PETITE INITIATION
au
TURBO BASIC

TABLE DES MATIÈRES

I – INTRODUCTION

Un langage de programmation est un outil qui permet de faire exécuter à un ordinateur un certain nombre de tâches : instructions ou fonctions.

Il n'y a pas de miracle, le nombre des possibilités n'est pas infini et pour faire exécuter ce que l'on désire, il faut :

– avoir une bonne connaissance du problème à traiter, savoir le structurer en petits éléments qui assurent un enchaînement logique et sans contradiction, car un ordinateur ne fait que ce que les ordres écrits lui disent de faire,

– connaître le langage, au moins en partie, et savoir où trouver le renseignement ou l'explication nécessaire (un voisin est souvent plus utile qu'un gros fascicule),

– en connaître les limites (entre autre pour la précision des calculs),

– traduire le but recherché en ordres compréhensibles par la machine.

Exemple simple de tracé d'un cercle :

Compas et une feuille de papier	Ordinateur
Tracer un cercle c'est : – prendre une feuille de papier – prendre un compas – régler le compas à la bonne ouverture et positionner le centre sur le papier, – tracer le cercle en faisant tourner le compas.	Tracer un cercle sur l'écran : – lancer le programme, écrire 3 lignes où il y a – effacer l'écran, – utiliser la fonction CIRCLE, en donnant la position du centre et le rayon – ordre de fin – faire exécuter la fonction...

Tout ce que nous faisons inconsciemment dans notre vie courante pour réaliser nos objectifs doivent être décomposés et repensés pour les faire exécuter par l'ordinateur, s'il a les fonctions adéquates.

Le langage simple utilisé est le **BASIC**. Pourquoi ? D'abord, c'est un **interpréteur**, chaque ligne de commandes est immédiatement interprétée par l'ordinateur quand elle doit être exécutée contrairement à d'autres langages qui doivent passer par une compilation (analyse et transcription en codes machines) puis charger dans un module contenant tout. En BASIC, tout changement est immédiatement pris en compte puisque chaque ligne est réanalysée et traitée à chaque exécution, mais on perd en vitesse. Quand le programme est au point, en Turbobassic, on peut créer un fichier programme que l'on pourra exécuter sans passer par le Turbobasic.

Le langage BASIC est un bon langage d'initiation parce que les ordres compréhensibles par la machine sont relativement clairs pour le novice avec une petite adaptation à l'anglais :

PRINT	faire écrire
GOTO A	aller au label A:
CLS	abréviation de Clear screen effacer l'écran...
XBORD = 25	mettre 25 dans la mémoire appelée XBORD
etc ...	

Le tracé du cercle au centre de l'écran de diamètre moitié de l'écran se réduit à :

```
CLS
CIRCLE(320,240),50
```

Dans ce fascicule vous trouverez les possibilités élémentaires du BASIC classique. Pour une connaissance plus approfondie et plus détaillée du BASIC et de la programmation de votre ordinateur, se référer au manuel de base livré avec le logiciell.

II – BASIC GENERALITES

II – 1 – MISE EN MÉMOIRE

Toutes valeurs numériques, caractère ou chaîne de caractères peuvent être rangés dans la mémoire de l'ordinateur avec des noms de variables (car leur contenu peut être changé à tout moment)

On distingue :

– les mémoires alphanumériques contenant des caractères ou des chaînes de caractères (string en Anglais). Caractères simples : A, B, C, d, y, %, *, 5, 0,:, etc . . . ou chaînes de caractères placées entre guillemets "JULES FAIT DE L'INFORMATIQUE". Les noms de mémoires contenant de l'alphanumérique doivent être terminés par un '$' (ex.: A$, NOM$, TH$, REPONSE$ . . .).

– les mémoires numériques contenant des nombres, subdivisées en entier (integer) ou entier naturels, les flottants simple précision (décimaux peu précis) et double précision (décimaux précis).

On les identifie par le dernier caractère du nom de la variable :

– terminé par '%' entiers naturels (I%, NITER%, ZEBU%)
– terminé par '!' flottants 8 digits significatifs (AA!, PAYE!, MOY!)
– terminé par '#' double précision 16 digits significatifs (RESUL#, TT#)

Par défaut, s'il n'y a pas de %,! ou #, la variable sera genre *flottant simple précision.*

On peut définir des classes de variables par les fonctions DEFINT, DEFSNG, DEFDBL, DEFSTR :

DEFINT I-K définira toutes les variables dont les noms commencent par I, J ou K comme entières sans avoir à mettre un "%" en fin du nom.
DEFSTR T définira toutes les variables commençant par T comme variables chaîne sans "$" à la fin.
DEFDBL A-C,W-Z définira les variables commençant par A, B, C, W, X, Y, Z comme double.

Il y a deux façons de mettre en mémoire :

– par rangement direct

A = 12	on met 12 dans la mémoire A
ZA = R2 + 5 – FD/N	on range le résultat du calcul dans ma mémoire ZA
C$ = "OUF"	la chaîne de caractère est rangée dans la mémoire C$

– par saisie au clavier : INPUT, LINE INPUT, INKEY$ ou par lecture de fichiers (INPUT# ou LINE INPUT#). LINE INPUT ne prend qu'une chaîne de caractères. INKEY$ donne le dernier caractère tapé au clavier sinon rien.

INPUT "Nbre d'itérations :";NITER écrit à l'écran "Nbre d'itérations" et attend que l'on rentre une valeur au clavier suivie de la frappe de la touche "Entrée". Cette valeur sera rangée dans la variable NITER.

```
INPUT "Option s.v.p. :";NOPT
```

Attention, le texte entre guillemets est optionnel, il permet de renseigner l'utilisateur sur ce qu'il faut rentrer en mémoire pour continuer le programme. Mais le programmateur est libre de ne rien mettre (INPUT NOPT), de mettre des injures ou toute autre flatterie. La syntaxe de INPUT ci-dessus est l'équivalent de :

```
PRINT "Option s.v.p. ";
INPUT NOPT
```

ce qui veut dire, que le *texte entre guillemets n'a rien à voir avec ce qui sera mis en mémoire.*

Tableaux

Le BASIC permet de réserver de la place pour une variable qui doit contenir plusieurs valeurs (tableau) par l'ordre **DIM**

```
DIM T(30), POSX(20)
```

Pour accéder à chaque valeur du tableau, on se sert de son rang repéré par un indice fixe : T(1), T(2), T(3).... ou par un

indice variable T(I). Pour faire lister, par exemple toutes les valeurs du tableau T(30) :

```
FOR IND = 1 TO 30
PRINT T(IND)
NEXT IND
```

Les tableaux peuvent être à plusieurs dimensions

```
DIM PTXY(20,2)     pour un tableau à deux dimensions
```

II – 2 – PROGRAMMES

Dans un éditeur de texte, (dans l'éditeur du Tturbo Basic ou le Notepad par exemple), on peut écrire des lignes de texte numérotées de façon croissantes. Chaque ligne correpondant à un ordre basic.

Quand on exécute le programme, chaque ligne successivement est lu et envoyée à l'interpréteurqui la fait *exécuter immédiatement* si la syntaxe est correcte.

Un programme est une suite de lignes d'instructions numérotées mises ainsi en mémoire. Elles ne seront exécutées que lorsque l'ordre **RUN** sera donné. Alors les instructions seront exécutées *séquentiellement* suivant l'ordre croissant des numéros sauf si une instruction force le déroulement à passer à une ligne antérieure ou postérieure.

PRINT "BONJOUR" fait écrire BONJOUR à l'écran mettra quand l'exécution des lignes arrivera à ce numéro.

Plusieurs instructions peuvent être écrites sur une même ligne, mais il faut qu'elles soient séparées par des ':' (deux points).

```
CLS : PRINT "BONJOUR" : BEEP
```

Sauvegarde des programmes

Lorsque l'on arrête l'ordinateur, les mémoires perdent leur contenu : variables et programme disparaissent de la mémoire. Pour conserver un programme pour une utilisation ultérieure, il faut le "sauver" sur un support magnétique permanent comme une disquette, un disque ou une bande magnétique.

Pour sauver un programme sous Turbo Basic, sortir de l'Editeur par la touche **Esc**, et dans le menu **FILE** choisir **SAVE** ou **WRITE** si c'est un nouveau programme.

Pour le récupérer

FILE / **LOAD** et sélectionner le nom du programme

Noms de programmes

Convention des noms de programmes : vous avez droit à 8 caractères, le BASIC ajoutant un suffixe '.BAS'pour rappeler qu'il est écrit en BASIC.
Le première caractère doit être une lettre de l'alphabet. Sinon on peut utiliser les chiffres et autres caractères. Ne sont pas acceptables les caractères suivants ; : " , . ? *.

Lignes commentaires

Toute ligne programme commençant par REM (remarque) ou l'apostrophe ' n'est pas exécutée, mais sert simplement à mettre dans le programme des commentaires pour en faciliter la relecture. Il est conseillé de *ne pas avoir peur d'abuser de commentaires* qui permettent de mieux structurer l'ensemble.

Entre autre en début mettre le nom du programme, auteur, date, date de révision, but du programme etc ...

```
' prg SOUPE.BAS 25/12/89  – menu diététique

' boucle sur le nombre de caractères...
```

Labels et numeros de lignes

Dans les premières versions du Basic, il fallait numéroter les lignes séquentiellement. Ceci peut toujours se faire sous Turbo Basic, mais cela n'est plus nécessaire. Par contre, pour retourner à une ligne lors d'un branchement, il faut introduire des **LABEL**s c'est à dire des noms suivis de "**:**" qui serviront à préciser les retours à une lignes précédentes ou ultérieure.

Exemple :

```
LAB1:
...
...
INPUT OPT$
IF OPT$ = “S” GOTO LAB2 ELSE GOTO LAB1
...
...
LAB2:
```

A la ligne input, le programme attend que vous rentriez un caractère. Si ce caractère est “S”, le programme continuera à la ligne LAB2, sinon il retournera à la ligne LAB1.

II – 3 –FONCTIONS MATHÉMATIQUES ET CALCUL

Opérateurs arithmétiques

+ – / *	les quatre opérations (attention au signe : / diviser à la place de : et * multiplier au lieu de x ou .)
@	division entière ou \
^	élévation à une puissance
MOD	fonction modulo
–	négation

Fonctions mathématiques

SIN, COS, TAN et fonctions inverses ASIN, ACOS, ATAN.

INT	le plus grand entier inférieur ou égal
FIX	la partie entière
CINT	l'entier le plus proche
EXP	fonction exponentielle
LOG	fonction logarithme naturel
SQR	fonction racine carrée
ABS	fonction valeur absolue

Opérateurs logiques

NOT AND OR XOR EQV IMP
travaille bit à bit.

II – 4 – TEST ET CONDITIONS

Teste si (**IF**) une condition est réalisée, si c'est vrai (**THEN**) on fait ce qui suit sinon (**ELSE**) ont fait autre chose.

Syntaxe générale :	IF -	THEN -	ELSE -
	SI condition vraie	ALORS on fait ceci	SINON on fait cela
ou	IF -	GOTO -	ELSE -
	SI condition vraie	ALLER A numéro ligne	SINON on fait cela

Forme simplifiée : SI condition vraie ALORS on fait ceci
sinon on continue à la ligne suivante

Opérateurs logiques

=	égalité	<	plus petit que
<=	plus petit ou égal à	>=	plus grand ou égal
<>	différent de	<	plus petit que

```
IF REP$ = "F" THEN PRINT "C'EST FINI" : END
IF A > BUT THEN PRINT "TROP GRAND" ELSE PRINT "TROP PETIT"
```

Une variable numérique nulle est considérée comme <u>fausse</u> sinon elle est vraie :

IF XP THEN XP = XP–1 diminue XP de 1 tant que cette valeur n'est pas nulle.

II – 5 – CHAINES DE CARACTÈRES

Est caractère tout ce qui peut s'écrire sur l'écran ou sur imprimante. Les lettres de l'alphabet (A,B,C... a,b,c...), les chiffres (0,1,2...), les signes de ponctuation (.,;:!? >< . . .), les signes mathématiques (+,–,*,/), les signes spéciaux tels @, #, $, { etc...

Une chaîne est une suite de caractères. Elle doit être placée entre guillemets et sa mise en mémoire se fait dans des variables dont le nom se termine par un '$'.

```
PRINT "Ca va ?"
A$ = " Je viendrai le 3 JUIN."
PRINT A$
```

Traitement des chaînes

Les opérations possibles sur les chaînes sont l'addition et les comparaisons.

– l'addition des chaînes s'appelle la **concaténation** et consiste à mettre bout à bout plusieurs chaînes pour n'en former qu'une seule.

– comparaison : on peut tester l'**égalité** de deux chaînes, leur **inégalité**, et l'**ordre** de deux chaînes. Une chaîne est plus petite qu'une autre si le code ASCII du premier caractère de la première chaîne est plus petit que le code du premier caractère de la seconde. Si les premiers caractères sont identiques on compare les seconds, etc... Pour le Code ASCII voir paragraphe ci-dessous.

"BATEAU" < "NAVIRE" car le code de "B" est plus petit que le code de "N".
"BATEAU" > "BARQUE" en comparant les 3èmes lettres.

Code ascii (**A**merican **S**tandart **C**ode for **I**nformatique **I**nterchange).

Chaque mémoire élémentaire de l'ordinateur est faite de 8 mémoires simples (bit) ou l'on peut mettre 0 ou 1. La mémoire élémentaire s'appelle un **OCTET**.

En comptant en binaire (base 2), on peut mettre dans un octet toutes les valeurs allant de 0 à 255.

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
36 s'écrit dans 1 octet	0	0	1	0	0	1	0	0	$2^5 + 2^2 = 36$
135	1	0	0	0	0	1	1	1	$2^7 + 2^2 + 2^1 + 2^0 = 135$

Chaque caractère est inscrit dans un octet, mais sous forme codée (voir tableau annexe B). Le code le plus utilisé en BASIC est le code **ASCII**. La lettre "A" est codée 65 en décimal, c'est à dire 01000001, "B" codée 66 etc...

Les autres signes aussi sont codés : "+" est codé 43, le "." 44 etc... Les chiffres comme caractères sont aussi codés "0" 48, "1" 49 etc... et pour comparer des chaînes on compare les valeurs des codes des caractères.

Instructions et fonctions

Fonction	définition	example
ASC$(ch) ASCII code ASCII	donne le code ASCII du 1er caractère de la chaîne ch	F$ = "BONJOUR" G = ASC(F$) PRINT G donne : 66
CHR$(code) character caractère	donne le caractère dont le code ASCII est indique par code	A$ = CHR$(79) + CHR$(85) PRINT A$ donne : OU
INSTR(ch1,ch2) in string dans la chaîne	donne la position de la chaîne 2 dans la chaîne 1	A$ = "BATIFOLER" B$ = "FOL" PRINT INSTR$(A$,B$) donne : 6

INSTR(i,ch1,ch2)	donne la position de la chaîne 2 dans la chaîne 1, mais la recherche se fait à partir du ième caractère.	A$ = "RANTANPLAN" B$ = "AN" PRINT INSTR(6,A$,B$) donne : 9
LEFT$(c,n) left gauche	renvoie les n premiers caractères de la chaîne c	PRINT LEFT$("TRUCAGE",4) donne : TRUC
LEN(ch) lenght longueur	longueur de la chaîne	D$ = "BOUTON" PRINT LEN(D$) donne : 6
MID$(c,n,m) middle milieu	extrait de la chaîne c, à partir du caractère n, m caractères	B$ = "AVENTURE" PRINT MID$(B$,2,4) donne : VENT
RIGHT$(c,n) right droite	renvoie les n derniers caractères de la chaîne c	X$ = RIGHT$("PARACHUTE",5) PRINT X$ donne : CHUTE
STR$(n) string chaîne	conversion d'une valeur numérique en chaîne	R1 = –19.65 : R2 = 33.5 PRINT R1 + R2 donne : 13.85 PRINT STR$(R1) + STR$(R2) donne : –19.65 33.5
VAL(c) value valeur	donne la valeur numérique du nombre représenté par les premiers caractères de chaîne ou 0 si la chaîne ne commence pas par un nombre (les signes + ou – sont compris).	PRINT VAL("127 RUE LONGUE") donne :127 PRINT VAL(NUMERO 127) donne : 0

Remarque : les fonctions qui se terminent par un "$" donnent un résultat "*caractère*", c'est-à-dire une chaîne ou un caractère, les autres un résultat "*numérique*".

II – 6 – INSTRUCTIONS ÉCRAN

Les instructions et fonctions décrites sont des fonctions de bases. Pour chaque Basic il existe des tas d'autres fonctions que l'on trouve dans les manuels.

Les [] indiquent que ces données sont facultatives.

Fonction	définition
CLS	efface l'écran.
COLOR c1,c2 [,i]	détermine la couleur de l'écriture des caractères, c1 : couleur des caractères c2 : couleur du fond des caractères i : inversion avec i = 0 ou 1, inversion coul caract et fond.
CSRLIN	donne le numéro de ligne où se trouve le curseur.
DEFGR$(n)	construction de caractères graphiques n : numéro de caractère (voir annexe C)
LINE	trace un segment défini par ses extrémités – mode graphique : LINE(x1,y1)–(x2,y2),col – mode caractères : LINE(x1,y1)–(x2,y2),car,c1,c2,c3
LOCATE c,l [,m]	positionne le curseur à la colonne "c" et à la ligne "l" m = 0 : curseur absent m = 1 : curseur présent (valeur par défaut).
POINT (x,y)	donne le code de la couleur du point de l'écran en (x,y).
POS	donne la position du curseur (colonne).

PSET (x,y)	allume un point ou un caractère mode graphique : PSET(x,y),c mode caractère : PSET(x,y),car,c1,c2,c3	
SCREEN	configure l'écran graphique SCREEN *n* *n*= 1 écran CGA (320x240) *n* = 2 écran SCGA (640x320) *n* = 11 écran VGA (640x480) Noir et blanc *n* = 12 écran VGA (640x480) couleur	L'origine de l'écran étant en haut à gauche. Par défaut, l'écran n'est pas graphique.

Gestion de l' écran

En définissant par l'ordre SCREEN 12 l'écran, on définit un écran de 24 lignes de 80 caractères ou un écran de 640x480 points de résolution.

Pour placer un texte au point écran (X,Y) par l'ordre LOCATE n° ligne, n° colonne, il faut diviser les valeurs X et Y par 8.

Pour passer des coordonnées caractères aux coordonnées graphique, il faut utiliser un facteur 8.
L'écran texte est limité à 24 lignes seulement.

Couleurs

couleurs	code couleur	couleur	code couleur
noir	0	gris pale	8
bleu royal	1	bleu	9
vert foncé	2	vert clair	10
bleu turquoise	3	bleu clair	11
rouge	4	rouge clair	12
violet	5	mauve	13
brun	6	jaune	14
gris	7	blanc	15

I – 8 – RÉPÉTITIONS, BOUCLES ET BRANCHEMENTS

Quand il faut répéter un certain nombre de fois les mêmes instructions dans un programme, on fait ce que l'on appelle une boucle : on revient sur une partie du programme qui a déjà été exécutée pour la recommencer.

Instruction FOR ... NEXT

Si l'on connaît le nombre exact de répétitions, on exécute une boucle appelée FOR ... NEXT.

Syntaxe :

```
FOR compteur = val_ dep TO val_arriv STEP incr
...
lignes de programmes à répéter
...
NEXT compteur
```

'*compteur*' est une mémoire numérique qui variera au cours des boucles de '*val_dep*' jusqu'à '*val_arriv*' par saut de '*incr*'. Si STEP incr est absent, il vaut 1 par défaut.

val_dep est un nombre ou une mémoire contenant un nombre. Idem pour *val_arriv* et *incr*. Les valeurs de départ, arrivée et incrément peuvent donc être calculées dans le programme ou bien fixées une fois pour toute par le programmeur.

L'incrément peut être négatif, alors *val_dep* est plus grand que *val_arriv*.

Ex.: table de multiplication par deux

```
FOR I = 1 TO 10
PRINT 2 * I
NEXT I
```

Instruction GOTO

Si l'on ne connaît pas le nombre de répétitions à exécuter, à la fin de la séquence, on renvoie l'exécution en début de

la séquence par un GOTO (aller à), mais il faudra qu'à l'*intérieur de la séquence, il y ait un test d'arrêt* pour pouvoir poursuivre le programme ailleurs lorsque les conditions sont requises.

```
CLS
A: : INPUT "nombre s.v.p. ";N
IF N <= 0 GOTO B
PRINT N;" x 3 = ";N* 3
GOTO A
B: ...
```

Instructions ON ... GOTO ..., ON ... GOSUB ...

D'autres instructions permettent de quitter une séquence pour se brancher à une autre ligne, mais avec un choix suivant une variable donnée: ON ... GOTO ..... et ON ... GOSUB .....

Syntaxe : ON *variable* GOTO *LAB1,LAB2,LAB3....*

si *variable* vaut 0, le programme continu à la ligne suivante,
si *variable* = 1 le programme continu à la ligne de label *LAB1*
si *variable* = 2 le programme continu à la ligne de label *LAB12*
. . .
si *variable* est plus grand que le nombre de ligne écrite, le programme continu à la ligne suivante.

Idem pour ON ... GOSUB ..., mais on se branche à un sous-programme, et au retour de celui-ci, le programme continue à la ligne suivante.

II – 9 – SOUS-PROGRAMMES

Toutes parties d'un programme qui sont similaires en différents endroits peuvent être mises sous forme d'un même sous-programme.

Les lignes d'un sous-programme commencent à un numéro quelconque du programme et doivent se terminer obligatoirement par l'instruction **RETURN**.

Pour exécuter un sous-programme au cours d'un programme, il suffit d'écrire l'ordre GOSUB *label*, *label* label du début du sous-programme.

```
CLS
4 tracé d'une étoile en 40,50 de l'écran
XE = 40 : YE = 50
GOSUB POS
..
.
' tracé d'une étoile en 110,20
POS:
XE = 110 : YE = 20
GOSUB ETO
.
.
ETO:
' tracé d'étoile
LINE(XE–5,YE)–(XE+5,YE)
LINE(XE,YE–5)–(XE,YE+5)
LINE(XE–5,YE+5)–(XE+5,YE–5)
LINE(XE–5,YE–5)–(XE+5,YE+5)
RETURN
```

II – 10 – ENTRÉES SORTIES DES DONNÉES – FICHIERS

Un fichier sert à sauvegarder de façon permanente (ou presque, attention aux aimants baladeurs) des données sur supports magnétiques : valeurs ou textes. Les fichiers sont désignés par des noms. Chaque donnée, valeurs ou texte est écrite sous forme codée (code ASCII ou binaire le plus souvent) dans des enregistrements.

Les noms de fichiers sont composés d'une partie principale (8 caractères maximum) suivi de façon optionnelle d'un suffixe (3 caractères maximum). S'il y a un suffixe, un point le sépare de la partie principale.

– le nom comporte un à huit caractères, les lettres de l'alphabet, les chiffres et autres, exceptés les virgules, ? , ; : *, et le point qui sert de délimiteur entre le nom et le suffixe.
– le suffixe optionnel comprend un à trois caractères mêmes restrictions que pour la partie principale.

Exemples : TELEP.PAR RECET.DAT ADRES01.DAT RESUL.A15 ...

Pour gérer des données sur disque ou autre support magnétique, il faut établir une liaison avec le support considéré en donnant l'indication d'écriture ou de lecture des données : fonction **OPEN** (ouverture d'un canal), lire les données par les

fonctions **INPUT#** ou **LINE INPUT#**, écrire les données par **PRINT#**, tester la fin d'un fichier par la fonction **EOF**, et fermer le fichier **CLOSE**.

Fonction	syntaxe	définition
OPEN ouvrir	OPEN "O",#numéro canal,"nom de fichier"	"O" (out) : ouverture du fichier en écriture. Jusqu'à 16 canaux peuvent être ouverts simultanément. Exemples : OPEN "O",#4,"BUDGET" ou A$ = "BUDGET" : OPEN "O",#4,A$
	OPEN "I",#numéro canal,"nom de fichier"	"I" (in) : ouverture du fichier en lecture. Exemple : OPEN "I",#1,"TELEPH" A: : INPUT#1,NUR IF EOF(1) THEN CLOSE #1:END PRINT NUR : GOTO A
PRINT# écrire, imprimer	PRINT #no canal,liste des variables numériques ou alpha numériques.	Lecture des données Exemple : PRINT#2,A,B,C,TEX$
INPUT# **LINE INPUT#** ligne mettre dedans	INPUT #no canal,liste des variables numériques ou alpha numériques. LINE INPUT #no canal,variable chaîne. chaîne : au maximum 256 caractères.	Exemple : INPUT#5,A,B,T$ LINE INPUT#1,C$
EOF end of file fin de fichier	EOF(numéro de canal)	cette fonction donne –1 (vrai) si la fin d'un fichier est atteinte, sinon 0 (faux).
CLOSE fermer	CLOSE #numéro de canal. CLOSE seul ferme tous les canaux ouverts.	Cette instruction ferme le canal de numéro donné.

II – 11 – TRAITEMENT DES ERREURS

Le BASIC donne la possibilité de traiter les erreurs qui se produisent dans un programme en définissant deux variables ERR et ERRL qui contiennent le code d'erreur (ERR) et le numéro de ligne où s'est produit l'erreur (ERRL). Ces erreurs peuvent être traités par les instructions ON ERROR GOTO et RESUME.

ON ERROR GOTO *label* qui permet de retourner à la ligne *label* de programme quand une erreur est détectée,
RESUME qui permet de continuer après une erreur en un endroit choisi :

RESUME [0] reprend à la ligne où s'est produit l'erreur,
RESUME NEXT reprend à la ligne suivante,
RESUME *n* reprend à la ligne *n* du programme.

II – 12 – DIVERS

Fonction	définition
RND	tire une valeur au hasard entre 0. et 1.
RANDOMIZE	initialise le générateur de pseudo-hasard
OCT$	transforme un entier en chaîne contenant sa valeur octale (base 8)
HEX$	transforme un entier en chaîne contenant sa valeur hexadécimale (base 16)
DATE$	variable système donnant la date de l'horloge interne de l'ordinateur
TIME$	variable système donnant l'heure de l'horloge interne de l'ordinateur
CLEAR	instruction permettant de remettre des mémoires à zéro, de réserver de la place pour des grandes chaînes, pour des caractères graphiques, et des programmes binaires.
STOP	force un programme à s'arrêter à cette ligne
SWAP	permet d'intervertir les valeurs de deux variables IF A < B THEN SWAP A,B

Pour plus de précision sur l'existence et la syntaxe de ces instructions, fonctions et variables, se reporter à votre fascicule habituel.

Toutes les fonctions, variables ou instructions ne sont pas forcément implémentées sur l'ordinateur que vous utilisez, par contre d'autres peuvent exister, à vous d'aller à la pêche pour utiliser au mieux votre 'bécane'.

II – 13 – CONSEILS PRATIQUES – RAPPELS

❒ Il est illusoire en simple précision de faire de grands calculs (7 chiffres significatifs). L'opération 10000. + .00001 n'a pas de sens.

❒ L'ordinateur n'est capable d'exécuter des instructions que si leur syntaxe est correcte.

❒ La connaissance du sens des mots anglais et des abréviations des instructions et fonctions facilite la compréhension (voir feuille lexique).

❒ Les lignes de programmes seront exécutées séquentiellement suivant l'ordre de leur numéro par l'ordre "RUN" sauf branchement (GOTO) à une autre ligne ou répétitions.

❒ Tout programme n'est souvent que la prise de conscience qu'il faut faire faire à l'ordinateur tout ce que l'on fait inconsciemment par habitude et réflexe.

❒ Le signe "=" n'a pas le sens mathématique d'égalité, mais celui de rangement ou affectation à une mémoire.

```
A = 10.5        on range 10.5 dans la mémoire appelée A.
P1$ = "Toto"    on range la chaîne de caractère "Toto" dans la mémoire P1$.
```

❒ Les valeurs décimales sont codés avec un point "." (syntaxe anglo-saxonne), la virgule est un séparateur de données.
10.6 est une valeur de 10 et 6/10èmes,
10,6 est la suite de deux valeurs 10 et 6.

❒ On distingue les mémoires contenant des valeurs numériques de celles contenant les caractères ou chaînes de caractères par l'adjonction d'un "$" à la fin du nom des mémoires chaînes.

```
attention    A = 2              2 est un nombre,
             A$ = "J'AI 2 ANS"  2 est un caractère.
```

❒ Sauf non ambiguïtés, les chaînes de caractères sont mises entre guillemets. L'ordinateur distingue Z = 2 de Z$ = 2, mais Z = "TOTO" est refusé (sauf s'il y a eu déclaration DEFSTR T).

❒ Pour enchaîner plusieurs instruction sur une même ligne, il faut les séparer par le caractère ":"

```
A = 20 : B + 31.6 : PRINT "somme : ";A+B
```

❒ Toute parenthèse ouverte dans une expression numérique doit être fermée, l'erreur sera signalée par le BASIC.

❒ Ne pas avoir peur dans des calculs d'utiliser les parenthèses qui permettent de lever toute ambiguïté sur l'ordre des calculs intermédiaires.

❒ La fonction INKEY$ est extrêmement commode, pour faire attendre un programme et continuer en appuyant sur n'importe quelle touche mettre la ligne :

```
RET: :  IF INKEY$ = "" GOTO RET        tant que INKEY$ est vide, on boucle sur la même ligne
```

III – PRATIQUE DU BASIC

Chaque partie de ce chapitre vous propose quelques petits exercices sur des sujets simples pour apprendre à manier concepts, instruction, fonctions et raisonnements en BASIC.

Il est conseillé de broder autour du sujet principal pour exploiter et comprendre toutes les possibilités des différentes fonctions et instructions.

ENTREES – SORTIES ÉCRAN : VALEURS et TEXTES

◆ Ecrire sur l'écran vide et au centre, un texte de bienvenue : Bonjour, bonsoir, salut...)
(CLS, LOCATE, PRINT)
◆ La même chose, mais en grand caractères.
(CLS, LOCATE, PRINT, ATTRB)
◆ Demander un prénom et faire écrire, centré automatiquement, Bonjour untel à l'écran.
(CLS, LOCATE, PRINT, LEN, INPUT)
◆ Avec les fonctions TIME$ et DATE$ faire afficher la date et l'heure. Si possible encadrer et mettre de la couleur.
(CLS, LOCATE, PRINT, DATE$, TIME$, LINE, COLOR, BEEP)
Faire sonner à la demi et aux heures.

MANIEMENTS DES CHAÎNES ET CARACTERES

◆ Entrer un titre au clavier, le faire écrire en gros caractères en intercalant un blanc entre chaque caractère.
(CLS, INPUT, LEN, ATTRB, MID$, LOCATE, PRINT)
◆ Du programme précédent, y introduire des lettres de couleurs (attention à ne pas mettre la couleur du fond).
(CLS, INPUT, ATTRB, LEN, MID$, LOCATE, PRINT, COLOR, IF ... THEN)
◆ Entrer un nom au clavier et le faire écrire inversé.
(CLS, INPUT, LEN, MID$, LOCATE, PRINT)

DESSIN GRAPHISME

◆ Construire une pyramide vue de dessus avec donnés en entrée au clavier, la largeur du côté, et la position centrale et le nombre de marches. Faire les tests si elle tient dans l'écran, sinon redemander position ou largeur.
(CLS, INPUT, IF ... THEN, GOTO, BOX, FOR ... NEXT)
◆ Tracer la fonction $y = x^2$ sur l'écran pour x allant de 0 à 10, l'écran représentant une feuille de 0 à 10 en abscisse et 0 à 100 en ordonnées.
◆ Tracer la fonction $y = x^3$ pour x allant de –4 à +4. Cadrer dans l'écran.

MOUVEMENT

◆ Faire traverser une lettre à l'écran.
(CLS, PRINT, LOCATE, GOTO ou FOR ... NEXT avec IF)
◆ Déplacer un caractère au moyen des flèches du curseur
(INKEY$, LOCATE, PRINT, CLS, IF, code ASCII)
◆ La même chose, mais si la lettre bute sur un bord, la faire réapparaître de l'autre côté.
(INKEY$, LOCATE, PRINT, CLS, IF, code ASCII)
◆ Entrer un nom au clavier. Le faire inscrire au centre en faisant glisser une à une ses lettres de la droite à sa place au centre.
(INPUT, LOCATE, PRINT, FOR ... NEXT, MID$, LEN...)

RÉPÉTITIONS ET BOUCLES

◆ Faire écrire 10 fois "l'INFORMATIQUE c'est formidable".
◆ Une table de multiplication n'est que la répétition d'une opération dans laquelle une valeur varie. Avec une boucle FOR ... NEXT se servir du compteur de la boucle pour faire varier cette valeur. Utiliser pour la présentation les instructions PRINT USING ou TAB.

FICHIERS DE DONNÉES

◆ Lire un fichier texte en comptant ses lignes, ses caractères. Le faire écrire et après donner les résultats.
◆ De ce programme ajouter le comptage d'apparition de chaque lettre de l'alphabet. Écrire les résultats sur deux colonnes puis ensuite tracer l'histogramme.
Fichiers d'essai P1.DAT ; P2.DAT ; P3.DAT

IV – LEXIQUE

ABS absolu
ASC ascii
ATN atangente
BEEP onomatopée d'un petit bruit
CALL appeler
CDBL conversion en double
CHR$ character : caractère
CINT convert integer : conversion en entier
CIRCLE cercle
CLEAR mettre au propre, classer
CLOSE fermer
CLS clear screen : effacement écran
COLOR couleur
COM communication
COMMON partie commune
CONT continue : continuer
COS cosinus
CSNG convert single : conversion en simple
CRSLIN cursor line : ligne du curseur
DATA données
DATE$ date en chaîne de caractères
DEFFN define function : définit une fonction
DEFINT,DEFSNG
DEFDBL,DEFSTR : définit entier, simple, double et chaîne
DEFSEG définit segment
DEFUSR define user : définit utilisateur
DIM dimension
END fin
EOF end of file : fin de fichier
ERASE effacer
ERDEV,ERDEV$ error device : erreur d'appareil
ERR,ERRL error : erreur
ERROR erreur
EXP exponentielle
FIELD champ (de données ou caractères)
FILES fichiers
FIX entier
FOR...NEXT pour...suivant
GET obtenir
GOSUB...RETURN go subroutine...return : aller à sous programme... retour
GOTO aller à
HEX$ hexadécimal
IF...THEN(...ELSE)IF...GOTO : si...alors(...autrement) si...aller à
INKEY$ entrer aux touches
INP entrer au port
INPUT entrer au clavier
INPUT# entrer par fichier
INPUT$ entrer de chaîne de caractères
INSTR in string : recherche dans la chaîne de caractères
INT integer : entier
KEY clé ou touche du clavier
LEFT$ gauche
LEN lenght : longueur
LINE ligne
LINE INPUT entrée de ligne au clavier
LINE INPUT# entrée de ligne par fichier
LOCATE placer
LOG logarithme
LSET, RSET left set, right set : justification chaîne à gauche, à droite
MID$ middle : milieu
NAME nom
OCT$ octal
ON ERROR GOTO sur erreur aller à
ON...GOSUB, ON...GOTO sur... aller à sous-programme sur...aller à
ON KEY sur touche aller à
ON TIMER sur temps
OPEN ouvrir (un fichier, une ligne)
OPEN COM ouvrir une ligne de communication
OUT sortir sur port
PAINT colorer
POINT point
POS position
PRINT écrire
PRINT USING écrire suivant usage
PRINT, PRINT# USING
PSET, PRESET point set : mettre point
PUT mettre
RANDOMIZE hasard
READ lire
REM remarque
RESET remettre à zéro
RESTORE ramener
RESUME résumer (en erreur)
RETURN retour
RIGHT$ droite
RMDIR remove directory : effacer directory
RND random hasard
RUN courir
SAVE sauver
SCREEN écran
SGN sign : signe
SHELL enveloppe
SIN sinus
SOUND son
SPACE$ espace
SPC espace
SQR squarre : carré (racine carré)
STOP stop
STR$ string chaîne de caractères
STRING$ chaîne de caractères
SWAP échanger
SYSTEM système
TAB tabulation
TAN tangente
TIME$ temps
TRON, TROFF trace on, off
USR user : utilisateur
VAL valeur
VARPTR variable pointeur
VIEW voir
WAIT attendre
WHILE...WEND tandis que...alors
WIDTH largeur
WINDOW fenêtre
WRITE écrire
WRITE# écrire fichier

V – CODES ASCII

Représentation des caractères sur un octet (ASCII)

32	espaces	56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	"	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	`	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	-	69	E	93	]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	-	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	|
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g	127	