rayon ^ 2
END FUNCTION
```

Remarquez que le résultat est donné par la variable portant le même nom que la fonction.

Remarque.- Une variable qui est utilisée dans une procédure ou une fonction n'est accessible que dans celle ci. On dit que c'est une **variable locale**. Si on veut la rendre accessible par toutes les procédures et fonctions, il faut la déclarer publique en début de programme :

```
COMMON SHARED MaVariable
```

et, dans ce cas, `MaVariable` est une **variable globale**.

On peut aussi créer des variables globales en utilisant juste SHARED suivit des noms de variables dans une procédure ou une fonction.

Une procédure ou une fonction peut modifier la valeur d'une variable lorque celle-ci lui est passée en argument.

11.4.2 Tableaux

Lorsque dans un programme, les données se multiplient, au lieu de créer 4 0000 variables, on peut utiliser un tableaux qui aura pour effet de nous simplifier grandement la tâche.

Déclaration d'un tableau.- Pour créer un tableau, on utilise l'instruction DIM. Pour entreproser les notes de dix élèves, on écrit :

```
DIM Note(9)
```

Remarquez que l'on a écrit 9 et pas 10. PParce qu'on part de 0 et non de 1. Si on avait voulu partir de 1, on aurait écrit :

```
DIM Note(1 TO 10)
```

Si on veut faire commencer tous les tableaux par 1 au lieu de 0, on écrit :

```
OPTION BASE 1 'Tous les tableaux commencent par 1 et non 0
DIM Note(10)
```

Accès à un élément d'un tableau.- Pour stocker nos données dans ce tableau, il faut spécifer l'index de l'élément dans lequel on veut placer la donnée :

```
OPTION BASE 1
DIM Note(10)

Note(3) = 17
Note(7) = 12.5
```

Tableaux à plusieurs dimensions.- On peut utiliser des tableaux à plusieurs dimensions :

```
DIM MonTableau(5, 10)     'Tableau a 2 dimensions
MonTableau(0,2) = 4
```

Type de tableaux.- Les éléments d'un tableau peuvent être d'un type déterminé. Il faudra alors le spécifier avec le mot-clef AS :

```
DIM Bottin(1000) AS STRING 'Un tableau de chaines de caracteres
```

Limites d'un tableau.- Lorqu'on ne connaîit pas les limites d'une dimension d'un tableau, on peut utiliser les instructions LBOUND et UBOUND : LBOUND spécifie l'index inférieur d'une dimension d'un tableau et UBOUND son index supérieur. Pour les utiliser, il faut préciser le tableau concerné et la dimension à tester, en sachant que la première dimension est 1 et non pas 0 :

```
DIM MonTableau(3 TO 27, 4 TO 56)

FOR i = 1 TO 2
  PRINT "La limite inferieure de la dimension "; i; " est "; LBOUND(MonTableau, i)
  PRINT "La limite superieure de la dimension "; i; " est "; UBOUND(MonTableau, i)
NEXT i
```

11.5 Historique

11.5.1 Les langages de programmation

Pour VON NEUMANN, le langage de programmation le plus parfait et le plus universel est le langage machine. Malheureusement le cerveau humain - à part peut-être celui de VON NEUMANN - perdait rapidement pied dans cette suite de 0 et de 1. De plus, si l'on voulait qu'un jour un non-informaticien puisse commander le moindre travail à la machine, il fallait bien qu'il puisse le faire sans pour antant devenir un spécialiste de la structure de l'ordinateur.

Le premier rédacteur d'un langage de programmation plus convivial est Alan TURING, qui veut faciliter l'usage du Manschester MARK 1. Ce premier langage contient cinquante instructions, qui sont automatiquement transcrites en langage machine par l'ordinateur lui-même. La même idée est reprise, par Grace HOPPER, sur l'UNIVAC 1, premier ordinateur civil, qui dispose d'un *short code* ; elle développe ce que la firme appelle la « programmation automatique », un programme interne qui transforme les instructions de l'utilisateur en instructions machine codées en binaire. À partir de ce moment, il est admis qu'un langage de programmation doit servir à écrire des programmes d'une façon qui permette des économies du point de vue du temps d'utilisation de la machine.

FORTRAN

Le premier langage de programmation dit « évolué » est le FORTRAN (pour *FORmula TRANslation*), mis au point sur l'IBM 701 de 1953 à 1956 par l'équipe de John BACKUS. Il faut écrire un programme intermédiaire, appelé *compilateur*, qui traduit un programme écrit en FORTRAN en langage machine. Un programme écrit en FORTRAN peut fonctionner sur n'importe quel ordinateur à condition que, pour chaque modèle, un compilateur approprié soit conçu.

Le premier compilateur écrit pour le FORTRAN à destination de l'IBM 704 est mis au point en avril 1957. Il comporte 25 000 lignes de programme.

COBOL

Le FORTRAN est plus particulièrement dédié aux calculs scientifiques. Le COBOL (pour *COmmon Business-Oriented Language*) est dédié, comme son nom l'indique, aux applications dans le secteur tertiaire. Sa mise au point, achevée en 1960, est financée par le département de la Défense américain, qui souhaite, pour traiter de problèmes de gestion, un langage qui soit indépendant d'un modèle d'ordinateur. Ce langage s'impose à l'administration tout entière comme langage normalisé.

ALGOL

L'ALGOL (pour *ALGOrithmic Language*), mis au point en 1960, est considéré par les utilisateurs comme très théorique et peu utilisé mais, par contre, beaucoup étudié comme modèle des langages à venir. Sa conception, par des chercheurs européens, s'inspire de l'idée qu'il est nécessaire de disposer d'un langage très général, qui ne soit pas lié à une application donnée. Son universalité en fit une réussite pédagogique mais un échec commercial.

BASIC

BASIC (pour *Beginner's All-purpose Symbolic Instruction Code*) est un langage développé au *Dartmouth College* en 1965 sous la direction de J. KEMENY et T. KURTZ, implémenté pour l'ordinateur de *General Electric* G.E. 225. Inspiré du FORTRAN, c'est un langage très simple à apprendre et facile à traduire [GZ-66, KK-66, KK-67].

Sources.- Il existe quatre références majeures pour l'histoire des langages de programmation. Les tout premiers langages de programmation sont traités dans [KP-77] :

> *Cet article passe en revue l'évolution des langages de programmation de « haut-niveau » durant la première décade d'activité de la programmation des calculateurs. Nous discutons des contributions de Zuse en 1945 (le « Plankalkül »), Goldstine et von Neumann en 1946 (« Flow Diagrams »), Curry en 1948 (« Composition »), Maucly* et al. *en 1949 (« Short Code »), Burks en 1950 (« Intermediate PL »), Ru-*

tishauser en 1951 (« Klammerausdrücke »), Böhm en 1951 (« Formules »), Glennie en 1952 (« Autocode »), Hopper et al. *en 1953 (« A-2 »), Laning et Zierler en 1953 (« Algebraic Interpreter »), Backus* et al. *en 1954–1957 (« Fortran »), Brooker en 1954 (« Mark I Autocode »), Kamynin et Lîubimskii en 1954 (« ПП-2 »), Ershov en 1955 (« ПП »), Grems et Porter en 1955 (« BACAIC »), Elsworth* et al. *en 1955 (« Kompiler 2 »), Blum en 1956 (« ADES »), Perlis* et al. *en 1956 (« IT »), Katz* et al. *en 1956–1958 (« MATH-MATIC »), Bauer et Samelson en 1956–1958 (U.S. Patent 3 047 228). Les caractéristiques principales de chaque contribution sont illustrées et discutées. Pour pouvoir les comparer, un algorithme particulier fixé a été codé (autant que faire se peut) dans chacun de ces langages. Cette recherche est fondée sur des sources non publiées et les auteurs espèrent qu'ils ont été capables de compiler une image aussi complète que possible des premiers développements dans ce thème.*

Ensuite un point est fait tous les dix ans dans [Sam-69], [Wex-81] et .

11.5.2 Le BASIC pour les micro-ordinateurs

MITS BASIC

La petite histoire ([Cri-92], ch. 4) dit que Bill Gates se promène dans les jardins de Harvard quand Paul Allen arrive en brandissant le numéro de janvier 1975 de la revue *Popular Electronics* qui traite dans un de ses articles du micro-ordinateur *Altair 8800* de MITS. Dès cet instant, tous les deux pressentent qu'une industrie de la micro-informatique va voir le jour et que cette industrie va avoir besoin de langages de programmation. Bien qu'aucune industrie de logiciels pour micro-ordinateurs n'existe à l'époque, Bill, qui n'a que dix-neuf ans, craint d'arriver trop tard : « *Nous avions peur que la révolution ait lieu sans nous. À la lecture de cet article, nous n'avions plus de doutes à propos de l'orientation de notre vie* ». Gates devine que le premier langage, celui revendu par MITS, revendeur de l'Altair, deviendra le standard unique de toute la micro-informatique.

Ne disposant pas d'un ordinateur *Altair 8800*, Gates et Allen écrivent un programme qui permette de simuler le microprocesseur Intel 8080 utilisé par l'*Altair* sur le mini-ordinateur PDP-10 du centre informatique de l'université d'Harvard. Le BASIC a, en 1975, la réputation d'être le langage de programmation le plus accessible. En six semaines, ils bouclent une version du langage de programmation BASIC qui tourne sur le pseudo-Altair modélisé dans le mini-ordinateur. Ils espèrent qu'il tournera aussi sur un authentique *Altair* doté d'au moins 4 096 octets de mémoire vive. Leur premier essai de fonctionnement sur un vrai micro a lieu le jour où Paul Allen fait une démonstration à l'intention de Ed Roberts, fondateur de MITS, au siège de l'entreprise à Albuquerque. À leur grande surprise et soulagement, ça marche.

Le MITS BASIC, comme on l'appelle, utilise de manière particulièrement efficace la mémoire vive et, avancée hardie, propose des commandes permettant aux programmeurs de gérer directement cette mémoire. Gates abandonne ses études à Harvard. Allen quitte son emploi de programmeur chez *Honeywell* et tous les deux partent s'installer dans le Nouveau-Mexique pour être proches de leur clientèle. Ils s'installent provisoirement au *Sundowner Motel* sur la Nationale 66.

Gates et Allen ne restreignent pas leur champ d'intérêt à MITS. Ils écrivent des versions de BASIC pour d'autres micros dès leur mise sur le marché. Ils finissent par se quereller avec Ed Roberts parce que celui-ci revendique la propriété de MITS BASIC et de ses dérivés. Le conflit se termine à l'avantage de Gates et Allen. Ils fondent *Microsoft* dans le but clairement établi de mettre « *un ordinateur qui utiliserait des logiciels Microsoft sur chaque bureau et dans chaque foyer* ».

BASIC Apple

Le micro qui est un vrai succès commercial est l'Apple II. Il est, plus que tout autre micro jamais conçu, l'œuvre d'une seule personne, puisque même son interpréteur BASIC, toujours disponible séparément sous la forme d'un programme en mémoire vive, est l'œuvre de Steve WOZNIAK. Les deux autres succès, le *Commodore* et le *Radio Shack*, possédent aussi un BASIC intégré.

IBM

Lorsque IBM se lance dans la conception d'un micro-ordinateur en 1980, la firme choisit *Microsoft* pour concevoir son interpréteur BASIC puisque c'est le plus ancien et le plus connu.

11.6 Bibliographie

[Cri-92] CRINGELY, Robert X., **Accidental Empires**, Addison-Wesley, 1992 ; tr. fr. **Bâtisseurs d'empires par accident : origines et dessous de la Silicon Valley**, Addison-Wesley, 1993, VIII + 307 p.

[Dit-92] DITTRICH, Stefan, **Le grand livre du Qbasic**, Data Bekker, 1992 pour l'édition allemande. Traduction française, Éditions Micro Application, 1993, 710 p.

[GZ-66] **'BASIC' Language Reference Manual**, General Electric Information Systems Division, sept. 1966 (revised).

[KK-66] KEMENY, J. G., KURTZ, T. E., **BASIC (User's Manual)**, 3rd ed., Dartmouth College Computation Center, Hanover, N. H., jan. 1966.

[KK-67] KEMENY, J. G., KURTZ, T. E., **BASIC Programming**, Wiley, 1967.

[KP-77] KNUTH, Donald E., TRABB PARDO, Luis, *The Early Development of Programming Languages*, in BELZER, J., HOLZMAN, A. G., and KENT, A. (eds), **Encyclopedia of Computer Science and Technology**, vol. 6, Dekker, New-York, 1977, pp. 419–493. Reprinted in [Met-80], pp. 197–273. Traduction française dans Donald E. Knuth , **Éléments pour une histoire de l'informatique**, articles choisis et traduits par Patrick CÉGIELSKI, Lecture Notes 190, CSLI Publications, Stanford et Société Mathématique de France, 2011, xvi + 371 p.

[Met-80] METROPOLIS, N. and HOWLETT, J. and ROTA, Gian-Carlo, eds, **A History of Computing in the Twentieth Century**, Academic Press, 1980.

[Sam-69] SAMMET, Jean E., **Programming Languages : History and Fundamentals**, Prentice-Hall, 1969, XXX + 785 p.

[Wex-81] WEXELBLAT, Richard, ed., **History of Programming Languages**, Academic Press, 1981.