ulp

Institut
de Recherches
sur l'Enseignement
des Mathématiques
10, rue du Général Zimmer
67084 STRASBOURG CEDEX
Tel. (88) 61.48.20

UNIVERSITÉ LOUIS PASTEUR

INITIATION A L'ALGORITHMIQUE

ET A LA PROGRAMMATION

D. GUIN
J.P. IGOT
N. VOGEL

Animateurs
à
l'I.R.E.M.

DOSSIER Ø

NOTIONS DE BASE

1. ANTICIPATION ... OU L'ON IMAGINE UN AUTOMATE.

Imaginons un automate fonctionnant ainsi : on peut lui faire exécuter une succession d'actions simples ordonnées à condition qu'elle soit écrite sur une liste (appelons-la **programme**) placée définitivement dans la machine avant usage et rédigée à l'aide d'un vocabulaire limité, fixé [illegible] l'emploi de la machine, et respectant une grammaire précise.

Cet objet impitoyable sera donc insensible à la poésie... Mais at[illegible], ce n'est pas aussi catastrophique que vous ne l'imaginez.

Nous allons d'abord donner des exemples d'actions simples et proposer une écriture pour les désigner.

1.1. Ranger sous un nom donné (dans un tiroir)

- Un nombre ou **une chaine de caractères** (c'est-à-dire une suite de caractères, lettres, chiffres, signes de ponctuation... éventuellement vide) donnés dans le programme.
- Le résultat d'une opération (voir 1.2.)
- L'image d'une valeur par une fonction (voir 1.3.)

NOTATION :

* D ← "15 OCTOBRE" . indique qu'on range la chaine entre guillemets dans un tiroir portant le nom D

* N ← 152.Ø36 . indique qu'on range le nombre 152.Ø36 dans un tiroir de nom N.

REMARQUES :

- Nous noterons ␣ pour indiquer un blanc, un simple espace n'étant pas toujours lisible dans l'écriture manuscrite.
- Dans 152.Ø36 le point sépare la partie entière 152 de la partie décimale Ø36
- On note Ø pour distinguer zéro de la lettre "O".

1.2. Effectuer les opérations suivantes :

- Sur les nombres, les opérations arithmétiques classiques
- Sur les chaines, une opération appelée **concaténation**, qui consiste à mettre bout à bout deux chaines afin d'en obtenir une seule.

Exemples :

* R ← 15 * 7.3 - (8.4/3 + 4.5 * 9)
* DATE ← "15␣" | "OCTOBRE" (concaténation : DATE contiendra "15␣OCTOBRE")
* A ← 15.35
  CARRE ← A * A
  CUBE ← A * CARRE

REMARQUES :

- Les symboles * et / désignent respectivement la multiplication et la division
- Le symbole | désigne la concaténation
- Une opération peut porter sur des valeurs données explicitement (comme dans les deux premiers exemples) ou sur des contenus de tiroirs remplis précédemment (troisième exemple)

1.3. Transformer

- Un nombre par les fonctions mathématiques usuelles
- Une chaine de caractères par quelques fonctions permettant par exemple d'en extraire des sous-chaines.

Exemples :

* PI ← 3.141592
  A ← SIN(PI/4)

*DATE ← "SAMEDI␣15␣OCTOBRE"
MOIS ← SOUS CHAINE (DATE, 11,3) (MOIS contiendra "OCT")

REMARQUES :

- Les fonctions trigonométriques portent sur des mesures d'angles exprimées en radians
- Dans le deuxième exemple, 11 indique qu'on commence la sous-chaîne au onzième caractère de DATE, 3 indique qu'on prend trois caractères de DATE à partir de cet onzième.

1.4. Ranger, calculer ... des objets d'un troisième type, à valeur logique, appelés **booléens** (de Boole, mathématicien anglais du XIXe s).

- Un booléen peut avoir deux valeurs : VRAI ou FAUX
- Il peut être le résultat d'une comparaison portant sur des nombres ou des chaînes (à l'aide des relations < , > , = ...)
- Il peut être le résultat d'une opération logique sur des booléens (ET, OU, NON ...)

Exemples :

* T ← 8 < 3 (T contiendra FAUX)
* U ← "BONHEUR" < "MALHEUR" (U contiendra VRAI)
* V ← ("FEVRIER" > "AOUT") ET ("JUIN" > "JUILLET") (V contiendra VRAI)

REMARQUE :

Pour des chaînes, " < " signifie "placé avant dans l'ordre alphabétique"

1.5. Demander une donnée à l'utilisateur et donc recevoir un message introduit dans la machine et en lire le contenu. Cette donnée est alors rangée dans un tiroir sous un nom indiqué par le programme.

Notation - Exemples :

* LIRE J
* PI ← 3.141592
  LIRE R
  P ← 2 * PI * R
  A ← PI * R * R

REMARQUES :

- Le texte-programme ne permet pas de savoir quelle sera la valeur contenue dans J ou dans R.
- "LIRE" permet d'écrire un programme qui fournit un résultat à une famille de problèmes (deuxième exemple)
- Le texte donné en réponse à "LIRE" ne subit pas les contraintes imposéesà la rédaction du programme, et ici les poètes en herbe peuvent donc généralement donner libre cours à leur imagination.

1.6. Ecrire un nombre ou une chaîne de caractères et le "sortir" de la machine.

Notation :

* ECRIRE "TO␣BE␣OR␣NOT␣TO␣BE..."
* ECRIRE 1 * 9, 2 * 9, 3 * 9, 4 * 9, 5 * 9

REMARQUE :

Dans le deuxième exemple, on résume les cinq lignes ECRIRE 1 * 9 ... ECRIRE 5 * 9 par la notation ECRIRE 1 * 9, ..., 5 * 9

Ces différentes actions seront étudiées de façon plus précise dans le DOSSIER 1.

On peut également répéter une partie d'un programme (il faut bien sûr que le programme indique quoi et sous quelle condition) ou exécuter des actions conditionnelles selon qu'un booléen est VRAI ou FAUX. Ces possibilités seront étudiées dans les DOSSIERS 2 et 3.

2. OU L'ON VOIT QUE LA REALITE DEPASSE LA FICTION.

Vous avez déjà compris que de tels automates existent.., vous en avez rencontrés !

Ils s'appellent **ordinateurs** -vous l'avez deviné- Mais attention ils n'ont aucun pouvoir magique et vous vous rendrez bien vite compte qu'ils ne savent pas lire entre les lignes.

2.1. L'ordinateur

Afin de pouvoir réaliser les actions 1.1, l'ordinateur doit pouvoir stocker des informations.

Afin de réaliser 1.2, 1.3, 1.4, il doit pouvoir traiter des informations.

Pour 1.5, il doit pouvoir recevoir des informations, et pour 1.6 communiquer des résultats.

Résumons ainsi : un ordinateur est une machine qui peut recevoir des informations pour les stocker, les traiter et qui peut communiquer des résultats.

2.2. Structure du micro-ordinateur

Un micro-ordinateur se présente généralement extérieurement comme un clavier (genre machine à écrire) accompagné d'un écran vidéo (ou d'une télé) et éventuellement d'un lecteur de cassettes (souvent un magnéto ordinaire) ou de disquettes (non, un tourne-disque ordinaire ne convient pas !)

En fait c'est un peu plus compliqué : le micro-ordinateur se compose des principaux éléments suivants :

permettant de donner des informations → - le clavier
- un lecteur de cassettes ou de disquettes (facultatif)

permettant de stocker des informations → - mémoire centrale

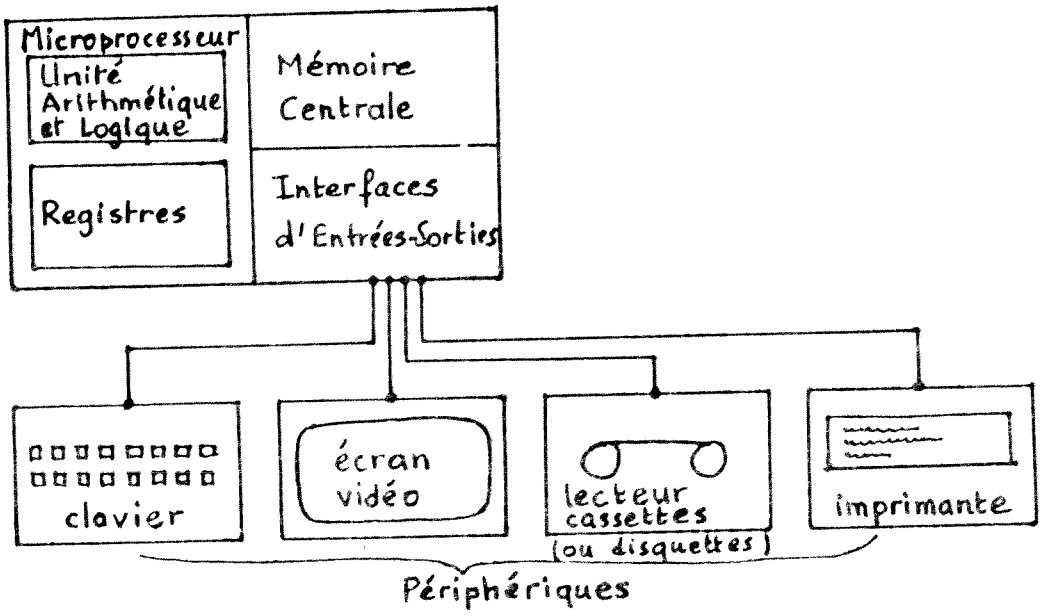
traiter des informations → - microprocesseur contenant l'unité arithmétique et logique

communiquer des résultats → - écran
- imprimante (facultatif)

Ces différents éléments sont reliés entre eux par des fils, appelés bus (Eh oui ! c'est un moyen de transport).

Cet ensemble constitue le **matériel.**

Schéma de l'organisation d'un micro-ordinateur



2.3. Principes de communication

2.3.1 Langages de programmation

En 1 nous avons évoqué un vocabulaire et une grammaire précise.

Un ensemble formé d'un vocabulaire et d'une grammaire fixant les règles de la rédaction d'un programme, s'appelle un **langage de programmation.**

En 1 nous avons proposé un vocabulaire qui ne dépend d'aucune contrainte technique. Ce n'est pas tout à fait un langage de programmation, car vous pourriez par exemple remplacer le mot LIRE par le mot Entrer ou ECRIRE par Affiche sans aucune conséquence sur la correction de la rédaction.

Mais nous l'avons choisi assez évocateur pour qu'un programme écrit en ces termes soit clair et lisible. Cette façon d'écrire est un outil de travail qui présente un double avantage :

- Elle reste suffisamment proche de la langue naturelle pour être claire
- Elle est suffisamment précise pour pouvoir se traduire immédiatement dans toute une famille de langages de programmation.

Un langage de programmation possède une grammaire beaucoup plus rigide et chaque faute de frappe provoque une erreur de syntaxe. Parmi les langages les plus répandus, citons : BASIC (Beginner's All - Purpose Symbolic Instruction Code, mis au point dans les années 6Ø) LSE (Langage Symbolique d'Enseignement) PASCAL, COBOL, FORTRAN, APL, PL1, ASSEMBLEUR ...

2.3.2 Programmes d'exploitation

Un **programme (d'exploitation)** est une suite ordonnée d'ordres d'actions simples -appelés **instructions**- écrits dans un langage donné

Le système et les programmes d'exploitation constituent le **logiciel** de l'ordinateur.

Un **programme BASIC** (ou LSE) se compose d'une suite de lignes numérotées contenant une ou plusieurs instructions.

Exemples :

Programme en BASIC Microsoft

```
10 REM CERCLE
20 LET PI=3.14159
30 INPUT "RAYON DU CERCLE ";R
40 LET P=2*PI*R:LET A=PI*R*R
50 PRINT P;A
60 END
```

Exécution sur l'écran

```
RAYON DU CERCLE ? 5
 31.4159   78.5398
```

Traduction LSE

```
* CERCLE
PI_3.141592
LIRE['RAYON DU CERCLE ']R
P_2*PI*R:A_PI*R*R
AFFICHER P,A
TERMINER
```

Traduction PASCAL

```
PROGRAM CERCLE;
CONST  PI=3.141592;
VAR R,P,A:REAL;
BEGIN
  WRITE ('RAYON DU CERCLE');READ (R);
  P:=2*PI*R;A:=PI*R*R;
  WRITELN(P,A)
END.
```

Une **ligne** de programme commence par un numéro et peut contenir jusqu'à 255 caractères. Sa frappe se termine par la frappe d'une touche spéciale enregistrant la ligne dans la mémoire centrale de l'ordinateur, appelée touche de validation (suivant les appareils, C.R (Carriage Return), R.C (Retour Chariot), RETOUR, RETURN, VALIDE, ENTER ...)

Les numéros de lignes doivent correspondre (sauf cas particulier - voir DOSSIERS 2 et 3) à l'ordre dans lequel les instructions doivent être exécutées. (Habituellement, on les numérote de 10 en 10, ce qui permet facilement d'intercaler des numéros en cas de modification). S'il y a plusieurs instructions sur une même ligne de programme, elles doivent être séparées par ":" (";" en LSE).

REMARQUE :

Ne pas confondre ligne de programme (jusqu'a 255 caractères) et ligne d'écran (souvent 80 caractères). Une ligne de programme peut contenir plusieurs lignes d'écran (un peu plus de trois) et le curseur de l'écran se déplace automatiquement sur la ligne suivante lorsqu'il arrive à la fin d'une ligne (de l'écran).

Afin de décrire les instructions à effectuer par l'ordinateur indépendamment d'un langage de programmation spécifié, nous introduirons la notion d'algorithme, se distinguant du programme proprement dit. Par exemple :

	* CERCLE
R(réel) : rayon du cercle	ECRIRE "RAYON DU CERCLE"
PI (réel) : 3.14159	LIRE R P←2 * PI * R
P,A (réels) : périmètre et aire du cercle	A←PI * R * R ECRIRE P,A

Nous favoriserons ainsi la séparation du travail de conception du programme, de sa réalisation dans un langage donné.

2.3.3 Systèmes d'exploitation

Généralement, lorsqu'on met un ordinateur sous tension, on ne peut pas y entrer directement un programme écrit dans un langage quelconque. On passe par les étapes suivantes :

- On charge (à partir d'une cassette ou d'une disquette) dans la mémoire de la machine, un premier programme, appelé **système d'exploitation** qui dirige les tâches de l'ordinateur (par exemple, copier une disquette)
- On charge ensuite un deuxième programme, qui est un "traducteur" (**interpreteur ou compilateur**) permettant à l'ordinateur de traduire un texte écrit dans un langage usuel en code compréhensible et exécutable par la machine.

REMARQUES :

- L'ensemble de ces deux programmes est appelé **système**
- Lorsqu'on coupe le courant, une seule partie de la mémoire (appelée mémoire morte ou R.O.M.) est conservée. Pour les ordinateurs fonctionnant comme on vient de l'indiquer, le système est chargé dans une partie de la mémoire appelée mémoire vive (ou R.A.M) et donc effacé lorsqu'on coupe le courant (mais heureusement conservé sur la disquette ou la cassette). Certains ordinateurs (TRS 80 par exemple) possèdent un système en mémoire morte qui est donc disponible dès qu'on met l'appareil sous tension.
- Un ordinateur peut généralement recevoir plusieurs langages de programmation. Il suffit de charger le système adapté, qui traduit le langage de l'utilisateur dans le langage de la machine, qui lui, est unique.

Le système chargé, on peut demander à l'ordinateur différentes tâches, choisies en général à l'aide d'une **commande** (voir annexe Ø).

Par exemple :

- Chercher un programme existant sur une disquette
- Exécuter un programme
- Copier le texte de ce programme sur l'écran
- Construire une nouvelle liste-programme dans la mémoire de la machine, en tapant le texte de ce programme au clavier. Cette dernière tâche, qui s'appelle **éditer** un programme, est celle qui est en général sélectionnée par le système en l'absence d'une autre commande.

Les ordinateurs personnels disposent généralement d'un système d'exploitation "personnalisé", tenant compte de leur configuration et usage particulier (p. ex. fonctions graphiques, manche à balai, sortie musique etc.) Les équipements professionnels fonctionnent sous des systèmes relativement normalisés qui permettent une certaine compatibilité des progremmes notamment des interpréteurs, éditeurs de texte, tableurs, etc...) on parle alors de **logiciels de base**.

Quelques "grands" systèmes se partagent le marché :

. Control Program for Micro computers (CP/M) de Digital Research

. MS-DOS (Disk Operating System de Microsoft)

. UNIX ...

Des logiciels comme WORDSTAR, MULTIPLAN, CALCSTAR, DBASE III, MBASIC,

TURBOPASCAL etc... sont alors disponibles et compatibles sous ces divers systèmes d'exploitation.

On trouvera en annexe, quelques rudiment sur **CP/M et BASIC Microsoft**, qui sont disponibles sur les microordinateurs T07 et analogues, équipant les établissements scolaires.

ANNEXES AU DOSSIER 0

1. COMMANDES (Système CP/M, par exemple sur TO7)

Ce sont les ordres pour sélectionner l'une des fonctions possibles de l'ordinateur. (Pour la mise en marche de l'ordinateur, voir le paragraphe 2 ou 3).
Les principales commandes sont les suivantes :

- **Consulter le catalogue** des programmes présents sur une disquette :

taper DIR (CR) ou DIR B : (CR) retour de chariot

fournira par exemple :

CERCLE . BAS (programme BASIC)
MBASIC . COM (Interpréteur BASIC)
TEST . COM (programme en langage machine)

- **Charger un programme** existant sur une disquette, dans la mémoire centrale de l'ordinateur.

Taper TEST (CR)
si l'on veut utiliser le programme TEST . COM écrit dans le langage interne de l'ordinateur.
Le programme s'exécute automatiquement

ou taper MBASIC (CR)
chargement de l'interpréteur BASIC qui prendra le contrôle des opérations (on dira que l'ordinateur travaille "sous BASIC"). Ensuite,

LOAD "CERCLE"
CR

permettra de charger le programme BASIC CERCLE . BAS (voir 2.3.2)

- **Exécuter un programme**

RUN (CR) permet alors de l'exécuter (c.a.d demander le rayon du cercle puis effectuer les calculs du périmètre et de l'aire, qui s'afficheront sur l'écran.

- **Voir la liste** des lignes d'instructions composant un programme

sur écran LIST

LIST 1 - 1ØØ (programme de la ligne 1 à la ligne 1ØØ)

LIST 5Ø - (de la ligne 5Ø à la fin)

LIST - 1ØØ (du début à la ligne 1ØØ)

LIST 6Ø (la ligne 6Ø)

sur imprimante (assurez-vous qu'elle est branchée et en marche)

LLIST avec les mêmes variantes que LIST

- **Ranger un programme** existant dans la mémoire centrale de l'ordinateur sur une disquette

SAVE "ESSAI" CR

le programme ESSAI sera rangé sous le nom ESSAI . BAS

- **Interrompre** l'exécution d'un programme ... très souvent parce qu'il ne s'arrête pas comme on l'avait espéré ...

appuyer simultanément sur les touches **CTRL et C**

- En outre, après avoir quitté le BASIC (QUIT$_{CR}$), le système CP/M offre la possibilité de gérer ses fichiers :

- effacer un fichier : ERA nom CR
- modifier son nom : REN nom = ancien nom (CR)
- copier un fichier : PIP nom destinataire = nom source (CR)

2. UTILISATION DU MICRAL 80-22 EN BASIC MICROSOFT

1. Démarrage

- mettre la prise de courant
- mettre sous tension (sur le côté gauche)
- mettre la disquette GBASIC (logement gauche) étiquette vers l'avant à droite)
- taper sur [VALIDE] . Attendre le chargement ; rouvrir la porte de disquette : l'écran affiche O.K. On peut travailler en BASIC.

2. Fonctions particulières

CTRL-C	interrompt un exécution
CTRL-H	efface le dernier caractère
CTRL-O	Interrompt un listage sur écran (CTRL-O le reprend
CTRL-S	suspend une exécution (CTRL-Q la reprend)
CTRL-U	efface la ligne en cours de frappe
LLIST	listing sur imprimante (après mise sous tension (gauche) et sélection [SEL]
LPRINT	écrit sur l'imprimante

3. SAUVEGARDE D'UN PROGRAMME SUR DISQUETTE

- mettre la disquette MICRAL 1 à droite
- faire SAVE "nom" [VALIDE]

4. RECHERCHE PROGRAMME

FILES	donne le catalogue
LOAD "nom"	charge le programme en mémoire
LOAD "nom",R	charge et exécute

5. CORRECTION D'UNE LIGNE DE BASIC

- si la ligne n'est pas trop longue, on préférera écrire une nouvelle ligne BASIC en utilisant le numéro de la ligne erronée. La nouvelle ligne se substitue alors à l'ancienne.

- La frappe d'un numéro de ligne suivi d'aucun texte permet d'effacer la ligne correspondante du programme.
- Pour des corrections sur la ligne existante :

* Frapper EDDIT n° ligne
* la barre d'espacement permet d'avancer sur la ligne corrigée
* I "texte" [ESC] insère "texte" à l'endroit indiqué par le curseur
* [←] efface le caractère
* n [D] efface n caractères à droite du curseur
* n [C] "car" remplace n caractères à droites par les n frappes suivant la frappe de [C]
* [VALIDE] valide l'ensemble des corrections sur ligne
* n [S] "car" recherche la n^e occurence de "car" et s'y positionne
* K "car" recherche avec effacement jusqu'à la n^e occurence de "car"
* X insertion en fin de ligne
* A reprend les corrections sur la ligne éditée
* H efface à droite du curseur et attend une insertion.

DOSSIER 1

INSTRUCTIONS ELEMENTAIRES - ALGORITHMES

1. Des précisions sur les instructions évoquées dans le Dossier Ø. Premiers Algorithmes.

1.1 Un programme est formé d'une liste d'instructions rédigée dans un langage approprié. La démarche préparatoire consiste à définir la succession des opérations à effectuer **(algorithmes)** que l'on pourra présenter sous la forme suivante :

Exemple d'algorithme

Table	Programme
	***PRIX**
PHT(nombre) : prix hors taxe TTVA(nombre) : taux TVA	. LIRE PHT, TTVA . ECRIRE "PRIX TTC :" . ECRIRE PHT * (1 + TTVA/1ØØ)

La partie droite regroupe l'énoncé formel des instructions. Les données manipulées sont décrites de façon informelle, dans un **lexique** que l'on pourra compléter au fur et à mesure dans la partie gauche.

1.2 Dans un programme, on peut manipuler différentes sortes de données, appelées **Types** de données. (Ce ne sont pas les mêmes pour tous les langages). En général, on dispose de :

- Nombres **entiers** (entiers signés compris entre -32768 et 32767 (-2^{15} et $2^{15}-1$))
- Nombres **réels** (nombre décimal signé, en notation décimale ou sous forme d'un nombre décimal multiplié par une puissance de 1Ø)
- **Chaînes** de caractères, (en général Ø à 255 caractères, lettres, chiffres, signes de ponctuation...)
- **Booléens** (voir Dossier Ø)
- **Tableaux** de l'un de ces types de données (voir Dossier 3)

Dans certains langages de programmation, tous les noms et les types correspondants doivent être donnés à l'ordinateur (PASCAL, COBOL), dans une partie appelée déclarative, alors que pour d'autres langages (BASIC) cette partie est très réduite et ne concerne que quelques types précis.

. Lorsqu'on manipule une donnée explicite, on dit qu'il s'agit d'une **constante.**

Exemples : 152.Ø36 est une constante "réelle"
"15 ␣ OCTOBRE" est une constante chaîne.
" " représente une constante chaîne, c'est la chaîne vide (à ne pas confondre avec le caractère espace (" ␣ ")

. Lorsqu'une information à une valeur susceptible d'évoluer au cours du programme, ou d'être émise ou reçue de l'extérieur, on parle de **variable** - symbole identifiant une zône de la mémoire où les valeurs successives sont conservées.

Exemples :

Variable	**Valeur**	**Type**
A	"A"	Chaîne
A	"MARDI ␣ 11"	Chaîne
JOUR	15	Nombre

Attention : Ne pas confondre identificateur avec constante chaîne (voir aussi 1.5. exemple 1). Comme en mathématique, le symbole utilisé désignera aussi bien la zone mémoire que son contenu.

1.3 On appelle **affectation** l'instruction qui consiste à donner à une variable d'un certain type une valeur du même type (Dossier Ø - 1.1) La valeur est donnée sous la forme d'une **expression** c'est-à-dire :

- une variable déjà affectée (C← A dans l'exemple 3))
- une constante (A ← - 5 dans l'exemple 1))
- une **opération** portant sur des constantes, des variables déjà affectées ou plus généralement des expressions.
- la valeur d'une **fonction** appliquée à une constante, une variable ou plus généralement une expression
- le résultat (booléen) d'une comparaison à l'aide d'une **relation** portant sur des expressions, nombres ou chaînes.

Opérations (Voir exemples 1), 2), 3)) :

- sur les nombres +, -, *, /, ↑ (exponentiation)
- sur les chaînes concaténation (notée |)
- sur les booléens ET, OU, NON

Fonctions (voir exemples 1), 2), 4)) :

- **sur les nombres** (résultats numériques)

SIN( ), COS( ), LOG( ), EXP( ), ABS( ), RAC( )

(sinus) (cosinus) (Log. népérien) (exponentielle) (valeur absolue) (racine carré)

ENT( ), ATG( ), TG( )

(partie entière) (arc tangente) (tangente)

- **sur les chaînes** (résultat numérique)

LRG( ) donne la longueur de la chaîne, c.a.d. le nombre de caractères la composant (résultat chaîne)

SCH(, ,) fonctions à trois arguments

- nombre de caractères
- point de départ
- chaîne d'origine

ex : SCH ("BONJOUR", 3,2) représente "NJ"

Relations (voir exemple 4)) :

< , > , =, < = (inférieur ou égal), > =,
< > (différent)

Remarque importante :

On peut affecter à une variable le résultat d'une opération ou la valeur d'une fonction portant sur le contenu actuel de la zone. Le calcul est effectué lorsqu'il est terminé, le résultat prend la place de l'ancien contenu.
(voir exemple 1), 3), et 4));

1.4 Exemples d'affectations

	* ZONE OCCUPEE
1) A (nombre)	. A ← - 5 . A ← A + 2 * A * A → A contiendra 45 . A ← ENT(RAC(A)) → A contiendra 6

	* REVES
2) A(chaîne) B,C (chaîne) : modifiées à partir de A	. A ← "RELEVER" . B ← SCH(A,1,2) | SCH(A,5,3) . C ← "NOUS␣" | SCH(B,1,3) | "ONS" ⟶ que contiennent B et C ?

3)	* VOLTIGE
A,B(nombre)	. A ← 3 . B ← 5 . A ← A + A . A ← A + A . C ← A . A ← B . B ← C → que contiennent A,B et C ?

4)	* ENCORE
A,B(chaîne) T,U(booléen)	. A ← "ASSEZ␣!" . B ← "PARDON␣?" . T ← A < B . A ← SCH(A,2,LGR(A) - 1) . U ← A < B → que contiennent U et V ?

1.4 On appelle instruction **d'entrée** l'instruction notée **LIRE** A qui

- stoppe l'exécution d'un programme
- attend la frappe d'une constante du type de A terminée par la frappe de la touche de validation
- range cette constante dans la zone mémoire associée à la variable A (Dossier Ø - 1.5)

Exemples :

1)	* INITIALES
N(chaîne) : nom P(chaîne) : prénom I(chaîne) : initiales	. LIRE N,P . I ← SCH(P,1,1) |"."| SCH(N,1,1)

2)	* DISCRIMINATION
A,B,C(nombre) : coefficients d'un trinôme D(nombre) : discriminant TEST(booléen) : indique si D ⩾ Ø	. LIRE A,B,C . D ← B * B - 4 * A * C . TEST← D > = Ø

1.5 On appelle instruction de **sortie** l'instruction notée **ECRIRE** qui écrit un résultat sur l'écran ou sur l'imprimante (DOSSIER Ø - 1.6).

Exemples :

1)

T(chaîne)
SOS (nombre)

*** FIN**

```
.ECRIRE "BONJOUR␣TOUT␣LE␣MONDE.."
. T← "JE CRAQUE␣!"
. ECRIRE T,T,T | "!!"
. SOS←1983
. ECRIRE "SOS",SOS
```

(Ne pas confondre la constante chaîne "SOS" avec l'identificateur de variable SOS)

2)

A,B(booléen) :
le couple (A,B)
prend toutes les
valeurs possibles
pour A VRAI ou FAUX
et B VRAI ou FAUX

*** IMPLICATION**

```
. A ← VRAI
. B ← VRAI
. ECRIRE A,B,NON(A) ou B
. B ← FAUX
. ECRIRE A,B, NON(A) ou B
. A ← FAUX
. B ← VRAI
. ECRIRE A,B, NON(A) OU B
. B ← FAUX
. ECRIRE A,B, NON(A) ou B
```

(la valeur logique de NON(A) ou B est celle de l'implication A ⇒ B).

1.6 On appelle **bloc** une suite d'une ou plusieurs instructions. En général, on donne un nom à un bloc. Le même bloc peut être utilisé en plusieurs endroits d'un programme, en rappelant simplement son nom s'il a déjà été défini.
Un programme est lui-même un bloc.
Tous les programmes écrits jusqu'ici sont constitués d'un seul bloc qui se confond donc avec le programme.

Mais nous verrons des exemples où la situation est différente dans les DOSSIERS 2 et 3. Par exemple, IMPLICATION donné en 1.5 pourrait s'écrire plus simplement en considérant comme bloc une partie de ce programme qui se retrouve plus d'une fois.
On appelle bloc **séquentiel** une suite d'instructions où chacune d'elles doit être exécutée exactement une fois et dans l'ordre où elles figurent dans le programme.
Tous les exemples du DOSSIER 1 sont de ce genre. On verra d'autres structures dans les DOSSIERS 2 et 3.

2 Et maintenant un peu de BASIC pour apprendre à parler aux machines

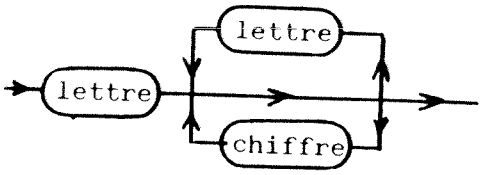
Nous allons reprendre le plan et les exemples de 1 et en donner l'écriture en BASIC. Pour cela, nous utiliserons des diagrammes syntaxiques dont la lecture vous deviendra rapidement familière.

2.1 Nous avons déjà indiqué que l'on déclare rarement les types en BASIC, sauf pour les tableaux, mais l'identificateur d'une variable permet d'un reconnaître le type.

. Identificateur d'un réel :

à lire ainsi :

- un identificateur commence obligatoirement par une lettre
- ensuite, on peut parcourir le schéma de différentes façons, pourvu que ce soit dans le sens des flèches. Après la lettre on trouve donc une suite de caractères, lettres ou chiffres, ou rien.

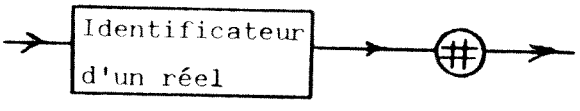


- En général, la longueur d'un identificateur est limitée à un certain nombre de caractères (différents suivant les machines : 40 sur MICRAL mais seulement 2 significatifs sur TRS 80, c'est-à-dire p.ex. que TAUX et TAXES sont possibles, mais désignent la même variable).
- Un certain nombre de mots, appelés **mots réservés**, sont exclus comme identificateurs. Ce sont les mots du langage BASIC (LET, END, PRINT...).

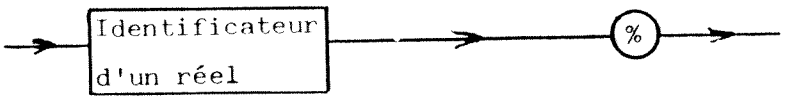
. Dans certains BASIC, il existe des "**réels double précision**" (les calculs et l'affichage de ces nombres se font avec plus de chiffres).

. Identificateur d'un réel double précision :

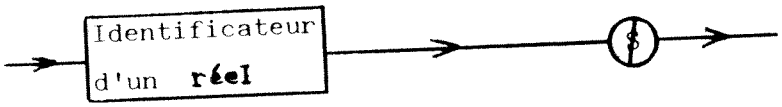
(dans les rectangles d'un diagramme syntaxique, on met des noms d'objets dont la syntaxe a été définie).



. Identificateur d'un entier



. Identificateur d'une chaîne



. Le type booléen n'est pas un type particulier en BASIC. Une variable booléenne a un identificateur de nombre.
Souvent la valeur **VRAI** correspond à - 1 et la valeur **FAUX** à Ø.
On peut appliquer à ces valeurs les opérations sur les booléens et les obtenir comme résultats de comparaisons.

2.2 Nous avons expliqué en 1.3 ce qu'est une expression. Nous n'en détaillerons pas la syntaxe, qui est assez naturelle.
Précisons simplement les symboles opératoires et les fonctions standard du BASIC :

Opérations : (voir exemples CALCUL, PUISSANCES, CONCATENATION... p.1.8 et 1.9)

- nombres : +, -, *, /, ∧ (exponentiation) (ou ↑ suivant les claviers)
- chaînes : concaténation notée +
- booléens : AND, OR, XOR (ou exclusif), NOT

Fonctions : (voir exemples TRIGO, SOUS CHAINE, REVES, ZONE OCCUPEE ... p. 1.9 et 1.1Ø)

- sur les nombres (résultat nombre) :
  SIN, COS, TAN, ATN, LOG, EXP, INT (partie entière) ABS, SQR (racine carrée)

  RND (générateur de nombres pseudo-aléatoires - Voir annexe 1)
- sur les chaînes (résultat chaîne) :
  MID$(chaîne, indice de départ, nombre de caractères)
  (SCH - Voir 1.3)

LEFT$ (chaîne, nombre de caractères)
(donne : partie gauche d'une chaîne, nombre de caractères indiqué)

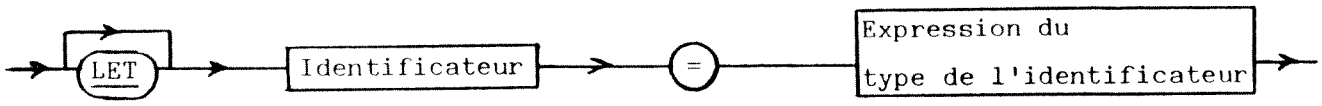
RIGHT$ (chaîne, nombre de caractères)
(donne : partie droite d'une chaîne, avec le nombre de caractères précisé)

- sur les chaînes (résultat nombre)
  LEN(chaîne) : donne la longueur de la chaîne.

Comparaisons :

< , > , = , < = , > = < >

Instruction d'affectation :



. Les mots soulignés dans un diagramme syntaxique sont les mots réservés.

. LET est facultatif

Exemples de programme : (DOSSIER Ø - 1.2 et 1.3 ; DOSSIER 1 - 1.3)

Programme BASIC — **Résultat de l'exécution** (sur l'écran)

```
* 10 REM CALCUL
  20 LET R=15*7.3-(8.4/3+4.5*9)                    66.2
  30 PRINT R
  40 END
```

Remarque : - Les lignes sont numérotées (voir dossier Ø, 2.3.3)
- REM permet de donner le nom du programme.
  La ligne REM... est ignorée à l'exécution
- L'instruction PRINT de la ligne 3Ø est l'instruction de sortie (ECRIRE) - Voir 2.4
- L'instruction END termine tout bloc programme.

```
10 REM PUISSANCES
20 LET A=15.35
30 LET CARRE=A*A
40 LET CUBE=A*CARRE
50 PRINT A;CARRE;CUBE
60 END
```

15.35 235.623 3616.81

```
10 REM CONCATENATION
20 LET DATE$="15 "+"OCTOBRE"
30 PRINT DATE$
40 END
```

15 OCTOBRE

```
10 REM TRIGO
20 LET PI=3.14159
30 LET A=SIN(PI/4)
40 PRINT A
50 END
```

.707107

```
10 REM SOUS CHAINE
20 LET DATE$="SAMEDI 15 OCTOBRE"
30 LET MOIS$=MID$(DATE$,11,3)
40 PRINT MOIS$
50 END
```

OCT

```
10 REM SURPRISES
20 LET T=8<3
30 LET U="BONHEUR"<"MALHEUR"
40 LET V=("FEVRIER">"AOUT") AND ("JUIN">"JUILLET")
50 PRINT T;U;V
60 END
```

0 -1 -1

```
10 REM ZONE OCCUPEE
20 LET A=-5
30 LET A=A+2*A*A
40 PRINT A
50 LET A=INT(SQR(A))
60 PRINT A
70 END
```

45
6

*
```
10 REM REVES
20 LET A$="RELEVER"
30 LET B$=MID$(A$,1,2)+MID$(A$,5,3)
40 LET C$="NOUS "+MID$(B$,1,3)+"ONS"
50 PRINT B$
60 PRINT C$
70 END
```

```
REVER
NOUS REVONS
```

Remarque : Variante pour les lignes 30 et 40

```
10 REM REVES
20 LET A$="RELEVER"
30 LET B$=LEFT$(A$,2)+RIGHT$(A$,3)
40 LET C$="NOUS "+LEFT$(B$,3)+"ONS"
50 PRINT B$
60 PRINT C$
70 END
```

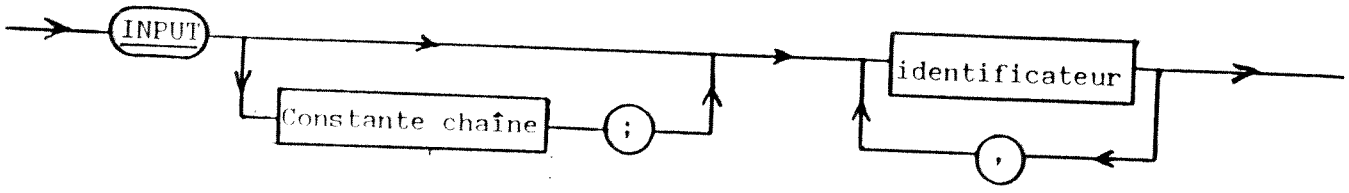
*
```
10 REM VOLTIGES
20 LET A=3:LET B=5
30 LET A=A+A:LET A=A+A
40 PRINT A;B
50 LET C=A:LET A=B:LET B=C
60 PRINT A;B;C
70 END
```

```
 12  5
 5  12  12
```

*
```
10 REM ENCORE
20 LET A$="ASSEZ!":LET B$="PARDON?"
30 LET T=A$<B$
40 LET A$=RIGHT$(A$,LEN(A$)-1)
50 PRINT A$
60 LET U=A$<B$
70 PRINT T;U
80 END
```

```
SSEZ!
-1  0
```

2.3 Instruction d'entrée



- On voit que BASIC permet d'insérer un commentaire (constante chaîne) avant l'identificateur.
- Ce commentaire est affiché avant l'interruption du programme et permet donc au programmeur d'indiquer à l'utilisateur quel genre de réponse il doit donner. S'il n'y a pas de commentaire, seul un point d'interrogation est affiché.
- Cette instruction peut demander l'entrée de plusieurs valeurs. Les réponses doivent alors être séparées par des virgules.
- Les réponses doivent être des constantes du type des identificateurs, par exemple 2.4 pour un nombre, "ZUT !" pour une chaîne.
  Quand on donne une chaîne, on peut omettre les guillemets, sauf lorsqu'elle contient une virgule ou quelquefois des blancs.

Exemples : (DOSSIER Ø - 1.5 ; DOSSIER 1 - 1.4)

Programme

```
*  10 REM CERCLE
   20 LET PI=3.14159
   30 INPUT "RAYON DU CERCLE ";R
   40 LET P=2*PI*R:LET A=PI*R*R
   50 PRINT P;A
   60 END
```

Exécution sur l'écran

```
RAYON DU CERCLE ? 5
 31.4159   78.5398
```

Programme

```
10 REM INITIALES
20 INPUT "DONNEZ VOTRE PRENOM ET VOTRE NOM SEPARES PAR UNE VIRGULE ";P$,N$
30 LET I$=LEFT$(P$,1)+"."+LEFT$(N$,1)+"."
40 PRINT I$
50 END
```

Exécution sur l'écran

```
DONNEZ VOTRE PRENOM ET VOTRE NOM SEPARES PAR UNE VIRGULE ? "VICTOR","HUGO"
V.H.
```

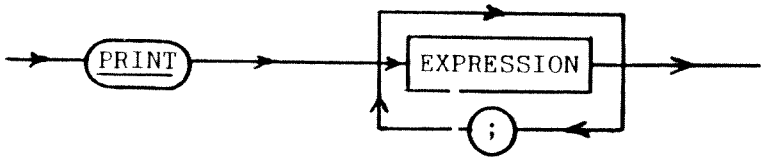
interruption de l'exécution

réponse de l'utilisateur

```
10 REM DISCRIMINATION
20 INPUT "COEFFICIENTS DU TRINOME ";A,B,C
30 LET D=B*B-4*A*C
40 LET TEST=D>=0
50 PRINT D;TEST
60 END
```

```
COEFFICIENTS DU TRINOME ? 2,4,-5
 56 -1
```

2.4 Instruction de sortie sur écran (voir "expression" en 1.3 et 2.2)



- Si on fait écrire plusieurs expressions séparées par des ";" elles seront toutes écrites côte à côte. Du point de vue du résultat, PRINT A$; B$ est équivalent à PRINT A$ + B$.
- Si on met ";" après la dernière expression, la prochaine instruction d'écriture affiche à côté de cette dernière expression.
- Si on termine sans ";" , la prochaine instruction d'écriture affiche au début de la ligne suivante.

Comparer :

```
10 PRINT "BON";"JOUR ":PRINT "MADAME"
```

```
BONJOUR
MADAME
```

```
10 PRINT "BON";"JOUR ";:PRINT "MADAME"
```

```
BONJOUR MADAME
```

- Ne pas confondre PRINT A$; B$ qui est une seule instruction, avec PRINT A$; PRINT B$ qui se compose de deux instructions, séparées par conséquent par ":". Mais ces deux instructions sont équivalentes.
- Pour écrire sur l'imprimante ou pour plus de précisions sur les possibilités d'écriture non évoquées ici, voir annexe 1.

Exemples : (DOSSIER Ø - 1.6 ; DOSSIER 1 - 1.5)

```
10 REM N'IMPORTE QUOI
20 PRINT "TO BE OR NOT TO BE..."
30 PRINT 1*9;2*9;3*9;4*9;5*9
40 END
```

```
TO BE OR NOT TO BE...
 9  18  27  36  45
```

Remarque : sur beaucoup de machines, tout nombre est affiché entre deux blancs.

```
10 REM PRIX
20 INPUT "PRIX HORS TAXES, TAUX TVA ";PHT,TTVA
30 PRINT "PRIX TTC : "
40 PRINT PHT*(1+TTVA/100)
50 END
```

```
PRIX HORS TAXES, TAUX TVA ? 568,33
PRIX TTC :
 755.44
```

```
10 REM FIN
20 PRINT "BONJOUR TOUT LE MONDE ..."
30 LET T$="JE CRAQUE ! "
40 PRINT T$;T$;T$+"!!!"
50 LET SOS=1983
60 PRINT "SOS";SOS
70 END
```

```
BONJOUR TOUT LE MONDE ...
JE CRAQUE ! JE CRAQUE ! JE CRAQUE ! !!!
SOS 1983
```

A titre d'exercice, traduire IMPLICATION (DOSSIER 1 - 1.5) en BASIC

3 Traduction LSE et PASCAL des exemples

```
1 * CALCUL
10 R_15*7.3-(8.4/3+4.5*9)
20 AFFICHER R
30 TERMINER
```

```
PROGRAM CALCUL;
VAR R:REAL;
BEGIN
  R:=15*7.3-(8.4/3+4.5*9);
  WRITELN(R)
END.
```

```
1 * PUISSANCES
10 A_15.35
20 CARRE_A*A
30 CUBE_A*CARRE
40 AFFICHER A,CARRE,CUBE
50 TERMINER
```

```
PROGRAM PUISSANCES;
VAR A,CARRE,CUBE:REAL;
BEGIN
  A:=15.35;
  CARRE:=A*A;
  CUBE:=A*CARRE;
  WRITELN(A,CARRE,CUBE)
END.
```

```
1 * CONCATENATION
10 CHAINE DATE
20 DATE_'15 '!'OCTOBRE'
30 AFFICHER DATE
40 TERMINER
```

```
PROGRAM CONCATEN;
VAR DATE:STRING;
BEGIN
  DATE:=CONCAT('15 ','OCTOBRE ');
  WRITELN(DATE)
END.
```

```
1 * TRIGO
10 PI_3.141592
20 A_SIN(PI/4)
30 AFFICHER A
40 TERMINER
```

```
PROGRAM TRIGO;
(* Commentaire eventuel *)
CONST PI=3.141592;
VAR A:REAL;
BEGIN
  A:=SIN(PI/4);
  WRITELN(A)
END.
```

```
1 * SOUS CHAINE
10 CHAINE DATE,MOIS
20 DATE_'SAMEDI 15 OCTOBRE'
30 MOIS_SCH(DATE,11,3)
40 AFFICHER MOIS
50 TERMINER
```

```
PROGRAM SCHAINE;
VAR DATE,MOIS:STRING;
BEGIN
  DATE:='SAMEDI 15 OCTOBRE';
  MOIS:=COPY(DATE,11,3);
  WRITELN(MOIS)
END.
(* On peut aussi utiliser le

   type ARRAY[  ] OF CHAR *)
```

```
 * SURPRISES
0 BOOLEEN T,U,V
20 T_8<3
30 U_'BONHEUR'<'MALHEUR'
40 V_'FEVRIER'>'AOUT' ET 'JUIN'>'JUILLET'
50 AFFICHER T,U,V
60 TERMINER
```

```
PROGRAM ZONE;
(* ZONE OCCUPEE *)
VAR A:REAL;
    B:INTEGER;
BEGIN
  A:=-5;
  A:=A+2*A*A;
  WRITELN(A);
  B:= TRUNC(SQRT(A));
  WRITELN(B)
END.
```

```
1 * ZONE OCCUPEE
10 A_-5
20 A_A+2*A*A
30 AFFICHER A
40 A_ENT(RAC(A))
50 AFFICHER A
60 TERMINER
```

```
PROGRAM SURPRISES;
VAR T,U,V:BOOLEAN;
BEGIN
  T:=8<3;
  U:='BONHEUR'<'MALHEUR';
  V:=('FEVRIER'>'AOUT')AND('JUIN'>'JUILLET');

  WRITELN(T,U,V)
END.
```

```
1 * REVES
10 CHAINE A,B,C
20 A_'RELEVER'
30 B_SCH(A,1,2)!SCH(A,5,3)
40 C_'NOUS '!SCH(B,1,3)!'ONS'
50 AFFICHER B
60 AFFICHER C
70 TERMINER
```

```
PROGRAM REVES;
VAR A,B,C:STRING;
BEGIN
    B:INTEGER;
  A:='RELEVER';
  B:=CONCAT(COPY(A,1,2),COPY(A,5,3));
  C:=CONCAT('NOUS',COPY(B,1,3),'ONS');
  WRITELN(B);
  WRITELN(C)
END.
```

```
1 * VOLTIGES
10 A_3:B_5
20 A_A+A:A_A+A
30 AFFICHER A,B
40 C_A:A_B:B_C
50 AFFICHER A,B,C
60 TERMINER
```

```
PROGRAM VOLTIGES;
VAR A,B,C:INTEGER;
BEGIN
  A:=3;B:=5;
  A:=A+A;A:=A+A;
  WRITE(A,B);
  C:=A;A:=B;B:=C;
  WRITELN(A,B,C)
END.
```

```
1 * ENCORE
10 CHAINE A,B:BOOLEEN T,U
20 A_'ASSEZ!':B_'PARDON?'
30 T_A<B
40 A_SCH(A,2,LGR(A)-1)
50 AFFICHER A
60 U_A<B
70 AFFICHER T,U
80 TERMINER
```

```
PROGRAM ENCORE;
VAR A,B:STRING;
    T,U:BOOLEAN;
BEGIN
  A:='ASSEZ!';B:='PARDON?';
  T:=A<B;A:=COPY(A,2,LENGTH(A)-1);
  WRITE(A);
  U:=A<B;
  WRITELN(T,U)
END.
```

```
1 * CERCLE
10 PI_3.141592
20 LIRE['RAYON DU CERCLE ']R
30 P_2*PI*R;A_PI*R*R
40 AFFICHER P,A
50 TERMINER
```

```
PROGRAM INITIALES;
VAR P,N,I:STRING;
BEGIN
  WRITE('DONNEZ NOM,PRENOM SEPARES PAR ');
  WRITE('LA FRAPPE DE <RETURN> ');
  READLN(N);READLN(P);
  I:=CONCAT(COPY(P,1,1),'.',COPY(N,1,1),'.');
  WRITELN(I)
END.
```

```
1 * INITIALES
10 CHAINE P,N,I
20 LIRE['DONNEZ VOTRE NOM ET VOTRE PRENOM SEPARES PAR LA FRAPPE DE <RETURN> ']
   P,N
30 I_SCH(P,1,1)!'.'!SCH(N,1,1)!'.'
40 AFFICHER I
50 TERMINER
```

```
PROGRAM DISCRIM;
VAR TEST:BOOLEAN;
    A,B,C,D:INTEGER;
BEGIN
  WRITELN('COEFF.DU BINOME SEPARES PAR UN BLANC ');
  READ(A,B,C);
  D:=B*B-4*A*C;
  TEST:=D>=0;
  WRITELN(D,TEST);
END.
```

```
1 * DISCRIMINATION
10 BOOLEEN TEST
20 LIRE['COEFFICIENTS DU TRINOME SEPARES PAR UN BLANC ']A,B,C
30 D_B*B-4*A*C
40 TEST_D>=0
50 AFFICHER D,TEST
60 TERMINER
```

```
PROGRAM PRIX;
VAR PHT,TTVA:REAL;
BEGIN
  WRITELN('PRIX H.TAXES,TAUX TVA ');
  READ(PHT,TTVA);
  WRITELN('PRIX TTC',PHT*(1+TTVA/100))
END.
```

```
1 * N'IMPORTE QUOI
10 AFFICHER 'TO BE OR NOT TO BE...'
20 AFFICHER 1*9,2*9,3*9,4*9,5*9
30 TERMINER
```

```
1 * PRIX
10 LIRE['PRIX HORS TAXES, TAUX TVA ']PHT,TTVA
20 AFFICHER 'PRIX TTC :'
30 AFFICHER PHT*(1+TTVA/100)
40 TERMINER
```

```
PROGRAM FIN;
VAR T:STRING;
    SOS:INTEGER;
BEGIN
  WRITELN('BONJOUR TOUT LE MONDE...'
  T:= 'JE CRAQUE ! ';
  WRITELN(T,'  ',T,'  ',CONCAT(T,'!!!'));
  SOS:=1985;
  WRITELN('SOS',SOS)
END.
```

```
1 * FIN
10 CHAINE T
20 AFFICHER 'BONJOUR TOUT LE MONDE...'
30 T_'JE CRAQUE ! '
40 AFFICHER T,T,T!'!!!'
50 SOS_1985
60 AFFICHER 'SOS' ,SOS
70 TERMINER
```

MINI TP 1

4. Exercices-Mini TP1

4.1 Ecrire un **algorithme** comportant :
en entrée, les coordonnées (abscisse, ordonnée) de deux points A et B
en sortie, les coordonnées du milieu de [AB] et la longueur AB.

4.2 Ecrire un **algorithme** qui aura :
en entrée, une somme en Francs
en sortie, son équivalent en DOLLARS et en MARKS
puis
en entrée une somme en MARKS
en sortie, son équivalent en Francs.
Essayer de résoudre le problème de l'arrondi de la somme au centime.

4.3 Ecrire un **algorithme** qui aura :
en entrée, un verbe (1er groupe, régulier, commençant par une consonne)
en sortie, la conjugaison de ce verbe au futur.

4.4 Même exercice que 4.3, mais conjugaison au présent.

4.5 En entrée, un nombre entier A, et un nombre entier B ($\neq \emptyset$)
en sortie, le quotient et le reste de la division entière de A par B.

4.6 En entrée, un groupe nominal sujet (singulier), un verbe du 1er groupe à l'infinitif, un groupe nominal objet (masculin pluriel)
en sortie, la forme active groupe sujet-verbe groupe objet et la forme passive groupe objet- groupe verbal - groupe sujet.
Exemple : Le chat mange les rats
Les rats sont mangés par le chat.

4.7 En entrée, deux prix hors taxes et les deux taux de T.V.A. correspondants

En sortie, le titre "FACTURES" en haut au milieu de l'écran, encadré par des "*", et le tableau suivant rempli :

PRIX H.T.	*	T.V.A.	*	PRIX T.T.C.
	*		*	
1er ARTICLE	*	.....	*	.....
2ème ARTICLE	*	.....	*	.....
	*		*	
TOTAL	*	.....	*	.....

(Voir annexe 1 pour la présentation)

4.8 En entrée, - le jour de la semaine où tombe Noël, cette année (de Ø à 6 par exemple)

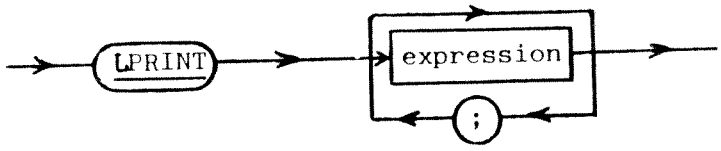
- une année AN

en sortie - le jour de la semaine où Noël eut lieu l'année AN

ANNEXE AU DOSSIER 1

1. Sortie sur imprimante

L'écriture sur imprimante se fait à l'aide de l'instruction



Il s'agit donc simplement de remplacer PRINT par LPRINT dans le paragraphe 2.4 du dossier 1, toutes les autres remarques restant valables.

Remarques

- s'assurer avant l'exécution que l'imprimante est prête, sinon le programme "se plantera"
- si on veut écrire à la fois sur l'écran et sur l'imprimante, le programme doit comporter à la fois l'instruction PRINT et l'instruction LPRINT.

2. Complément sur la mise en page

2.1 On peut effacer le contenu de l'écran à l'aide de l'instruction CLS

Exemple

```
10 REM ECRAN
20 CLS
30 PRINT "IL N'Y A QU'UNE PHRASE SUR L'ECRAN ."
40 END
```

(comparer l'exécution avec celle obtenue sans la ligne 20)

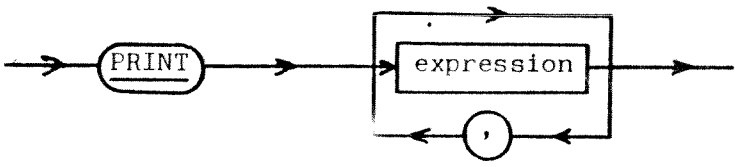
2.2 L'instruction PRINT (ou LPRINT), lorsqu'elle n'est suivie d'aucune expression, fait descendre le curseur d'une ligne.

Exemple :

```
10 REM SAUT
20 CLS
30 PRINT "PREMIERE LIGNE"
40 PRINT:PRINT:PRINT
50 PRINT "CINQUIEME LIGNE"
60 END
```

(Que se passe-t-il si on termine le ligne 30 avec ";" ?).

2.3 Nous avons déjà vu l'utilisation de ";" dans le paragraphe 2.4
Mais on peut aussi utiliser pour PRINT (ou LPRINT) la syntaxe suivante :



L'exé ainsi :
L'écran est virtuellement divisé zones verticales de 14 caractères
Chaque virgule positi r au début de la zone suivante de

Essayer :

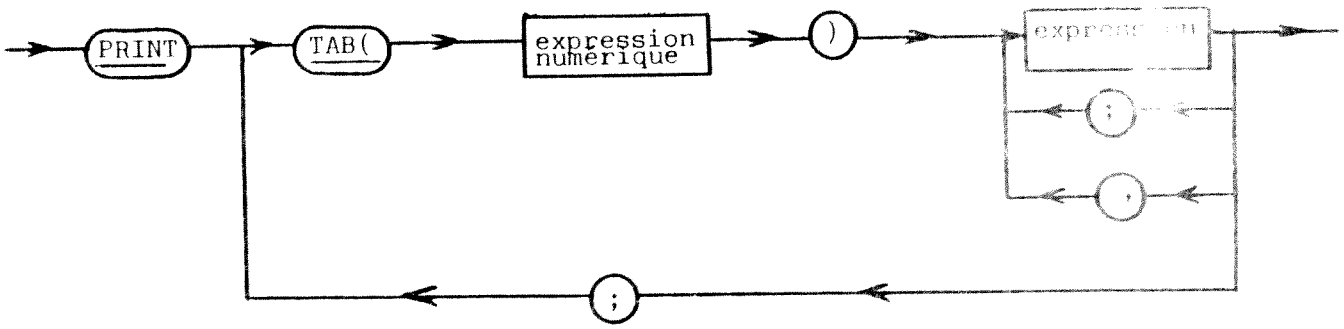
```
10 PRINT "1 ERE ZONE","2 EME ZONE",,"4 EME ZONE"
20 END
```

et

```
10 PRINT "C'EST UN PEU LONG POUR UNE ZONE","VOICI LA 4 EME"
20 END
```

2.4 Il est possible de choisir de façon précise la position où l'on veut commencer l'écriture sur une ligne-écran, en utilisant pour PRINT

(ou LPRINT) la syntaxe suivante :



Exemples :

1°

```
1Ø REM TABULATION
2Ø CLS
3Ø LET T$="I R E M ":LET T$=T$+T$
4Ø PRINT TAB(7)T$
5Ø PRINT TAB(5)T$
6Ø PRINT TAB(3)T$
7Ø PRINT T$
8Ø END
```

```
      I R E M I R E M
    I R E M I R E M
  I R E M I R E M
I R E M I R E M
```

2°

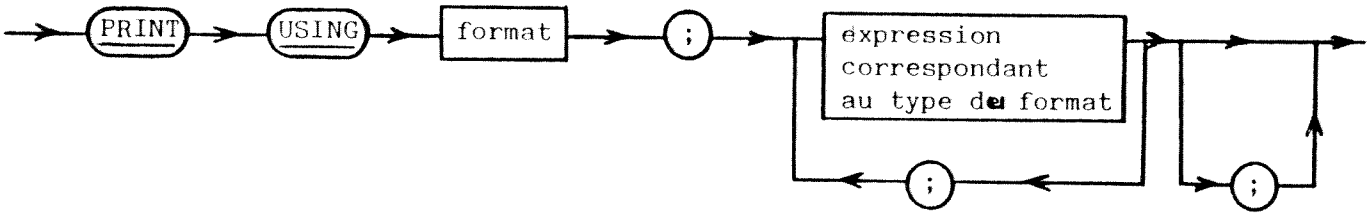
```
1Ø REM POSITION
2Ø CLS
3Ø INPUT "DONNER UN NOMBRE ENTRE 1 ET 4Ø ";N
4Ø PRINT TAB(N)"POSITION";N
5Ø END
```

Remarque :

On pourra faire des essais pour observer ce qui se passe lorsque la position d'écriture demandée est impossible. (Par exemple PRINT TAB(4)"ABCD" ; TAB(6)"EFGH").

2.5 Pour la mise en page de colonnes de nombres ou d'une facture par exemple, on dispose d'une instruction plus puissante que les précédentes, il s'agit de PRINT USING (ou LPRINT USING)

Syntaxe



Voyons ce qu'est un format :

Pour les chaînes de caractères :

1) "!" format qui imprime le ler caractère
2) "\ n blancs \" format qui imprime 2 + n caractères de la chaîne.

Exemple

```
10 A$="JUSTE POUR VOIR ":B$="LE RESULTAT"
20 PRINT USING "\    \";A$;B$
30 PRINT USING "!";A$;B$
```

```
JUSTELE RE
JL
```

3) "&" format qui imprime la chaîne entièrement

Exemple

```
10 A$="JUSTE POUR VOIR ":B$="LE RESULTAT"
20 PRINT USING "&":A$;B$
```

```
JUSTE POUR VOIR LE RESULTAT
```

Pour les nombres

1) # représente une position chiffre dans un format
2) . représente le point décimal dans un format

Exemples :

```
10 PRINT USING "##.##";.78

 0.78

10 PRINT USING "##.##";89.4387
89.44
```

Remarque Si le nombre est plus grand que le format, un % s'imprime devant le nombre.

Exemple :

```
10 PRINT USING "##.##";456.992
%456.99
```

3) + imprime le signe avant et après les nombres positifs

Exemple :

```
10 PRINT USING "+##.##";-68.986;2.4
-68.99 +2.40
```

4) - imprime le signe avant ou après les nombres négatifs

5) ** imprime des "*" dans les portions libres

Exemple :

```
10 PRINT USING "**#.#";12.39;8.987
*12.4**9.0
```

6) $ imprime un $ devant le nombre (se rappeler l'origine américaine du BASIC et l'utilisation de PRINT USING en facturation).

7) **$ combine l'effet des deux règles précédentes

Exemple :

```
10 PRINT USING "**#.#";12.39;8.987
*12.4**9.0
```

8) , la virgule à gauche du point décimal s'imprime tous les trois chiffres à gauche du point.

Exemple :

```
10 PRINT USING "####,.##";1234.5
1.234.50
```

9) ^^^^ les quatre " ^ " indiquent un format exponentiel (les nombres vont être imprimés sous forme d'un nombre décimal multiplié par la puissance de 10 indiquée)

Exemple :

```
10 PRINT USING "##.##^^^^";234.56
 2.35E+02
```

10) _ le caractère de soulignement indique que le caractère suivant doit être imprimé en tant que chaîne

Exemple :

```
10 PRINT US.NG "_!##.##_!";12.34
!12.34!
```

Fonction RND

Cette fonction donne un nombre pseudo-aléatoire et permet donc la programmation de jeux ou de problèmes où l'on veut faire intervenir le "hasard".

Sur certains équipements

RND(Ø) fournit un nombre "réel" compris entre Ø et 1

RND(n) où n est une expression entière fournit un nombre entier compris entre 1 et n.

Exemple :

```
10 REM RND
20 PRINT RND(0),RND(30),RND(10),RND(10)
30 END
```

```
 .0666569         5              7              8
```

Sur MICRAL

RND (l'argument est inutile) fournit un nombre "réel" compris entre Ø et 1. L'équivalent du RND(n) n'existe pas, mais on se convaincra que l'on peut obtenir le même résultat avec l'instruction : PRINT INT(RND*n+1)

Exemple :

```
10 REM RND
20 PRINT RND,RND,INT(RND*30+1),INT(RND*30)
30 END
```

Chaque exécution de ce programme donne les mêmes nombres.

Pour éviter cet inconvénient, on peut insérer la ligne 15 :

15 RANDOMIZE

Cette instruction a pour but de changer le générateur de nombres aléatoires.
A l'exécution, elle interrompt le programme et attend de l'utilisateur un nombre compris entre - 32768 et 32767 pour continuer.

Pour utiliser la fonction RND, voir les exercices 9 et 1Ø du Mini TP 2.

DOSSIER 2

Ø. Rappels sur les booléens - notion de condition

Un traitement informatique doit englober tous les cas possibles résultant des données fournies en entrée. Certaines instructions ou blocs d'instructions devront donc être effectués conditionnellement.
Une **condition** étant vraie ou fausse, pourra être représentée par un ou plusieurs booléens reliés par des opérateurs logiques :

Exemple :

```
A > 8
REPONSE = "    "
TROUVE
(I < N) et (DRAPEAU)
NON (REP = " ")
(SEXE = "H") ET ((AGE > 3Ø) OU (AGE < 5))
```

1. Structures alternatives (ou conditionnelles)

Une telle structure permet l'exécution d'un bloc d'instructions ou d'un autre, selon qu'une certaine condition est vérifiée ou non (alternative simple).
Plus généralement, on envisagera l'exécution d'un bloc parmi plusieurs selon la réalisation d'une condition parmi plusieurs s'excluant mutuellement (alternative généralisée).

1.1 La structure alternative simple

Sa forme générale est

```
SI  condition
ALORS
    * BLOC OUI
    .
    .
    .
SINON
    * BLOC NON
    .
    .
```

Exemples :

* Recherche du plus petit et du plus grand de deux nombres

	* MAXI MINI
X, Y, MINI, MAXI(entiers)	LIRE X, Y Si X < Y ALORS * CAS 1 MINI ← X MAXI ← Y SINON * CAS 2 MINI ← Y MAXI ← X ECRIRE X,Y ECRIRE MINI, MAXI

Cet algorithme peut être simplifié si on cherche à économiser les emplacements mémoires :

	* MAXI MINI 2
X, Y(entiers) : valeurs à classer ECH(entier) : case relais	LIRE X, Y SI X > Y ALORS * ECHANGE ECH ← Y Y ← X X ← ECH SINON * RIEN ECRIRE X, Y

On observera que :

- l'échange des contenus de deux zones mémoires nécessite une troisième zone afin de ne pas perdre l'une des valeurs comme le ferait Y ← X suivi de X ← Y qui mettrait le contenu initial de X dans les deux zones.

- un bloc d'instructions peut être **vide** (il n'y a rien à faire). On le mentionnera néanmoins, surtout au début, pour s'assurer de l'écriture de toutes alternatives, lorsqu'il y en a plusieurs. La plupart des langages de programmation prévoient que la branche "SINON..." de l'alternative est facultative (certains l'ignorent même ; nous verrons sur un exemple comment y pallier !)
- l'instruction "ECRIRE MINI, MAXI" doit être exécutée dans les deux branches de l'alternative. On la placera donc à la suite de la structure alternative, ce qui permet de ne l'écrire qu'une seule fois dans le programme.

* Rangement de trois nombres entiers dans l'ordre croissant

Il s'agit de lire trois nombres dans les zones X, Y, Z et de faire en sorte qu'à l'issue du traitement X contienne le plus petit et Z le plus grand des trois.

	*** RANGEMENT DE 3 NOMBRES**

X, Y, Z(entiers) : données lues
 	
ECH(entier) : case relais
	LIRE X, Y, Z
	SI X > Y
	ALORS
	  * ECHANGE 1
	  ECH ← X
	  X ← Y
	  Y ← ECH
	
	SI X > Z
	ALORS
	  * ECHANGE 2
	  ECH ← X
	  X ← Z
	  Z ← ECH
	SI Y > Z
	ALORS
	  * ECHANGE 3
	  ECH ← Y
	  Y ← Z
	  Z ← ECH
	
	ECRIRE X, Y, Z

Nous utiliserons ici trois alternatives simples consécutives sans branche "SINON..."

Remarque :

Le traitement à effectuer étant chaque fois le même (échange de deux zones mémoire) on peut imaginer de l'écrire une seule fois et d'y faire appel aux différentes étapes du programme. Nous ferons appel à la notion de **module** ou **procédure.**

2. Procédures - Programmation modulaire - Appel de fonction

2.1 L'appel de modules avec ou sans paramètres

Lorsqu'une séquence d'instructions revient plusieurs fois dans un même programme, ou lorsque cette séquence constitue un tout auquel il est intéressant de se référer ; on peut l'organiser en **module** que l'on appelle une ou plusieurs fois au cours du programme.

Exemple :

Reprenons ici le problème du rangement de trois nombres envisagé au début de ce dossier :

	*** RANGEMENT 3 NOMBRES 2**

X,Y,Z(entiers) : valeurs lues
ECHANGE(module) : traitement d'échange de deux données
	LIRE X,Y,Z
	SI X > Y
	ALORS
	  EXECUTER ECHANGE (X,Y)
	SI X > Z
	ALORS
	  EXECUTER ECHANGE (X,Z)
	SI Y > Z
	ALORS
	  EXECUTER ECHANGE (Y,Z)
	ECRIRE X,Y,Z

	* ECHANGE (A,B)
A,B,ECH(entiers) : données à permuter	ECH ← A A ← B B ← ECH

L'instruction "exécuter" suivie d'un nom de module, permet l'appel du module ECHANGE en différents endroits du programme.

Dans l'exemple ci-dessus, le module ECHANGE est appelé trois fois mais opère sur des données différentes. Celles-ci sont fournies en paramètres d'entrée (exécuter ECHANGE (X,Y)).
Dans cet exemple, les paramètres sont en même temps paramètres de sortie, rendus par le module au programme principal.

Les variables A,B,ECH sont, en quelque sorte des variables muettes, aux quelles seront substitués les arguments effectifs, lors de l'appel de la procédure. Selon les langages de programmation, elles existent réellement en mémoire et seules les **valeurs** des arguments leur sont transmises (transmission par valeur) ou alors les paramètres fictifs sont remplacés, lors de l'appel, par les **variables** du programme appelant (dans l'exemple, X est substitué à A, Y à B). On parle alors de transmission par variable.

La variable ECH est crée localement. Elle existe dans l'environnement de la procédure, on parlera de **variable locale** par opposition à X,Y,Z qui sont globales au programme appelant et à la procédure appelée).

Au niveau de l'algorithme, seul importent l'ordre dans lequel elles sont mentionnées, ainsi que la concordance des types (entier, réel, chaîne etc...) entre variables du programme appelant et paramètres de la procédure.

2.2 L'appel de fonctions programmées

En dehors des fonctions standard décrites dans le dossier 1, il s'agit de modules programmables dans lesquels le résultat, numérique, booléen ou alphanumérique est accessible en utilisant le nom de la fonction

Exemple

	*** SECOND DEGRE**
A,B,C(réels) DELTA(fonction)	LIRE A,B,C SI **DELTA** (A,B,C) > ∅ ALORS * DEUX . . SINON * UN OU ZERO . .
	*** DELTA**(X,Y,Z)
U,V, DELTA(réels)	U ← Y * Y V ← X * Z DELTA ← U - 4 * V

3. Alternatives emboitées ou consécutives

3.1 L'algorithme suivant prévoit trois structures **alternatives emboitées** permettant de gérer les quatre situations possibles.
La présentation **indentée** (décalages successifs à droite) permet de montrer clairement ces emboitements :

* Positionnement d'un nombre parmi trois nombres déjà classés

A l'issue du traitement X,Y,Z devrait contenir les 3 plus petits de X, Y, Z et d'un nombre N.

	* POSITION N
X,Y,Z(entiers) : nombres déjà classés N(entier) : nombre à situer par rapport aux précédents	LIRE X,Y,Z LIRE N SI N < X ALORS * CAS 1 Z ← Y Y ← X X ← N SINON *CAS 2 SI N < Y ALORS * CAS 21 Z ← Y Y ← N SINON * CAS 22 SI N < Z ALORS * CAS 221 Z ← N SINON * CAS 222 ECRIRE X,Y,Z

On aurait pu traiter le problème par 3 (ou 4) **structures alternatives simples ou consécutives,** à l'aide d'expressions booléennes plus complètes :

```
Si N < X
ALORS
    || * CAS 1
SI X ≤ N < Y
ALORS
    || * CAS 2
SI Y ≤ N < Z
ALORS
    || * CAS 3
SINON
    || * CAS 4
```

ou encore

```
SI N < X
ALORS
||* CAS 1
SI X ≤ N < Y
ALORS
||* CAS 2
SI Y ≤ N < Z
ALORS
||* CAS 3
SI Z ≤ N
ALORS
||* CAS 4
```

Ce dernier exemple nous conduit naturellement à la notion d'alternatives généralisées.

3.2 La structure **alternative généralisée**

Il s'agit de gérer plus que deux éventualités. La forme générale de l'algorithme est la suivante :

```
SI          Condition 1
ALORS           ||  * BLOC 1
            Condition 2
ALORS           ||  * BLOC 2
            Condition 3
ALORS           ||  * BLOC 3
          Condition (n-1)
ALORS           ||  *BLOC n-1
SINON
                ||  * BLOC N
```

Exemple :

*** CALCUL DE REMISE**

	*** PRIX**
QTE(entier) : quantité achetée PU(réel) : prix unitaire nominal PV(réel) : prix de vente après remise TT(réel) : prix facturé	LIRE QTE, PU SI QTE < 500 ALORS * CAS 1 PV ← PU QTE < 1000 ALORS * CAS 2 PV ← PU * 0.80 1000 ≤ QTE < 5000 ALORS * CAS 3 PV ← PU * 0.66 SINON * CAS 4 PV ← PU * 0.50 TT ← PV * QTE ECRIRE TT

Remarques

Les conditions C1, C2,... Cn, sont généralement 2 à 2 disjointes (Ci et Cj ne peuvent être vraies simultanément pour i = j) et la branche "SINON" facultative, englobe les cas non envisagés précédemment.
Dans le cas contraire, il faut s'assurer de la compatibilité des traitements de Ci et Cj aux frontières.

Exemple

CAS 1 QTE < 500

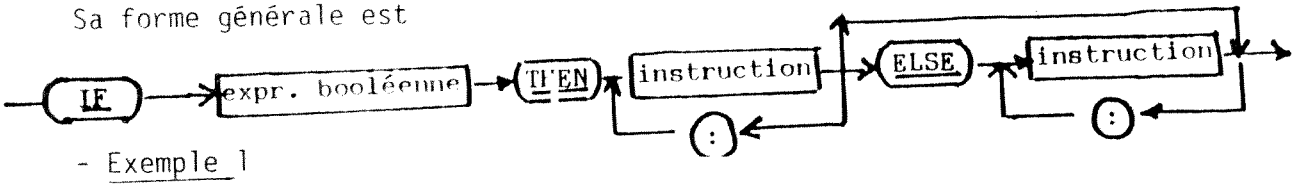
CAS 2 500 ≤ QTE < 1000

CAS 3 (QTE > 1000) OU (CLI = "GROS" ET QTE > 100)

4 Réalisation en BASIC

4.1 Instruction IF .. THEN ... ELSE...

Sa forme générale est



- Exemple 1

```
10 REM Maxi mini
20 INPUT "Nombres a ranger"!X,Y
30 IF X<Y THEN MINI=X!MAXI=Y ELSE MAXI=X!MINI=Y
40 PRINT "Nombres donnes"!X,Y
50 PRINT "Nombres ranges"!MINI,MAXI
60 END
Nombres donnes 45    12
Nombres ranges 12    45
```

- Exemple 2

```
10 REM Maxi mini
20 INPUT "Nombres a ranger"!X,Y
30 IF X>Y THEN ECH=Y!Y=X!X=ECH
40 PRINT "Nombres rangés",X,Y
60 END

                Nombres rangés      10      35
```

Remarque :

Certains BASIC restreignent l'écriture de plusieurs instructions après THEN ou/et ELSE ... Dans ce cas, on fera appel à GOSUB cf. § 4.2

- Exemple 3

```
10 REM Rangement de trois nombres
20 INPUT "Nombres a ranger"!X,Y,Z
30 IF X>Y THEN ECH=X!X=Y!Y=ECH
40 IF X>Z THEN ECH=X!X=Z!Z=ECH
50 IF Y>Z THEN ECH=Y!Y=Z!Z=ECH
60 PRINT "Nombres rangés"!X,Y,Z
70 END

                Nombres donnes 3    6    2
                Nombres rangés 2    3    6
```

4.2 Appels de modules

Les BASIC ordinaires ne disposent pas d'appel de procédures avec paramètres d'entrée ou de sortie à transmettre. Il est simplement possible d'interrompre le déroulement séquentiel du programme pour aller exécuter un bloc d'instructions repéré par un numéro de ligne donné. On dispose pour cela de :

où entier représente le numéro de ligne où débute le bloc. Ce dernier doit se terminer par RETURN permettant de retourner à l'instruction correspondante du programme principal (généralement celle qui suit l'instruction d'appel).

Exemple :

```
10 REM Position N
20 INPUT "Nombres ranges X,Y,Z";X,Y,Z
30 INPUT "Nombre a placer N";N
40 IF N<X THEN GOSUB 80 ELSE GOSUB 100
50 PRINT "Nombres places";X,Y,Z
60 END
80 REM Cas 1
85 ' Z=Y:Y=X:X=N
90 RETURN
100 REM Cas 2
110 ' IF N<Y THEN Z=Y:Y=N ELSE GOSUB 200
120 RETURN
200 REM Cas 22
210 ' IF N<Z THEN Z=N
220 RETURN
```

L'écriture la plus claire d'une structure alternative est la suivante :

```
(n)     IF condition THEN GOSUB n1 ELSE GOSUB n2
  .
  .
  .     END
(n1)    REM BLOC OUI
  .
  .
  .     RETURN
(n2)    REM BLOC NON
  .
  .
  .     RETURN
```

On veillera à placer l'instruction END (qui précise la dernière instruction exécutable en séquence) avant les différents blocs repérés par des GOSUB faute de quoi l'exécution du programme se poursuivra en séquence dans l'un des blocs et la rencontre d'un RETURN non consécutif à un GOSUB provoquera une erreur.
Ainsi dans l'exemple précédent, l'omission de 6Ø END conduirait le programme à afficher X, Y, Z (instruction 5Ø) puis à exécuter 1ØØ et 11Ø où l'instruction RETURN provoquerait une erreur.

Noter que le BASIC ne prévoit pas le transfert de paramètres ni de variables locales. Toutes les variables étant globales, elles seront utilisées indifféremment dans le module appelé et dans le module appelant.

Exemple :

```
10 REM Calcul de Cnp
20 INPUT "Valeurs de n et de p" ;N,P
30 I=N:GOSUB 1000:CNP=J
40 I=P:GOSUB 1000:CNP=CNP/J
50 I=N-P:GOSUB 1000:CNP=CNP/J
60 PRINT "Cnp= ";CNP
70 END
1000 REM Factorielle
1010 J=1
1020 WHILE I>1 DO
1030 ' REM Multiple
1040 ' J=J*I
1050 ' I=I-1
1060 WEND
1070 RETURN
```

4.3 Cas particuliers

- Certains BASIC ne disposent pas de l'option ELSE...

Dans ce cas, l'écriture à adopter est la suivante :

(n) IF condition, THEN Instructions du bloc OUI, : GOTO (n_2)

(n_1) { instructions du BLOC NON

(n_2) { suite du programme
⋮

L'instruction **GOTO** a la même syntaxe que GOSUB. Elle permet le branchement vers une ligne quelconque du programme. Il est préférable de ne l'utiliser qu'exceptionnellement, dans des cas similaires à l'exemple précédent, afin de conserver au programme une structure séquentielle clairement lisible. L'instruction GOTO n n'implique pas de retour.

```
0010 REM MAXIMINI
0020 INPUT "Nombres a ranger";X,Y
0030 IF X<Y THEN MINI=X:MAXI=Y:GOTO 50
0040   MAXI=X:MINI=Y
0050 PRINT "NOMBRES RANGES";MAXI,MINI
0060 END
```

La ligne 4Ø ne sera exécutée que si la conditions X < Y est fausse. La ligne 5Ø sera toujours exécutée.

4.4 Les fonctions en BASIC

4.4.1 Fonctions standard (rappel)

- Fonctions sur les nombres entiers ou réels

Trigonométriques :

SIN(, COS(, TAN(, ATAN(

Exponentielles

EXP(, LOG(

Diverses :

ABS(: valeur absolue de
INT(: partie entière de
SGN(: signe de
SQR(: racine carrée de
RND(: nombre pseudo-aléatoire

- Fonctions opérant sur les chaînes de caractères

MID$ (A$,M,N) représente la sous-chaîne extraite de A$ de longueur N à partir du Me caractère

LEFT$(A$,M) partie gauche de A$(M caractères)
RIGHT$(A$,N) partie droite de A$ (N caractères)
LEN (A$) entier représentant la longueur de la chaîne
SPACE$(n) génère n blancs
ASC(C$)et CHR$(n) conversion d'un caractère en son code ASCII et vice versa.

exemples :
ASC("A") vaut 65 (ou 41 H)
CHR$(65) vaut "A"

VAL("55.8") et STR$(55.8) conversion d'un nombre en une chaîne et vice-versa

- fonctions de gestion de l'écran

CLS (clear screen) efface l'écran

CURSOR (X,Y) positionne à la colonne X et la ligne Y

4.4.2 Les fonctions programmées

Exemple

```
10 REM Resolution equation second degre
20 DEF FND(X,Y,Z)=Y*Y - 4*X*Z
30 INPUT "coefficients A,B,C";A,B,C
40 IF FND(A,B,C)<0 THEN PRINT "Racines imaginaires" ELSE PRINT "Reelles"
50 END
```

(voir § 2 pour la structure IF... THEN... ELSE)

La fonction D est définie par le programmeur en ligne 2Ø à l'aide de DEF FN.
Elle est ensuite utilisée une ou plusieurs fois au cours du programme.
Les variables X,Y,Z sont "muettes", les arguments A,B,C s'y substituent à l'exécution (ligne 4Ø);
Rem : Certains BASIC n'admettent que des fonctions d'une variable.

4.4.5 Alternatives (cas plus complexes)

Exemples du § 3.1

```
10 REM Position N
20 INPUT "Nombres rangés X,Y,Z";X,Y,Z
30 INPUT "Nombre a placer N";N
40 IF N<X THEN GOSUB 80 ELSE GOSUB 100
50 PRINT "Nombres places";X,Y,Z
60 END
80 REM Cas 1
85 ' Z=Y:Y=X:X=N
90 RETURN
100 REM Cas 2
110 ' IF N<Y THEN Z=Y:Y=N ELSE GOSUB 200
120 RETURN
200 REM Cas 22
210 ' IF N<Z THEN Z=N
220 RETURN
```

```
10 REM Calcul du prix de vente
20 INPUT "Nombre,prix unitaire et client ";NB,PU,CL$
30 IF NB<=500 THEN R=1
40 IF (NB>500)AND (NB<1000) THEN R=0.9
50 IF (NB>1000)OR((CL$="GROS")AND(NB>100))THEN R=0.7
60 PT=PU*NB*R
70 PRINT "PRIX DE VENTE ";PT
80 END
```

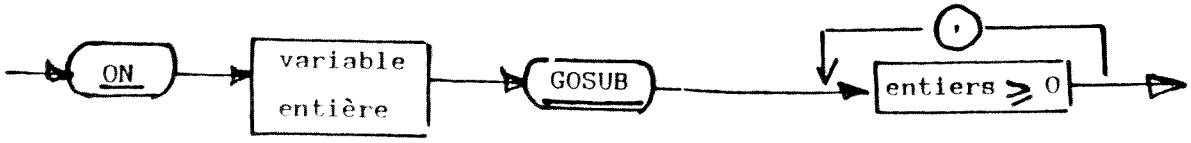
```
Nombre,prix unitaire et client  300     1000    GROS
PRIX DE VENTE  210000


Nombre,prix unitaire et client  1500    500     DETL
PRIX DE VENTE  525000


Nombre,prix unitaire et client  600     300     DETL
PRIX DE VENTE  162000
Nombre,prix unitaire et client  300     300     DETL
PRIX DE VENTE  90000
```

4.6 Alternatives généralisées

Les BASIC actuels ne permettent de programmer les alternatives généralisées que si les conditions C1, C2, ..., C_{n-1} se ramènent aux valeurs successives d'une variable entière



une variante (déconseillée) consiste à remplacer GOSUB par GOTO.

Pour les autres alternatives généralisées, on fera appel à plusieurs structures IF... THEN...ELSE...emboitées ou consécutives.

Exemple : (traitement en fonction du choix dans un "menu".

On affiche l'ensemble des traitements possibles (menu) ; à chaque traitement correspond un numéro.

L'utilisateur entre ce numéro, ce qui déclanche la sélection du traitement correspondant.

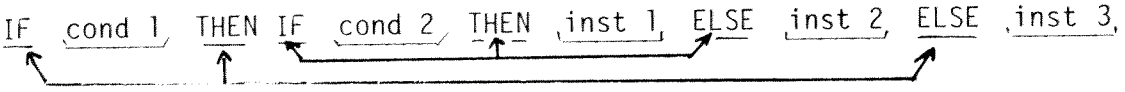
```
10 REM Affichages
20 PRINT "1.Liste brute"
30 PRINT "2.Liste alphabetique"
40 PRINT "3.Liste par numeros croissants"
50 PRINT "4.Statistiques sur le fichier"
60 PRINT "5.Fin du traitement"
70 INPUT "Votre choix "I
80 ON I GOSUB 100,200,300,400,500
90 END
100 REM Liste brute
190 RETURN
200 REM Liste alphabetique
290 RETURN
300 REM Liste par numeros
390 RETURN
400 REM Statistiques
490 RETURN
500 REM Fin du traitement
510 RETURN
```

A noter que certains BASIC autorisent les GOSUB calculés. L'instruction 9Ø de l'écriture précédente se simplifie alors en :

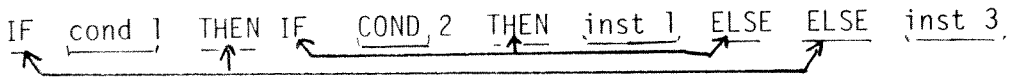
```
9Ø  ON I GOSUB 100 * I
```

Alternatives emboitées

Elles sont délicates à manipuler si l'on ne se sert pas de sauts par GOSUB.
Noter toutefois que à un stade donné un ELSE se rapporte habituellement au IF... THEN le plus proche :



Le cas échéant une clause ELSE sera écrite avec une instruction vide :



dans cet exemple il n'y a rien à faire si la condition 2 est fausse.

```
 * Maxi Mini
10 LIREC/,'Nombres a ranger ? 'JX,Y
20 SI X>Y ALORS DEBUT ECH←Y;Y←X;X←ECH FIN
30 AFFICHERC/J'Nombres ranges ',X,Y
40 TERMINER
```

```
PROGRAM MAXMIN;
VAR X,Y,ECH:INTEGER;
BEGIN
  WRITELN('NOMBRES A RANGER ');
  READ(X,Y);
  IF X>Y THEN
    BEGIN
    ECH:=Y;Y:=X;X:=ECH
    END;
  WRITELN('NOMBRES RANGES ',X,Y)
END.
```

```
 * Rangement de trois nombres
10 LIREC/,'Nombres a ranger ? 'JX,Y,Z
20 SI X>Y ALORS &ECHAN(X,Y)
30 SI X>Z ALORS &ECHAN(X,Z)
40 SI Y>Z ALORS &ECHAN(Y,Z)
50 AFFICHERC/J'Nombres ranges : ',X,Y,Z
60 TERMINER
100 PROCEDURE &ECHAN(A,B)
110 ECH←A;A←B;B←ECH
120 RETOUR
```

```
PROGRAM RANGMT;
VAR X,Y,Z:INTEGER;
PROCEDURE ECHAN(VAR A,B:INTEGER);
VAR ECH:INTEGER;
BEGIN
  ECH:=A;A:=B;B:=ECH
END;
(* PROGRAMME PRINCIPAL *)
BEGIN
  WRITELN('NOMBRES A RANGER ?');
  READ(X,Y,Z);
  IF X>Y THEN ECHAN(X,Y);
  IF X>Z THEN ECHAN(X,Z);
  IF Y>Z THEN ECHAN(Y,Z);
  WRITELN('NOMBRES RANGES ',X,Y,Z)
END.
```

```
1 * Position N
10 LIREC/,'Nombres ranges X,Y,Z ? 'JX,Y,Z
20 LIREC/,'Nombre a placer N ? 'JN
30 SI N<X ALORS &CAS1() SINON &CAS2()
40 AFFICHERC/J'Nombres places : ',X,Y,Z
50 TERMINER
100 PROCEDURE &CAS1()
110 Z←Y;Y←X;X←N
120 RETOUR
130 PROCEDURE &CAS2()
140 SI N<Y ALORS DEBUT Z←Y;Y←N FIN SINON &CAS22()
150 RETOUR
160 PROCEDURE &CAS22()
170 SI N<Z ALORS Z←N
180 RETOUR
```

```
PROGRAM POSITION;
VAR X,Y,Z,N:INTEGER;
PROCEDURE CAS1;
BEGIN
  Z:=Y;Y:=X;X:=N
END;
PROCEDURE CAS22;
BEGIN
  IF N<Z THEN Z:=N
END;
PROCEDURE CAS2;
BEGIN
  IF N<Y THEN BEGIN Z:=Y;Y:=N END
  ELSE CAS22;
END;
(* PROGRAMME PRINCIPAL *)
BEGIN
  WRITE('X Y Z');READ(X,Y,Z);
  WRITE('N ');READ(N);
  IF N<X THEN CAS1 ELSE CAS2;
  WRITELN('NOMBRES PLACES ',X,Y,Z)
END.
```

```
 1 * EXEMPLES
 10 AFFICHER RAC(25),SIN(3),ENT(8.66)
 20 CHAINE A,C,D
 30 A←'TO1TO';C←SCH(A,3,1)
 40 AFFICHERC/,U,2X,U,4X,UJA,C,LGR(A)
 50                 I←1
 55         D←SCH(A,I,1)
 60         AFFICHERC/,U,4X,U,4X,UJD,EQN(D),EQC(90)
 70 AFFICHER SCH(CCA(445.65),4,2)
 80 TERMINER
```

```
 1 * Resolution equation du second deare
 10 LIREC/,'Coefficients A,B,C ? 'JA,B,C
 20 SI &DELTA(A,B,C)<0 ALORS AFFICHERC/J'Racines imaginaires' SINON AFFICHER '
    Racines reelles'
 30 TERMINER
 100 PROCEDURE &DELTA(X,Y,Z) LOCAL Z,Y,X
```

```
* Affichages
  I←0
) FAIRE 110 TANT QUE I≠5
)       AFFICHER[/]'1 Liste brute'
)       AFFICHER[/]'2 Liste alphabetique'
)       AFFICHER[/]'3 LIste par numeros croissants'
)       AFFICHER[/]'4 Statistiques sur le fichier'
)       AFFICHER[/]'5 Fin du traitement'
)       LIRE[/,' VOTRE CHOIX ? ']I
)       SI I=1 ALORS &BRUTE()
)       SI I=2 ALORS &ALPHA()
00      SI I=3 ALORS &CROIS()
10      SI I=4 ALORS &STAT()
20 &FINT()
30 TERMINER
00 PROCEDURE &BRUTE()
90 RETOUR
00 PROCEDURE &ALPHA()
90 RETOUR
00 PROCEDURE &CROIS()
90 RETOUR
00 PROCEDURE &STAT()
90 RETOUR
00 PROCEDURE &FINT()
90 RETOUR
```

```
PROGRAM SCDEGRE ;
VAR A,B,C:REAL;
FUNCTION DELTA(X,Y,Z:REAL):REAL;
BEGIN
  DELTA:=Y*Y-X*Z
END;
(* PROGRAMME PRINCIPAL *)
BEGIN
  WRITELN('A B C ');READ(A,B,C);
  IF DELTA(A,B,C)<0 THEN WRITELN('0 RACINES ')
  ELSE WRITELN('RACINES REELLES ')
END.
```

```
PROGRAM AFFICHAGES;
VAR I:INTEGER;
PROCEDURE BRUTE;
BEGIN END;
PROCEDURE ALPHA;
BEGIN END;
PROCEDURE CROISS;
BEGIN END;
PROCEDURE STAT;
BEGIN END;
PROCEDURE FINT;
BEGIN END;
(* PROGRAMME PRINCIPAL *)
BEGIN
  WRITELN('1.LISTE BRUTE ');
  WRITELN('2.LISTE ALPHABETIQUE ');
  WRITELN('3.LISTE NUMEROS CROISSANTS ');
  WRITELN('4.STATISTIQUES SUR LE FICHIER ');
  WRITELN('5.FIN DE TRAITEMENT ');
  WRITELN('VOTRE CHOIX ');READ(I);
  CASE I OF
    1:BRUTE;
    2:ALPHA;
    3:CROISS;
    4:STAT;
    5:FINT
  END;
END.
```

MINI T.P. 2

0. Ecrire un algorithme permettant
   - de lire 3 nombres X,Y,Z
   - de déterminer celui dont la valeur est comprise entre les deux autres.

1. Ecrire l'algorithme complet de résolution de l'équation du second degré après lecture des coefficients A,B,C (pouvant être nuls).

2. Programmation d'une montre digitale.
   Soit une date sous la forme JJ, MM, AA
   Déterminer la date du lendemain.

3. Algorithme de contrôle de la saisie d'une donnée
   - lire le type de la donnée (N numérique, A alphanumérique)
   - lire le nombre de caractères (ou chiffres) prévus (y compris ponctuation)
   - lire la donnée

     - la cadrer à droite si elle est numérique
     - la tronquer si elle est trop longue
     - sinon la compléter
       avec des Ø à gauche si elle est numérique
       avec des blancs à droite si elle est alphanumérique

4. Ecrire un algorithme permettant de déterminer un nombre entier compris entre 1 et 16, choisi par le joueur, en lui posant un nombre minimal de questions auxquelles il peut répondre par "OUI" ou par "NON".

DOSSIER 3

STRUCTURES REPETITIVES ET TABLEAUX

1. Structures répétitives (ou itératives)

Les structures répétitives constituent un groupe d'instructions essentiel à la programmation d'algorithmes.

La **structure répétitive permet** de spécifier l'exécution, un certain nombre de fois, d'un bloc (c.f. §1.6 dossier 1). Nous étudierons 2 structures répétitives :

- Il est possible de demander l'exécution d'un bloc d'instructions tant qu'une certaine condition est vérifiée.
- On peut demander de répéter l'exécution d'un bloc d'instructions en fixant a priori le nombre de répétitions.

Avant de programmer tel ou tel exercice, il est important de rechercher la structure appropriée : dans chaque cas, l'une des structures est mieux adaptée à l'algorithme considéré. Nous étudierons tout d'abord la structure la plus classique :

1.1 La structure répétitive à bornes définies

Cette structure de boucle est caractérisée par une variable de contrôle qui varie d'une valeur initiale à une valeur connue a priori. La forme générale est :

```
POUR variable DE const 1 A const 2 PAS const 3
REPETER
  || * BLOC REPET
```

Le bloc d'instructions est exécuté une première fois, la variable de contrôle ou **compteur** ayant la valeur numérique initiale, donnée par const 1. Cette dernière est alors augmentée du **pas** (qui peut être négatif), puis le bloc d'instructions est de nouveau exécuté, et ainsi de suite, jusqu'à ce que la variable de contrôle prenne la valeur numérique finale donnée par const 2. Le bloc d'instructions est alors exécuté une dernière fois.

Remarques

Suivant les langages de programmation et même pour un langage identique, suivant les machines, les différences sont sensibles :

1° la variable de contrôle à la sortie de la boucle peut avoir la valeur correspondant à la dernière exécution ou une autre valeur.

2° Si la valeur initiale est supérieure à la valeur finale et le pas positif,(ou la valeur initiale inférieure à la valeur finale et le pas négatif) certaines machines exécuteront le bloc une fois, d'autres ne l'exécuteront pas du tout.

3° Si le pas n'est pas mentionné, il est réputé valoir + 1

Exemples

* - Edition d'une table d'entiers, carrés, cubes et racines carrées.

	* EXEMPLE 1
N (entier) : taille de la table C (entier) : carré I (entier) : variable de contrôle	. LIRE N . POUR I DE 1 A N REPETER || * LIGNE || . C ← I * I || . ECRIRE I,C,C * I,RAC(I)

* - Somme de N entiers et de leurs carrés.

	* EXEMPLE 2
N (entier) : nombre de données S (entier) : somme de données T (entier) : somme des carrés K (entier) : compteur M (entier) : nombre lu	. LIRE N . S ← Ø . T ← Ø . POUR K DE 1 A N REPETER |* SOMMES |. LIRE M |. S ← S + M |.T ← T + M * M . ECRIRE S,T

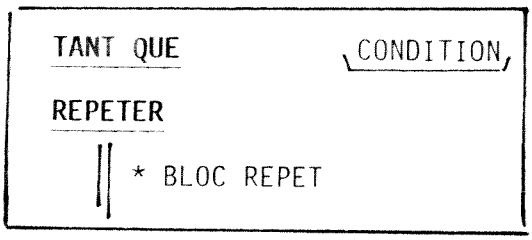
Toute variable apparaissant à droite dans une instruction d'affectation doit être initialisée auparavant.

exemple : S ← ∅ S ← S + M

Le bloc est répété sous le contrôle du compteur. Ce compteur (ou variable de contrôle) peut être (ou ne pas être) utilisé dans le bloc d'instructions à répéter : dans l'exemple 1 il est utilisé ; par contre, dans l'exemple 2, il n'intervient pas.

1.2 La structure répétitive conditionnelle

Cette structure permet de répéter l'exécution d'un bloc tant qu'une certaine condition est remplie. La forme générale de cette instruction est :



L'exécution du bloc d'instructions sera répétée tant que la valeur de l'expression booléenne est VRAIE. Si, notamment, la valeur booléenne est fausse, il n'y aura pas d'exécution.

Pour qu'une structure répétitive "tant que" se termine, il faut donc que la valeur booléenne soit modifiée par l'exécution du bloc. Cette structure répétitive se caractérise par la fait que le nombre de répétitions n'est pas fixé a priori comme dans la structure à bornes définies, l'exécution du bloc se répète tant que la condition est satisfaite. Cette structure est, par exemple, très utile dans les problèmes d'analyse numérique où il est nécessaire de répéter un calcul tant qu'une certaine précision n'a pas été obtenue (Cf. Mini TP exercice ∅) ou dans les problèmes de gestion où le nombre d'objets à manipuler n'est pas connu (le dernier objet est en général suivi d'une marque spéciale indiquant la fin.

Exemples :

* - La série harmonique diverge, trouver le nombre de termes de la série nécessaire au dépassement d'une valeur arbitraire donnée

$$1 + \frac{1}{2} + \ldots\ldots + \frac{1}{n} > \text{VALEUR}$$

*** EXEMPLE 3**

I(entier) : nombre de termes nécessaire au dépassement
S(réel) : somme partielle de la série
VALEUR(réel) : valeur donnée

```
. S ← 0
. I ← 1
. LIRE VALEUR
. TANT QUE S < VALEUR faire
  REPETER
      * AJOUTER
      . S ← S + 1/I
      . I ← I + 1
. ECRIRE I,S
```

* - Lire un mot, lettre par lettre, le réecrire, codifié à l'envers

*** CODIFICATION**

CAR(caractère) : lettre courante du mot
MOT(chaîne) : mot résultant

```
LIRE CAR
MOT  " "
TANT QUE CAR < > " ␣"
REPETER
    *ENVERS
    MOT ← CAR | MOT
    LIRE CAR
ECRIRE MOT
```

Le bloc ENVERS est répété aussi longtemps que CAR reste différent de "l'espace" (ou "blanc"), à ne pas confondre avec la chaîne vide qui sert à initialiser MOT.
L'instruction LIRE CAR, dernière instruction du bloc, met à jour après chaque répétition, la condition CAR < > " ␣".

De même qu'il fallait initialiser I à 1 puis incrémenter I de 1 dans le premier exemple, il faut ici une première lecture de CAR puis une lecture courante.

Noter que si la première lecture fournit un espace, le bloc ENVERS ne sera pas exécuté, le MOT affiché sera alors vide.

Noter également le procédé de concaténation à gauche de CAR avec MOT

1.3 Structures répétitives emboitées

Dans une structure répétitive, une des instructions peut encore être une structure répétitive : on peut emboiter les structures répétitives. Par contre, elles ne peuvent se chevaucher. Voici deux exemples de structures répétitives emboitées, nous en verrons beaucoup d'autres quand nous aurons introduit la notion de tableau. (Cf. §2 de ce dossier)

Exemples :

* - Table de multiplication pour les entiers de 1 à 10

	*** TABLE MULTIPLICATION**
I(entier) : désigne la ligne J(entier) : désigne la colonne	. Pour I de 1 A 10 REPETER * LIGNE . POUR J DE 1 A 10 REPETER * ECRIS . ECRIRE I * J, . ECRIRE "LIGNE SUIVANTE"

* - Moyennes de classe

	*** MOYENNES**
CUMCLASS(réel) : total des notes de la classe NOM(chaîne) : nom de l'élève NOTE(réel) : note(s) de l'élève CUMELEV (réel) : total de l'élève ANOM(chaîne) : nom de l'élève précédemment traité NBRE(entier) : nombre de notes de l'élève NBRETOT(entier) : nombre de notes de la classe	CUMCLASS ← Ø ; INBRETOT ← Ø LIRE NOM, NOTE TANT QUE NOM< > "FIN" REPETER * CUMUL ELEVES NOMBRE ← Ø CUMELEV ← Ø ANOM ← NOM TANT QUE ANOM = NOM REPETER * MEME ELEVE NBRE ← NBRE + 1 CUMELEV ← CUMELEV + NOTE ANOM ← NOM LIRE NOM, NOTE

suite page 3.6.

```
                CUMCLASS ← CUMCLASS + CUMELEV
                NBRETOT ← NEBRETOT + NBRE
                ECRIRE "MOYENNE DE L'ELEVE"
                ECRIRE CUMELEV/NBRE
            ECRIRE "MOYENNE DE LA CLASSE"
            ECRIRE CUMCLASS/NBRETOT
```

Le programme s'interrompt quand on saisit l'élève "FIN". La condition ANOM = NOM est fausse quand on a lu un nouveau nom différent du précédent. Il est donc possible d'avoir un nombre variable de notes par élève et un nombre inconnu d'élèves. Il faut cependant au moins une note par élève pour éviter les divisions par zéro lors de l'affichage des moyennes.
Seul le décalage d'écriture permet de délimiter le bloc devant être répété. Ainsi CUMCLASS←CUMCLASS + CUMELEV est répétée sous le contrôle du seul premier TANT QUE du programme MOYENNES.

2. Les données structurées en tableaux

Un **tableau** est une collection de variables de même type rangées en mémoire de façon à être systématiquement repérées par un ou plusieurs **indices.**

Cette collection a un nom, le nom du tableau, les variables sont les éléments du tableau : leur position est déterminée par les valeurs numérique des indices.

Exemples :

(i) : un vecteur X de coordonnées $X_1, \ldots, X_n$

(ii) : un polynôme P de coefficients $a_\emptyset, \ldots, a_n$

(iii) : une matrice A de coefficient a_{ij} $i = 1,\ldots,n$ $j = 1,\ldots,n$

(iv) : la liste des élèves d'une classe

(v) : une liste de réponses par vrai ou faux à un test de 20 questions

Déclaration du tableau

Nous devons spécifier le nombre d'éléments d'un tableau avant de l'exploiter (nous demandons à l'ordinateur de réserver un certain nombre de zones mémoires indicées) et nous précisons le type (cf. §1.2 dossier 1) des éléments du tableau :

Exemples :

(i) X(tableau réel [1,N]) : composantes d'un vecteur

(v) REP(tableau booléen [1,20]) : réponses du test

Ainsi, dans l'exemple (i), une composante du vecteur X est notée X(I) où I désigne l'indice qui permet de donner la position de l'élément dans le tableau.

* - Calcul de somme pondérée

	*** SOMME PONDEREE**
N(entier) : nombre de notes par élèves M(entier) : nombre d'élèves A(tableau réel [1,N]) : coefficients de pondération S(tableau réel [1,M] : sommes pondérées X(réel) : note entrée J(entier) : indice du tableau S I(entier) : indice du tableau A	. LIRE N,M . LIRE A . POUR J DE 1 A N REPETER * ELEVE J . S(J)←∅ . POUR I DE 1 A N REPETER * AJOUT . LIRE X . S(J)← S(J) + X * A(I) . ECRIRE S(J)

Dans cet exemple LIRE A correspond à la saisie des N composantes du tableau. On aurait pu écrire :

```
POUR I DE 1 A N  REPETER
    LIRE A(I)
```

* - Calculer pour une matrice A donnée, le cumul par ligne et colonne de ses coefficients.

	* CUMULS
M(entier) : nombre de lignes N(entiers) : nombre de colonnes A(tableau réel [1,M] * [1,N]) : coefficients de la matrice donnée S(tableau réel [1,M]) : cumul de ligne T(tableau réel [1,N]) : cumul de colonne I(entier) : indice des tableaux S et A J(entier) : indice des tableaux T et A	. LIRE M,N . POUR J DE 1 A N REPETER * INIT T(J)← Ø . POUR I de 1 A N REPETER * TRAITER . S(I)← Ø . POUR J DE 1 A N REPETER * CUMUL . LIRE A(I,J) . S(I)← S(I) + A(I,J) . T(J)← T(J) + A(I,J) * EDITIONS

Ici, les éléments A(I,J) sont lus un à un et immédiatement cumulés dans les lignes S(I) et les colonnes T(J) correspondantes.

(iv) une liste d'élèves pourra être décrite par ELEVE(**tableau chaîne** [1,5Ø]) par exemple

ELEVE(I) désignera alors la chaîne de caractères nom de l'élève. Certains langages (PASCAL par exemple) considèrent cette chaîne elle-même comme un **tableau de caractères** ; on accède alors au Je caractère du nom par ELEVE(I,J).

3. Réalisation en BASIC

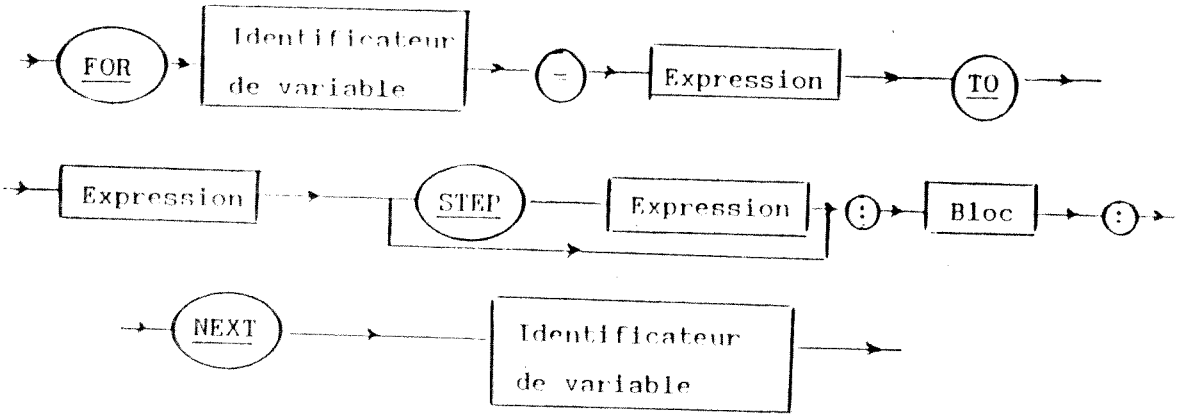
3.1 Instruction **FOR NEXT** (structure itérative à bornes définies)

La structure

POUR ⌴ DE ⌴ A ⌴ PAS ⌴ REPETER

‖ * BLOC

se traduit en BASIC par :



(Pour cette instruction ainsi qu'au § 3.2, les : peuvent être remplacés par un passage à une nouvelle ligne.)

```
10 REM EXEMPLE 1:EDITION D'UNE TABLE
20 INPUT "TAILLE DE LA TABLE";N
30 PRINT "I","I*I","CUBE","RACINE CARREE"
40 FOR I=1 TO N
50      REM * LIGNE
60      C=I*I
70      PRINT I,C,I*C,SQR(I)
80 NEXT I
90 END
```

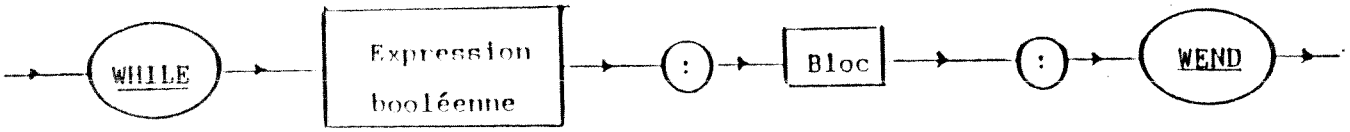
Nous voyons que l'instruction **NEXT** ␣ précise que la suite des instructions à répéter est terminée, et que l'on peut passer à la valeur suivante de la variable de contrôle en l'incrémentant du pas.

```
10 REM EXEMPLE 2
20 INPUT "NOMBRE D'ENTIERS A ENTRER";N
30 S=0:T=0
40 FOR K=1 TO N
50      REM * SOMMES
60      INPUT "ENTRER UN ENTIER";M
70      S=S+M:T=T+M*M
80 NEXT K
90 PRINT "SOMME DES ENTIERS";S
100 PRINT "SOMME DES CARRES DES ENTIERS";T
110 END
```

3.2 Instruction WHILE...WEND (Structure itérative "Tant que")

```
10 REM EXEMPLE 3
20 INPUT "ENTRER LA VALEUR DESIREE";VALEUR
30 S=0:I=1
40 WHILE S<VALEUR
50       REM * AJOUTER
60       S=S+1/I
70       I=I+1
80 WEND
90 PRINT "ON S'ARRETE AU";I;"EME TERME"
100 PRINT "LA VALEUR APPROCHEE EST";S
110 END
```

En BASIC, la structure Tant que s'écrit :



WEND joue ici le rôle de **NEXT** : il précise que la suite des instructions à répéter est terminée.

Remarquons que cette instruction n'existe pas sur tous les micro-ordinateurs :

- Simulation du WHILE...WEND

Lorsqu'on ne dispose pas de l'instruction WHILE, le palliatif suivant peut être utilisé :

(n) IF Condition, THEN GOTO (n_3)

(n_1) Bloc exécuté tant que condition fausse

(n_2) GOTO (n)

(n_3) suite du programme

Le Bloc (n1) — (n2) sera répété tant que la condition de la ligne (n) sera fausse. Quand elle sera vraie on saute à la suite du programme

```
10 REM Exemple 3 BIS
20 INPUT "Entrer valeur maxi "!VALEUR
30 S=0!I=1
40 IF S>=VALEUR THEN 90
50   REM AJouter
60   S=S+1/I
70   I=I+1
80 GOTO 40
90 PRINT "On s'arrete au "!I!"eme terme"
100 PRINT "La valeur approchee est "!S
110 END
```

3.3 Structures emboitées

L'affichage de la table de multiplication présentée dans l'exemple 4 se traduit en BASIC de la manière suivante :

```
10 REM TABLE DE MULTIPLICATION
20 FOR I=1 TO 10
30      REM * LIGNE
40      FOR J=1 TO 10
50          REM * ECRIS
60          PRINT I*J;"*";
70      NEXT J
80      PRINT
90 NEXT I
100 END
```

3.4 Instructions DIM (déclaration de tableaux)

DIM X(1ØØ)	déclare un tableau réel de 1Ø1 zones x(Ø), X(1), ... X(1ØØ)
DIM X(N)	déclare un tableau réel de N + 1 zones (où N a été précédemment renseigné)
DIM NOM$(5Ø)	déclare un tableau de 51 chaînes de caractères
DIM A(5Ø,1ØØ)	déclare un tableau à deux dimensions (51 lignes et 1Ø1 colonnes) dont l'élément courant sera désigné par A(I,J).

le tableau

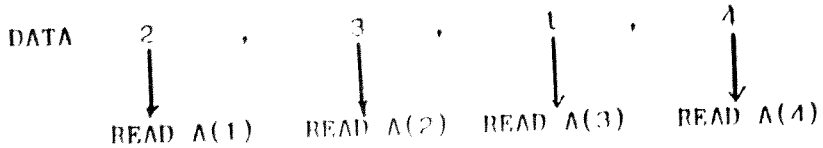
Certains BASIC initialisent lors de la rencontre de l'instruction DIM
- à zéro pour les tableaux numériques
- à blanc pour les tableaux de chaînes de caractères

Ne pas confondre DIM X(1ØØ) et PRINT X(1ØØ). Cette dernière instruction affiche la composante d'indice 1ØØ du tableau X réservé par l'instruction DIM.
Le type du tableau est implicitement donné par son nom
X(I) est réel, NOM$(K) est du type chaîne

3.5 Instruction READ ... DATA Lectures de tables

```
10 REM EXEMPLE 5:SOMME PONDEREE DE 4 NOTES
20 INPUT "NOMBRE D'ELEVES DANS LA CLASSE";M
30 DIM A(4),S(M)
40 FOR K=1 TO 4:READ A(K):NEXT K
50 FOR J=1 TO M
60     REM * ELEVE J
70     S(J)=0
80     FOR I=1 TO 4
90         REM * AJOUT
100        INPUT "NOTE";X
110        S(J)=S(J)+A(I)*X
120    NEXT I
130    PRINT "ELEVE",J,S(J)
140 NEXT J
150 END
160 DATA 2,3,1,4
```

Dans ce programme, nous avons utilisé, pour entrer le tableau des coefficients, les instructions **DATA et READ.** Ces instructions sont très pratiques en BASIC pour rentrer les éléments d'un tableau de constantes du programme, sans les saisir au clavier.
La table **DATA** 2, 3, 1, 4 est consultée à chaque occurence de l'instruction **READ**, elle est donc équivalente aux instructions :



A(1) = 2
A(2) = 3
A(3) = 1
A(4) = 4

Les éléments de la table doivent être séparés par des virgules, et la ligne commence par le mot réservé DATA. Un pointeur se déplace dans la table et avance d'un cran à chaque instruction READ.
Remarquons qu'il est équivalent d'écrire :

60 DATA 1,2,3
70 DATA 4,5,6 ou 60 DATA 1,2,3,4,5,6

Une même table peut être consultée plusieurs fois. Si l'on veut se repositionner au début de la table, il faut insérer l'instruction **RESTORE**

exemple : Si l'on rajoute dans le programme précédent
162 RESTORE
164 READ A
166 PRINT A

la machine affichera 2. En effet, le pointeur est repositionné au début de la table. L'instruction READ A équivaut donc à A = 2.

Remarques :

La table DATA peut être placée à n'importe quel endroit dans le programme. Le nombre d'éléments dans la table doit être supérieur ou égal au nombre d'instructions READ figurant dans le programme, sinon une erreur sera signalée.

Dans une table on peut mélanger les types :
exemple : DATA 2, A, 6.3
En effet, le type de l'élément se reconnaît à l'instruction READ.
Nous aurons par exemple :
READ A : READ A$: READ C
Ces instructions sont compatibles avec la table précédente et sont équivalentes à A = 2, A$ = "AB", C = 2.3

On peut ainsi consulter une table de noms :

```
.
.
30 FOR I = 1 TO 4 : READ NOM$(I) : NEXT I
.
.
.
100 DATA "JEAN", "SOIE", "ALPHA", "GAMMA 12"
```

3.6 Autres exemples de programmes BASIC

```
0010 REM MOYENNES
0020 CUMCLASS=0:NBRETOT=0
0030 INPUT NOTE,NOM$
0040 WHILE NOM$<>"FIN"
0050   NBRE=0:CUMELEV=0
0060   ANOM$=NOM$
0070   WHILE ANOM$=NOM$
0080     NBRE=NBRE+1
0090     CUMELEV=CUMELEV+NOTE
0100     ANOM$=NOM$
0110     INPUT NOTE,NOM$
0120   WEND
0130 CUMCLASS=CUMCLASS+CUMELEV
0140 NBRETOT=NBRETOT+NBRE
0150 PRINT "MOYENNE ELEVE ",CUMELEV/NBRE
0160 WEND
0170 PRINT "MOYENNE CLASSE ",CUMCLASS/NBRETOT
0180 END
```

```
20 INPUT "NOMBRE DE LIGNES";M
30 INPUT "NOMBRE DE COLONNES";N
40 DIM A(M,N),S(M),T(N)
50 FOR J=1 TO N
60     REM * INIT
70     T(J)=0
80 NEXT J
90 FOR I=1 TO M
100    REM * TRAITER
110    S(I)=0
120    FOR J=1 TO N
130        REM * CUMUL
140        INPUT A(I,J)
150        S(I)=S(I)+A(I,J)
160        T(J)=T(J)+A(I,J)
170 NEXT J,I
180 REM * EDITIONS
190 FOR I=1 TO M:PRINT S(I);:NEXT I:PRINT
200 FOR J=1 TO N:PRINT T(J);:NEXT J
210 END
```

Liste des programmes en L.S.E.

```
10 * EXEMPLE 1:EDITION D'UNE TABLE
20 LIRE[/,'TAILLE DE LA TABLE']N
30 AFFICHER[/,'I',8X,'I*I',8X,'CUBE',8X,'RACINE CARREE',/]
40 FAIRE 70 POUR I_1 JUSQUA N
50            * LIGNE
60            C_I*I
70            AFFICHER[U,8X,U,8X,U,8X,U,/]I,C,I*C,RAC(I)
80 TERMINER
```

```
10 * EXEMPLE 2
20 LIRE[/,'NOMBRE D''ENTIERS A ENTRER']N
30 S_0;T_0
40 FAIRE 70 POUR K_1 JUSQUA N
50            * SOMMES
60            LIRE[/,'ENTRER UN ENTIER']M
70            S_S+M;T_T+M*M
80 AFFICHER 'SOMME DES ENTIERS:',S
90 AFFICHER 'SOMME DES CARRES DES ENTIERS:',T
100 TERMINER
```

```
110 * EXEMPLE 3
120 LIRE[/,'ENTRER LA VALEUR DESIREE']VAL
130 S_0;I_1
140 FAIRE 170 TANT QUE S<VAL
150             * AJOUTER
160             S_S+1/I
170             I_I+1
180 AFFICHER 'SOMME DES ENTIERS:',S
190 AFFICHER 'ON S''ARRETE AU',I,'EME TERME'
200 AFFICHER 'LA VALEUR APPROCHEE EST',S
210 TERMINER
```

```
10 * TABLE DE MULTIPLICATION
20 FAIRE 65 POUR I_1 JUSQUA 10
30           * LIGNE
40           FAIRE 60 POUR J_1 JUSQUA 10
50                    * ECRIS
60                    AFFICHER[U,2X]I*J
65           AFFICHER[/]
70 TERMINER
```

```
10 * SOMME PONDEREE
20 LIRE[/,'NOMBRE DE NOTES PAR ELEVE']N
30 LIRE[/,'NOMBRE D''ELEVES DANS LA CLASSE',U,/]M
40 TABLEAU A[N],S[M]
50 LIRE A
60 FAIRE 130 POUR J_1 JUSQUA M
70           * ELEVE J
80           S[J]_0
90           FAIRE 120 POUR I_1 JUSQUA N
100                    * AJOUT
110                    LIRE[/,'NOTE']X
120                    S[J]_S[J]+A[I]*X
130          AFFICHER[/,'ELEVE',2X,U,4X,U]J,S[J]
140 TERMINER
```

```
 10 * EXEMPLE 6
 20 LIRE[/,'NOMBRE DE LIGNES']M
 30 LIRE[/,'NOMBRE DE COLONNES']N
 40 TABLEAU A[M,N],S[M],T[N]
 50 FAIRE 70 POUR J_1 JUSQUA N
 60           * INIT
 70           T[J]_0
 80 FAIRE 150 POUR I_1 JUSQUA M
 90            * TRAITER
100            S[I]_0
110            FAIRE 150 POUR J_1 JUSQUA N
120                     * CUMUL
130                     LIRE A[I,J]
140                     S[I]_S[I]+A[I,J]
150                     T[J]_T[J]+A[I,J]
160 AFFICHER S
170 AFFICHER T
180 TERMINER
```

Liste de programme en PASCAL

```
PROGRAM EDITION;
VAR I,C,N:INTEGER;
BEGIN
  WRITELN('TAILLE DE LA TABLE');READ(N);
  WRITELN('I ',' I*I ','I*I*I',' RAC(I)');
  FOR I:=1 TO N DO
  BEGIN
    C:=I*I;
    WRITELN(I:2,C:4,I*C:6,SQRT(I))
  END
END.
```

```
PROGRAM HARMO;
VAR I:INTEGER;
    S,VAL:REAL;
BEGIN
  WRITELN('VALEUR DESIREE ');READ(VAL);
  S:=0;I:=1;
  WHILE S<VAL DO
  BEGIN
    S:=S+1/I;I:=SUCC(I)
  END;
  WRITELN('ARRET AU ',I,' EME TERME');
  WRITELN('VALEUR APPROX. ',S)
END.
```

```
PROGRAM SOMME;
VAR M,N,K,S,T:INTEGER;
BEGIN
  WRITELN('NOMBRE D"ENTIERS A TRAITER ');
  READ(N); S:=0;T:=0;
  FOR K:=1 TO N DO
  BEGIN
    WRITELN('ENTRER UN ENTIER ');READ(M);
    S:=S+M;T:=T+M*M
  END;
  WRITELN('SOMME DES ENTIERS ',S);
  WRITELN('SOMME DES CARRES ',T)
END.
```

```
PROGRAM CLASS;
VAR
  CUMCLASS,CUMELEV,NOTE,NBRE,NBRETOT:INTEGER;
  NOM,ANOM:STRING ;
BEGIN
  CUMCLASS:=0;NBRETOT:=0;
  READLN(NOTE,NOM);
  WHILE NOM<>'FIN' DO
  BEGIN
    NBRE:=0;
    CUMELEV:=0;
    ANOM:=NOM;
    WHILE ANOM=NOM DO
    BEGIN
      NBRE:=NBRE+1;
      CUMELEV:=CUMELEV+NOTE;
      ANOM:=NOM;
      READLN(NOTE,NOM)
    END;
    CUMCLASS:=CUMCLASS+CUMELEV;
    NBRETOT:=NBRETOT+NBRE;
    WRITELN('MOYENNE ELEVE ',CUMELEV/NBRE)
  END;
  WRITELN('MOYENNE CLASSE ',CUMCLASS/NBRETOT)
END.
```

```
PROGRAM CUMULS;
VAR A:ARRAY[1..10,1..10] OF INTEGER;
    S,T:ARRAY[1..10] OF INTEGER;
    I,J,N,M:INTEGER;
BEGIN
  WRITELN('DONNER M,N ');READ(M,N);
  FOR J:=1 TO N DO T[J]:=0;
  FOR I:=1 TO M DO
  BEGIN
    S[I]:=0;
    FOR J:=1 TO N DO
    BEGIN
      READ(A[I,J]);
      S[I]:=S[I]+A[I,J];
      T[J]:=T[J]+A[I,J]
    END;
  END;
  FOR I:=1 TO M DO
  BEGIN
    FOR J:=1 TO N DO WRITE(A[I,J]:4);WRITELN(' * ',S[I])
  END;WRITELN;
  FOR J:=1 TO N DO WRITE(T[J]:4);WRITELN(' * ')
END.
```

Variante du programme PASCAL CUMULS, utilisant un tableau à deux indices définis comme tableau de tableau

```
PROGRAM CUMULS;
VAR A:ARRAY[1..10] OF ARRAY[1..10] OF INTEGER;
    S,T:ARRAY[1..10] OF INTEGER;
    I,J,N,M:INTEGER;
BEGIN
  WRITELN('DONNER M,N ');READ(M,N);
  FOR J:=1 TO N DO T[J]:=0;
  FOR I:=1 TO M DO
  BEGIN
    S[I]:=0;
    FOR J:=1 TO N DO
    BEGIN
      READ(A[I][J]);
      S[I]:=S[I]+A[I][J];
      T[J]:=T[J]+A[I][J]
    END;
  END;
  FOR I:=1 TO M DO
  BEGIN
    FOR J:=1 TO N DO WRITE(A[I][J]:4);WRITELN(' * ',S[I])
  END;WRITELN;
  FOR J:=1 TO N DO WRITE(T[J]:4);WRITELN(' * ')
END.
```

MINI T.P. 3

0. **Calcul de la racine carrée d'un nombre par itération (Méthode de Newton)** : calculer $\sqrt{a}$ en utilisant le fait que si b est une valeur approchée de $\sqrt{a}$ par défaut $\frac{a}{b}$ est une valeur approchée de $\sqrt{a}$ par excès, donc $\frac{1}{2}(b + \frac{a}{b})$ est une meilleure approximation de $\sqrt{a}$. On calcule donc la suite $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$. On prend comme premier terme $x_0 = \frac{1+a}{2}$, les itérations s'arrêtent lorsque

$$\left| \frac{x_{n+1} - x_n}{x_n} \right| < \varepsilon$$

Comparer au résultat obtenu en utilisant la fonction standard racine carré de a.

1. Entrer un mot ayant un nombre impair de lettres. Réaliser sur l'écran un losange formé à l'aide des lettre de ce mot selon le modèle suivant

R

CRA

ECRAN

CRA

R

2. Il s'agit de lire un texte, composé de mots séparés par un blanc et de générer par programme un nouveau texte dans lequel les mots sont codés par retournement (BONJOUR donne RUOJNOB) les espaces restant à leur place entre les mots.

3. Entrer une liste de noms, une liste de verbes d'état, une liste d'adjectifs. Ecrire un algorithme permettant de choisir au hasard un mot dans une liste donnée. (Utiliser READ....DATA).
Ecrire un algorithme permettant de générer des phrases aléatoires dont la forme est [nom][verbe d'état][adjectif]

Variante : Entrer (sous forme DATA) un certain nombre de préfixes et leur signification (par exemple PARA, qui protège de) et des noms associés, précédés de leur article (p. ex. : PLUIE, la pluie). Ranger les éléments dans 4 tableaux. Afficher de façon aléatoire un certain nombre de lignes formées de : un préfixe suivi d'un mot et la définition du nouveau mot obtenu. Par ex. : PARAPLUIE qui protège de la pluie.

4. Etablir un tableau d'amortissement pour un prêt remboursé à mensualités constantes :

* Lire en entrée : le capital emprunté C

le taux d'intérêt annuel τ

le montant T de la mensualité versée.

Editer le tableau suivant :

DATE ECHEANCE	CAPITAL EN DEBUT DE PERIODE	DECOMPOSITION CAPITAL	ECHEANCE INTERETS	TERME DE REMBT. CONSTANT
n	X_n	$X_n - X_{n+1}$	$(\tau/12).X_n$	T

où $T = (X_n - X_{n+1}) + \tau/12 * X_n$

(part de capital remboursé ; intérêts courants

Le tableau s'arrête quand X_n devient inférieur ou égal à T. On affiche alors une dernière échéance avec le solde du prêt

* **Variante** (durée fixe)

Au lieu de lire T on lit N, nombre de mensualités prévues. T se calcule alors par la formule :

$$T = C(1+\tau/12)^N \cdot \frac{\tau}{12} \cdot \frac{1}{(1+\tau/12)^N - 1}$$

Exemple : C = 1ØØØØØF $\tau = \frac{12,65}{1ØØ}$ N = 12Ø T = 1472.53

DATE ECHEANCE	CAPITAL EN DEBUT DE PERIODE	DECOMPOSITION CAPITAL	ECHEANCE INTERETS	TERME DE REMBT. CONSTANT
ØØ1	1ØØ.ØØØ,ØØ	418,36	1.Ø54,17	1.472,53
ØØ2	99.581,64	422,77	1.Ø49,76	1.472,53
ØØ3	99.158,87	427,23	1.Ø45,3Ø	1.472,53
ØØ4	98.731,64	431,73	1.Ø4Ø,8Ø	1.472,53
ØØ5	98.299,91	436,29	1.Ø36,24	1.472,53

ANNEXE 3 - LE CODE ASCII

Pour être transmis, chaque caractère ou fonction représentée au clavier est codée. Toute donnée qui entre ou sort du microordinateur est stockée sous forme binaire sur un octet. Le code le plus usuel est le code ASCII (abréviation de American Standard Code for Information Interchange) de 0 à 127 (ou 255 selon les machines). Exemple :Le code de "A" est 65 ou ØlØØ ØØØl

le code de retour de chariot CR est 13 ou ØØØØllØl

On peut obtenir le code ASCII d'un caractère en BASIC en utilisant la fonction standard ASC(A$) qui fournit le code ASCII du premier caractère de la chaine A$. A l'inverse, la commande CHR$( ) fournit le caractère correspondant du code ASCII fourni.

Exemple

Calcul du nombre d'occurences de chaque lettre de l'alphabet dans un texte. On utilisera un tableau T de 26 composante : T(i) représente le nombre d'occurences de la i-ème lettre de l'alphabet. Nous nous limiterons pour l'instant à une chaine ne comportant que des lettres de l'alphabet sans ponctuation ni blanc : nous pourrons plus tard écrire un programme sans ces restrictions.

	*** EXEMPLE OCCURENCES**
T(tableau entier [1,26]) : nombre d'occurences de chaque lettre. I(entier) : indice variant de 1 à la longueur de la chaine de caractères entrée. L(chaine) : i-ème caractère de la chaine. J(entier) : indice du tableau T TEXTE(chaine): chaine de caractères entrée	. LIRE TEXTE . POUR J DE 1 A 26 REPETER * INIT . T(J) ← Ø . POUR I DE 1 A LGR(TEXTE) REPETER * LETTRE I . L ← Ième LETTRE DE TEXTE . L ← T(CODE ASC(L)-64)+1 . POUR J DE 1 A 26 REPETER * AFFICHAGE . Ecrire "FREQUENCE DE", caract. dont le code ASC est 64 + J, T(J)

Voici la traduction de cet algorithme en BASIC :

```
10 REM EXEMPLE:OCCURENCES
20 DIM T(26)
30 INPUT "ENTRER LE TEXTE";TEXTE$
40 FOR I=1 TO LEN(TEXTE$)
50     REM * LETTRE I
60     L$=MID$(TEXTE$,I,1)
70     T(ASC(L$)-64)=T(ASC(L$)-64)+1
80 NEXT I
90 FOR J=1 TO 26
100    REM * AFFICHAGE
110    PRINT "FREQUENCE DE",CHR$(64+J),T(J)
120 NEXT J
130 END
```

et deux variantes d'affichage en L.S.E :

```
10 * EXEMPLE:OCCURENCES
20 CHAINE TEXTE
30 LIRE[/,'ENTRER LE TEXTE',/]TEXTE
40 TABLEAU T[26]
50 FAIRE 70 POUR J_1 JUSQUA 26
60           * INIT
70           T[J]_0
80 FAIRE 100 POUR I_1 JUSQUA LGR(TEXTE)
90           * LETTRE I
100          T[EQN(TEXTE,I)-64]_T[EQN(TEXTE,I)-64]+1
110 AFFICHER T
120 TERMINER
```

```
10 * EXEMPLE:OCCURENCES
20 CHAINE TEXTE
30 LIRE[/,'ENTRER LE TEXTE',/]TEXTE
40 TABLEAU T[26]
50 FAIRE 70 POUR J_1 JUSQUA 26
60           * INIT
70           T[J]_0
80 FAIRE 100 POUR I_1 JUSQUA LGR(TEXTE)
90           * LETTRE I
100          T[EQN(TEXTE,I)-64]_T[EQN(TEXTE,I)-64]+1
110 FAIRE 130 POUR J_1 JUSQUA 26
120             * AFFICHAGE
130             AFFICHER[/,'FREQUENCE DE',U,2X,U,/]EQC(64+J),T[J]
140 TERMINER
```

TRADUCTION EN PASCAL (avec une variante originale)

```
PROGRAM OCCURENCES;
VAR T:ARRAY[1..26] OF INTEGER;
    I,J:INTEGER;
    L:CHAR;
    TXT:STRING;
BEGIN
  WRITELN('ENTRER LE TEXTE ');READLN(TXT);
  FOR J:=1 TO N DO T[J]:=0;
  FOR I:=1 TO LENGTH(TXT) DO
  BEGIN
    L:=COPY(TXT,I,1);
    T[ORD(L)-64]:=T[ORD(L)-64]+1
  END;
  FOR J:=1 TO 26 DO
    WRITELN('FREQUENCE DE ',CHR(64+J),T[J];
END.
```

Possibilité, en PASCAL, d'indiquer les tableaux par des caractères alphanumériques (dans l'ordre du code ASCII)

```
PROGRAM OCCURENCES;
VAR T:ARRAY['A'..'Z'] OF INTEGER;
    L:CHAR;
BEGIN
  WRITELN('ENTRER LE TEXTE(Termine par un point)');
  FOR L:='A' TO 'Z' DO T[L]:=0;
  REPEAT
    READ(L);
    IF (L>='A') AND (L<='Z') THEN  T[L]:=T[L]+1;
  UNTIL L='.';
  FOR L:='A' TO 'Z' DO
    WRITELN('FREQUENCE DE ',L,T[L]:4)
END.
```

CODES ASCII

ASCII Code	Caractère	ASCII Code	Caractère	ASCII Code	Caractère
000	NUL	043	+	086	V
001	SOH	044	'	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	^
009	HT	052	4	095	_
010	LF	053	5	096	'
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

DOSSIER 4

CONSTRUCTION DE PROGRAMMES STRUCTURES

4.1 Un peu d'histoire .... de la programmation

```
6000  3E 1A 32 00 86 3E 02 32 01 86 4F 3A 00 86 16 00
6010  00 00 47 91 FA 1B 60 14 C3 12 60 78 32 02 86 7A
6020  32 03 86 C3 00 C4 00 00 00 00 00 00 00 00 00 00
6030  00


8600  1A 02 00 0D 00 00
```

4.1.1 Ceci est un programme en **langage machine**, implanté en mémoire à partir de l'adresse $(6000)_{16}$. Les codes sont présentés en notation hexadécimale octet par octet $(3E)_{16}$ représente 00111110.
Les premiers informaticiens manipulaient ces objets que l'on présentera ensuite sous une forme plus lisible, accompagnée de mnémoniques (Assembleur):

Ce programme effectue la division entière des entiers qui se trouvent à l'adresse $(8600)_{16}$ et $(8601)_{16}$. Le quotient et le reste se trouvent aux adresses $(8602)_{16}$ et $(8603)_{16}$

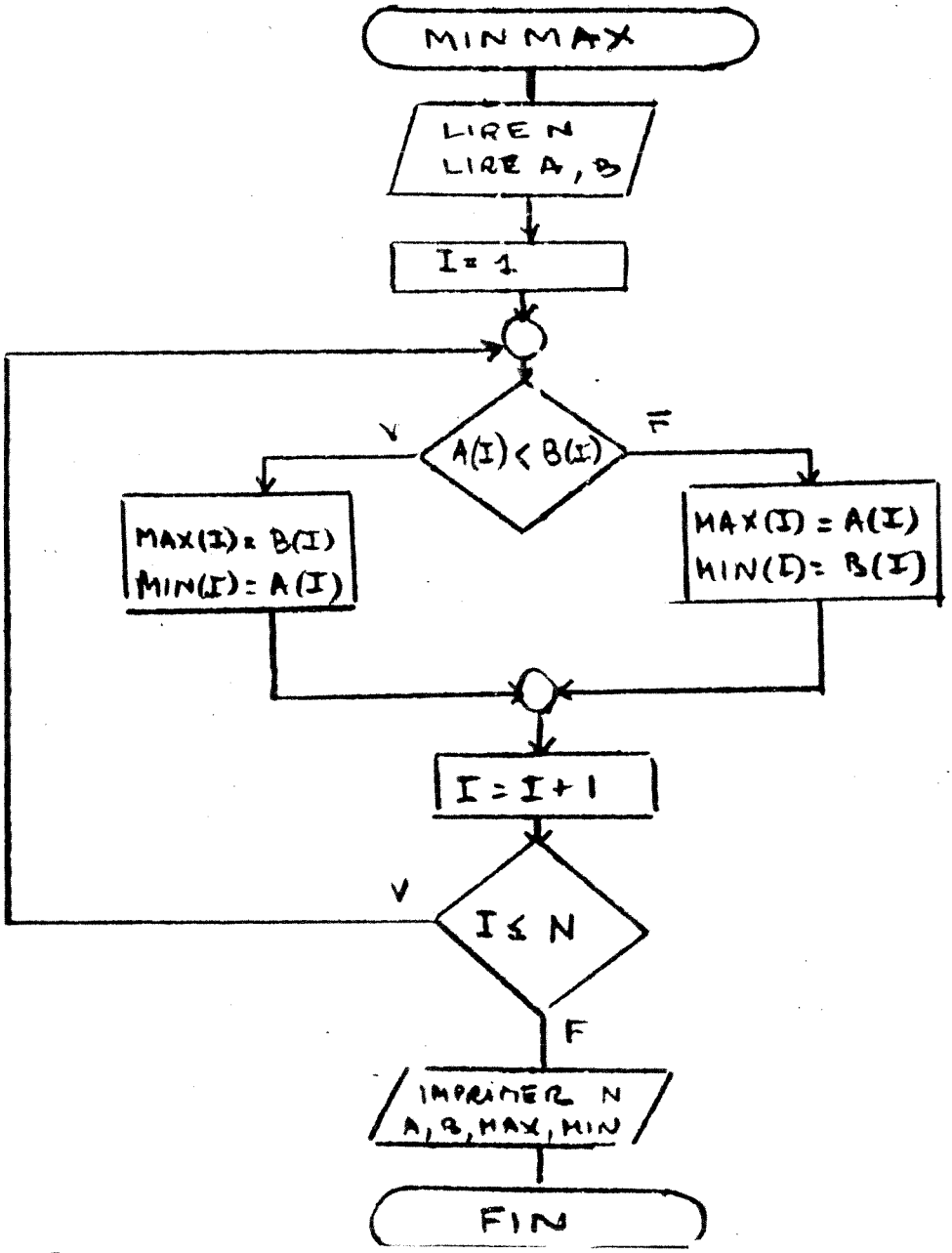
```
**  Assembleur  Z-80  **  Page 01  **

0000
0000                            ORG   6000H
6000 3E1A                       LD    A,1AH
6002 320086                     LD    (8600H),A
6005 3E02                       LD    A,02
6007 320186                     LD    (8601H),A
600A 4F                         LD    C,A
600B 3A0086                     LD    A,(8600H)
600E 1600                       LD    D,0
6010 00                         NOP
6011 00                         NOP
6012 47                 ETIQ:   LD    B,A
6013 91                         SUB   C
6014 FA1B60                     JP    M,SUITE
6017 14                         INC   D
6018 C31260                     JP    ETIQ
601B 78                 SUITE:  LD    A,B
601C 320286                     LD    (8602H),A
601F 7A                         LD    A,D
6020 320386                     LD    (8603H),A
6023 C300C4                     JP    C400H
6026                            END
```

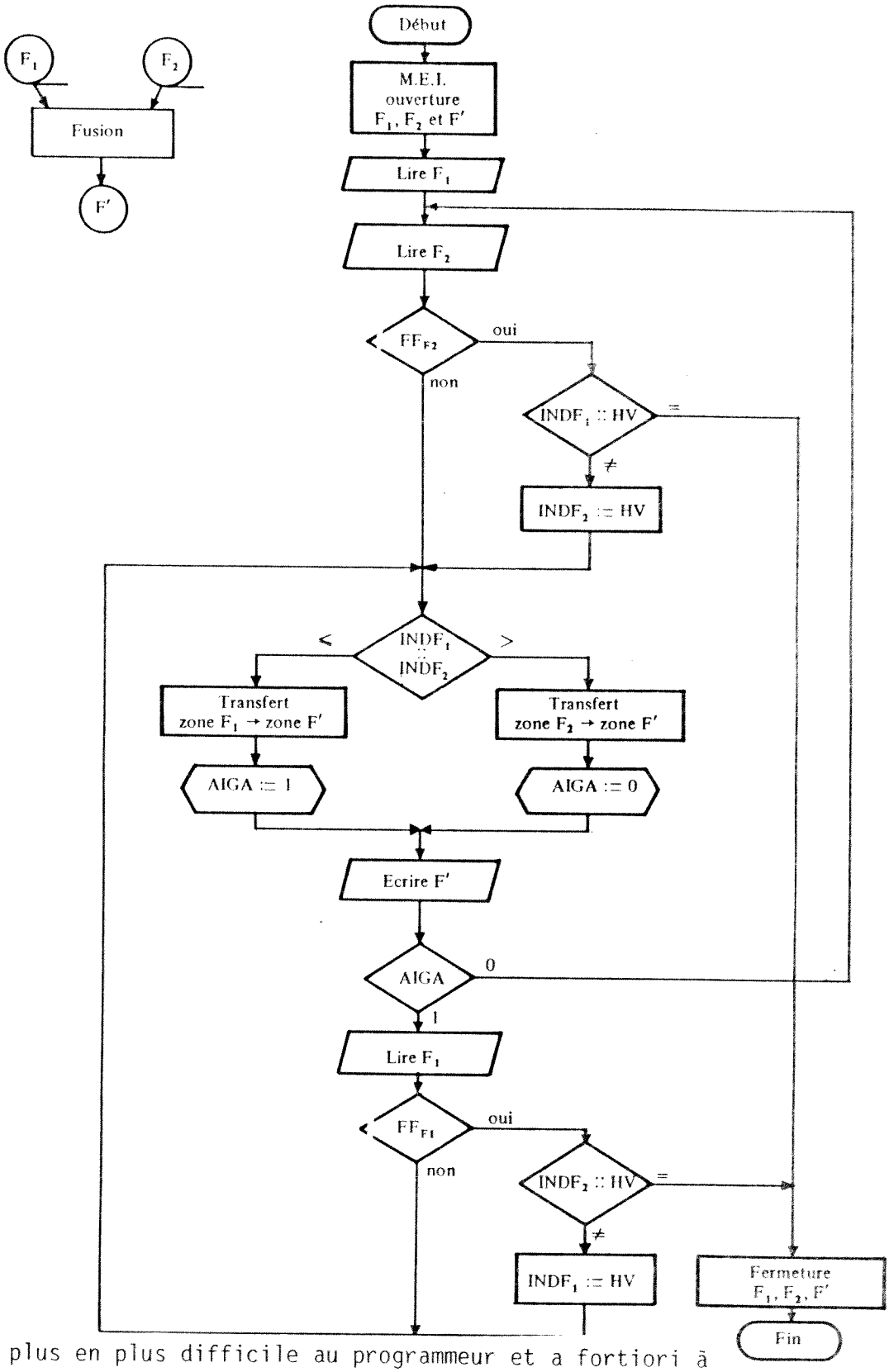
Les outils sont sommaires :

- chargements-transferts LD (load)
- soustraction ou addition SUB ou ADD
- sauts JP (jump) pouvant être conditionnels

4.1.2 L'apparition de langages plus évolués (FORTRAN) permet le maniement de variables ou de tableaux et des opérateurs arithmétiques et logiques. Le déroulement du programme est représenté par un **organigramme** du type suivant :



4.1.3 L'information de gestion élargit le champ d'action des ordinat aux **fichiers** sur supports magnétiques. L'organigramme ci-dessous décrit la fusion (interclassement) de deux fichiers.



Il est de plus en plus difficile au programmeur et a fortiori à ceux qui manipulent ces programmes de suivre ou de retrouver la logique sous-jacente, notamment en vue de faire des modifications

dans les programmes initiaux, pour tenir compte de souhaits ou d'impératifs nouveaux des utilisateurs. La **maintenance** des programmes devient couteuse et hasardeuse.

4.1.4 Les années 7Ø voient naître les premières méthodes de programmation visant à codifier les préparatifs et la démarche algorithmique. La méthode dite de **l'arbre programmatique** ainsi que **L.C.P.** (Langage de construction de programmes) se généralisent dans les grands services informatiques.

Elles visent à dégager dans tout programme à écrire, un assemblage des structures simples étudiées dans les dossiers précédents :

1. la **séquence simple** (ou BLOC) identifiée par un nom
2. la **structure alternative** (BLOC) exécutée conditionnellement.

```
Si  condition  ALORS  *Bloc OUI
               SINON  *Bloc NON
```

ou **l'affectation conditionnelle** de valeur à une variable :

```
variable = SI condition ALORS valeur 1
                        SINON valeur 2
```

3. **la structure répétitive**

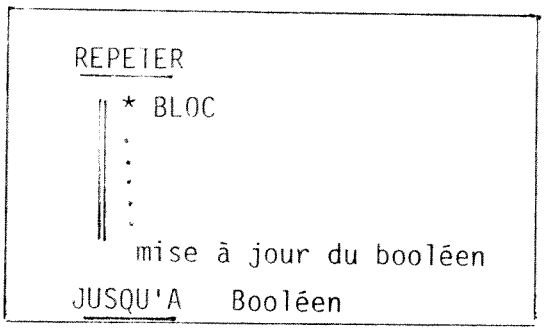
- à indice

```
. POUR indice DE val A val PAS nombre
  REPETER
    ‖ * BLOC REPETE
```

- à expression booléenne

```
. initialisation du Booléen
. TANT QUE booléen
  REPETER
    ‖ * Bloc REPETE
    ‖   .
    ‖   .
    ‖   Mise à jour du booléen
```

- et sa variante (exécution au moins une fois du bloc)



Pour dégager ces structures, il convient d'analyser le problème en le décomposant en problèmes plus simples (MODULES) et au sein d'un module, en partant des résultats pour remonter vers les données.

4.2 Analyse de programmes

4.2.1 Analyse par modules

Exemple :

Fusion de deux tableaux triés.

A partir de deux tableaux T1(tableau [1,N] d'entiers) et T2 (tableau [1,M] d'entiers), triés dans l'ordre croissant, construire le tableau T3 (tableau [1,M+N] d'entiers) fusionnant dans l'ordre croissant les deux tableaux T1 et T2.

Principe :

. Tant qu'aucun des deux tableaux n'est épuisé :
- on avance dans T1 tant que le prochain élément de T2 est plus grand
- on avance dans T2 tant que le prochain élément de T1 est plus grand.

. On complète par les éléments du tableau restant éventuel.

On peut alors faire articuler le programme autour des modules suivants :

. INITIALISATIONS
. AVANCE.T1
. AVANCE.T2
. EDITIONS

Principal

N,M(entiers) : dimensions de T1 et T2
I,J(entiers) : pointeurs de T1 et T2
K(entier) : pointeur de T3
T1(tableau[1,N] d'entiers)
T2(tableau[1,M] d'entiers)
T"(tableau[1,M+N] d'entiers)

*** FUSION**

```
EXECUTER  INITIALISATIONS
TANT QUE I≤N ET J≤M  REPETER
|| * PARCOURS 1
|| SI T1(I)<T2(J) ALORS EXECUTER
||                  || AVANCE. T1
||                  SINON EXECUTER
||                  || AVANCE. T2                    (1)
TANT QUE I≤N  REPETER
|| EXECUTER AVANCE. T1                               (2)
TANT QUE J≤M  REPETER
|| EXECUTER AVANCE. T2                               (3)
EXECUTER EDITIONS                                    (4)
```

Modules

*** INITIALISATIONS**

```
LIRE N,M
POUR I DE 1 A N  REPETER
|| LIRE T1(I)
POUR J DE 1 A M  REPETER
|| LIRE T2(J)
I ← 1   J ← 1   K ← 0
```

*** AVANCE. T1**

```
K ← K + 1
T3(K) ← T1(I)
I ← I + 1
```

*** AVANCE. T2**

```
K ← K+1
T3(K) ← T2(J)
J ← J + 1
```

* EDITIONS

Cette méthode permet une programmation progressive du niveau le plus général au plus fin et une modification des modules sans incidence sur la structure générale du programme.

4.2.2. Notions de niveau de structuration

Le niveau de départ (niveau 1) est celui d'un bloc simple d'instructions exécutables en séquence. L'apparition d'une instruction de contrôle du type "si... sinon" ou "tant que" donne naissance à 1 ou 2 blocs d'instruction de niveau 2 auxquels on donnera un nom et que l'on présentera de façon décalée par rapport au niveau 1. Il en sera de même au sein de chacun des blocs de niveau 2 si l'on utilise à nouveau une instruction de contrôle qui donnera naissance à 1 ou 2 blocs de niveau 3 etc....

4.2.3. Parcours d'une structure

Un programme sera correctement structuré si tout bloc d'instructions d'un niveau donné :

- admet un seul point d'entrée et un seul point de sortie (pas de "renvoi" ou "de saut" à l'extérieur du bloc en cours)
- l'entrée est commandée par une instruction de contrôle appartenant au bloc de niveau précédent, appelé bloc d'appel
- la sortie se fait obligatoirement vers la prochaine instruction du bloc d'appel.

Il en résulte que les différentes structures utilisées ne peuvent être que

consécutives ex : (2) → (3) → (4)

emboitées complètement ex : AVANCE. T1 et AVANCE.T2 emboitées dans PARCOURS (1) elle-même emboitée dans la structure de base FUSION

L'écriture - et la mise au point - d'un programme pourront alors se faire par étapes de finesse :

- écriture et mise au point d'un niveau donné en laissant momentanément vides les blocs appelés par ce niveau
- développement ultérieur de ces derniers.

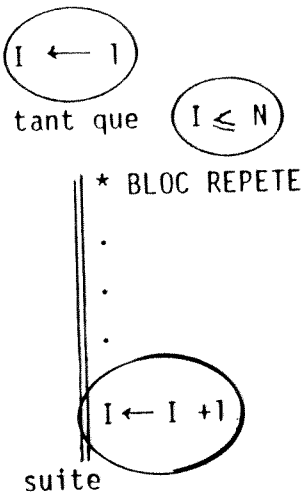
4.2.4 Remarques

Dans les structures répétitives, vérifier le bon avancement de la donnée permettant le contrôle de la répétition :

si c'est un indice-compteur :

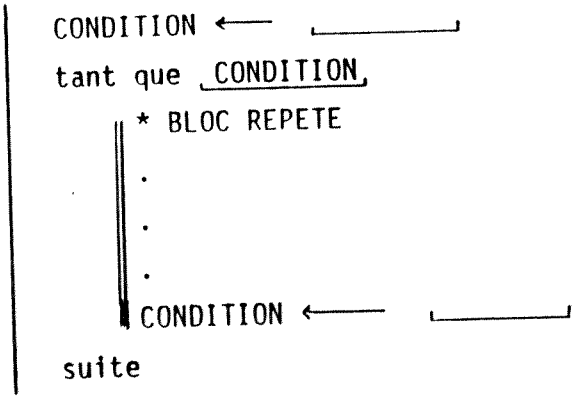
l'initialiser dans le bloc d'appel, le faire avancer dans le bloc appelé, (en principe, l'instruction d'avancement se place à la fin du bloc répété.

ex :

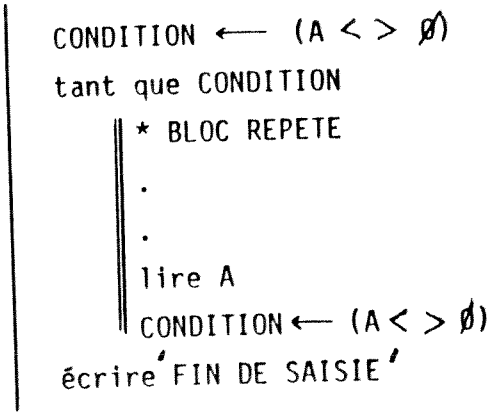


(voir les exemples du dossier 3)

si c'est une condition procéder de façon similaire avec le booléen correspondant :



ex :



La valeur du booléen CONDITION contrôle la sortie éventuelle du bloc répété après chaque passage dans ce bloc.

4.2.2 MEDE(E) Méthode déductive descendante

L'analyse modulaire est complétée par une procédure partant du résultat pour remonter aux données.

Exemple A : Calcul du prix SNCF sur conditions particulières de distance :

P(réel) : prix du billet PT(réel) : prix plein tarif D(entier) : distance	4 3 2 1	résultat = ECRIRE P P = SI D > 1000 ALORS PT*Ø.8Ø SINON PT PT = D*Ø.40 D = donnée

Exemple 2 : Reconnaître si une séquence M de caractères figure dans une chaine C donnée

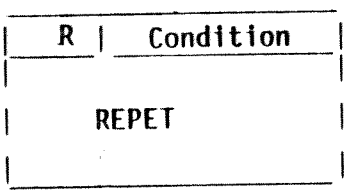
B(booléen) : vrai si M figure dans C	2 1	résultat = ECRIRE B B = EXECUTER RECHERCHE
module		***RECHERCHE**
INIT(module) : initiation de la recherche PARCOURS(module) : balayage	1 2	EXECUTER INIT TANT QUE NON FIN REPETER ‖ EXECUTER PARCOURS
		***PARCOURS**
C(chaîne) : M(chaîne) : I(entier) : indice de parcours de C	1 3 2	B = (SCH(C,I,LGR(M)) = M) I = ØI + 1 FIN = B OU (I > LGR(C) - LGR(M))
		***INIT**
	1 2 3 4	FIN = FAUX C = donnée M = donnée I = 1

Traduction BASIC

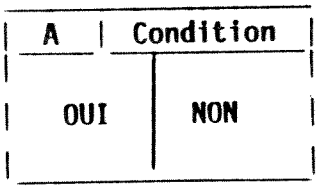
```
0010 REM PROGRAMME PRINCIPAL
0030 GOSUB 1000
0040 IF B THEN PRINT " M EST DANS C "
0050      ELSE PRINT " M N'EST PAS DANS C "
0060 END
1000 REM RECHERCHE
1010 GOSUB 2000
1020 WHILE NOT FIN : GOSUB 3000 : WEND
1030 RETURN
2000 REM INIT
2010 FIN=0
2020 INPUT "CHAINE C ";C$
2030 INPUT "CHAINE M ";M$
2040 I=1
2050 RETURN
3000 REM PARCOURS
3010 B=(MID$(C$,I,LEN(M$))=M$)
3030 FIN=B OR (I>LEN(C$)-LEN(M$))
3040 I=I+1
3050 RETURN
```

4.3. Représentation de la structure d'un algorithme

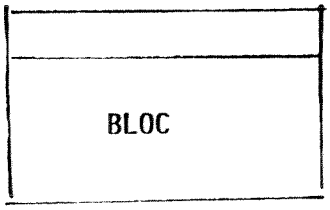
On représente un bloc par un rectangle contenant le nom du bloc. Un bandeau placé en tête du rectangle précise l'appartenance du bloc à une structure répétitive ou alternative.
Les blocs sont représentés sur un schéma arborescent en fonction du niveau d'imbrication.



Structure répétée

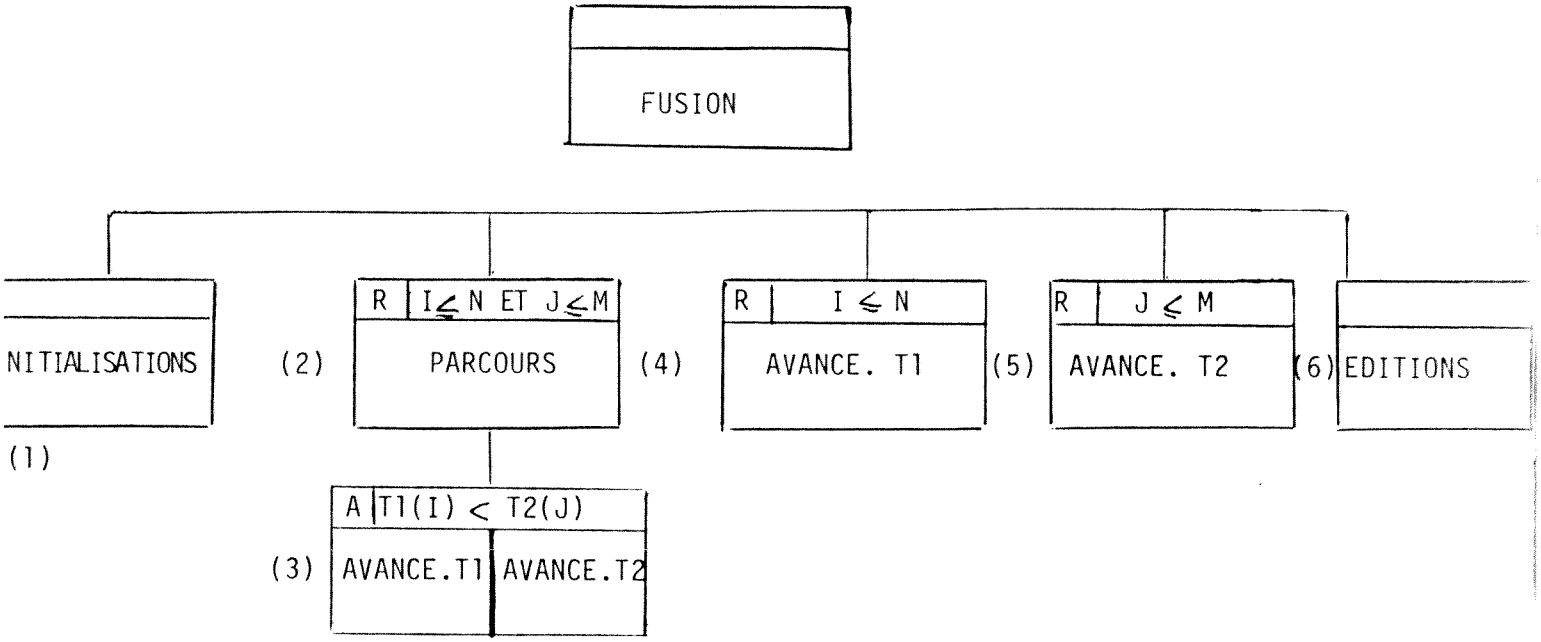


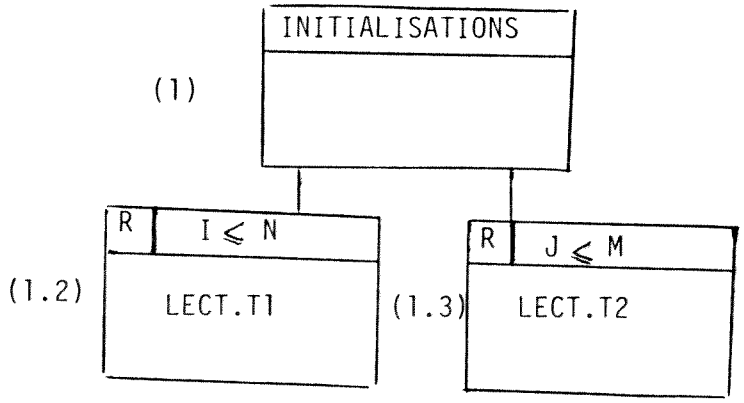
Structure alternée simple



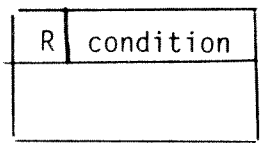
Module

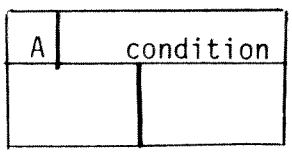
Exemple : **FUSION DE DEUX TABLEAUX**





Les structures consécutives se lisent de gauche à droite.
Les structures emboitées de haut en bas.
L'arbre est parcouru séquentiellement de haut en bas et de gauche à droite.
(1) à (6)

Les graphismes 

et 

regroupent

. les instructions d'initialisation des conditions
. le(s) bloc(s) de la structure proprement dite.

Ainsi (1.2) regroupe les instructions :

```
LIRE N
POUR I DE 1 A N
  ||  * LECT.T1
```

De même :

(2)

```
I ← 1
J ← 1
K ← 0
TANT QUE I ⩽ N ET J ⩽ M  REPETER
    ||* PARCOURS
    ||  SI T1(I) < T2(I)  ALORS EXECUTER AVANCE.T1
    ||                    |||  SINON EXECUTER AVANCE.T2
```

4.4 Techniques de vérification etde mise au point de programmes

Voici quelques techniques utilisables pour la phase (délicate ... et stressante) de la mise au point.

4.4.1 La relecture

Après avoir introduit, au clavier, un texte "source" dans le langage retenu, il est essentiel :

1° de sauver immédiatement ce texte sur un support externe (disquette ou cassette) afin de se prémunir des fausses manoeuvres ultérieures qui pourraient détruire des heures de travail

2° en cas de gros programme, il vaut même mieux sauver des morceaux en cours de route

3° de lister le programme à l'écran puis sur une imprimante pour le relire tranquillement et voir immédiatement les inévitables erreurs de frappe

4.4.2 L'élimination des fautes de syntaxe

Selon le cas, une exécution du programme ou l'enregistrement d'une ligne de programme provoquera habituellement un arrêt sur la première erreur de syntaxe rencontrée avec, selon les logiciels BASIC une indication codifiée (ou en clair) du genre d'erreur rencontrée.
Attention ! souvent une erreur en cache une autre.
La correction se fait, soit par déplacement du curseur, effacement et insertion si c'est un éditeur BASIC évolué (éditeur pleine page) pour passage en mode EDIT sinon on frappe alors

EDIT n° ligne à corriger

et un certain nombre de touches de fonction facilitent la correction (cf. annexe dossier Ø) sinon, n'oublions pas que, en BASIC, l'écriture d'une ligne numérotée remplace, le cas échéant, l'ancienne ligne de même numéro. On pourra donc simplement réécrire la ligne concernée.

Ne pas oublier de sauver la version corrigée, qui pour le moment ne réside que dans la mémoire centrale de l'ordinateur.
Les erreurs de ponctuation sont souvent les plus difficiles à détecter : (parenthèses ouvertes, non refermées; guillemets ouverts non refermés, double-points intempestifs, parfois un espacement est exigé entre les

mots-clés (cela est d'ailleurs plus clair !).
Avec un peu d'habitude de la programmation, on arrive à rendre le nombre d'erreurs de syntaxe négligeable.

4.4.3 L'élimination des erreurs d'exécution

Ce sont les plus difficiles à corriger, une fois le programme syntaxiquement correct. C'est ici que la structuration de l'algorithme est déterminante.

1. Le cas le plus difficile est le "bouclage" du programme sur une ou plusieurs instructions.
   Le curseur d'écran disparait, le clavier n'a plus accès à la mémoire, le temps passe.
   Il faut, en général interrompre de force par un BREAK ou l'extinction de l'appareil.
   Une analyse des conditions contrôlant les blocs répétitifs permet généralement d'apercevoir une condition ne se réalisant jamais (par exemple : N n'est jamais égal à zéro car il n'a pas été décrémenté après chaque passage sur le bloc).
   Pour localiser une erreur trop subtile, il existe plusieurs possibilités de recherche en BASIC :

   1) **interrompre** l'exécution d'un programme :
   (BREAK ou CTRLC)
   On peut alors faire afficher les valeurs des variables utilisées en mode machine de bureau.
   On peut ensuite poursuivre l'exécution à la ligne où on l'avait interrompue avec la commande CONT.

   2) **ajouter** des instructions dans le programme.
   Instructions demandant d'afficher les indices ou les variables de contrôles des structures répétitives à chaque répétition.
   Refaire une exécution du programme modifié.

   3) **N'y aurait-il pas de GOTO** incontrôlés ?

   4) Ajouter **des instructions STOP** dans des lignes critiques (fin de bloc ou aiguillage).
   La rencontre d'un STOP interrompt le programme.
   On peut alors demander en mode machine de bureau la valeur de variables.
   On peut continuer l'exécution par la commande de CONT

5) Demander **la trace** des exécutions (commande TRON annulée par TROFF)
Contrôler les numéros de lignes, et en particulier vérifier qu'il n'y a pas de boucle sans fin.

6) Pour vérifier qu'on est passé à un endroit du programme (par exemple dans la bonne partie d'une alternative) faire **afficher un commentaire** à cet endroit.

2. **Des résultats non conformes** aux prévisions ou des erreurs de division par zéro ou de dépassement de capacité.
Le programme s'interrompt, un message ou un résultat non prévu s'affiche. Le BASIC, très commode sur ce point permet de savoir où en sont les différentes variables manipulées.
Il suffit, après l'interruption et avant un nouveau RUN, de demander en **mode immédiat**, l'affichage des valeurs des différentes variables en cause, notamment des compteurs utilisés dans les répétitives ou des drapeaux figurant dans les conditions.
Le mode immédiat consiste à frapper au clavier une instuction BASIC, **sans numéro de ligne**, du type PRINT I, A, DRAP...
Cette instruction est exécutée **immédiatement** et la valeur actuelle des variables concernées est affichée à l'écran.

3. **Erreurs de transcription fréquentes et non détectées par le système**

- Pour les BASIC où IF ... THEN peut être suivi de plus d'une instruction : une instruction en fin de ligne, dont on voudrait qu'elle soit exécutée dans tous les cas, est sous le contrôle d'une condition (IF)

  Exemple : FOR I = 1 TO N : IF T(I) = 1 THEN COMPT⇐ COMPT + 1 : NEXT
  L'instruction NEXT n'est exécutée que tant que T(I) est égal à 1.
  Ensuite la répétition s'arrête, même si I n'est pas arrivé à N !

- Un sous-programme ne finit pas par RETURN.
  Erreur non détectée comme erreur de syntaxe si le sous programme en question est suivi d'un autre sous-programme, qui lui, finit par RETURN
- Il faut, enfin, vérifier la logique du programme, c'est-à-dire la construction même pour déceler des erreurs de conception, ou, ce qui arrive fréquemment, des programmes fonctionnant mal sur certains cas particuliers, fortuits ou mal analysés au départ.

Pour cela, il est bon de constituer un **jeu d'essai.**

4.5 Les jeux d'essais

Une erreur de logique peut longtemps rester inaperçue si elle ne se produit que pour une conjonction rare de données d'entrée ou d'anomalies de saisie.
Faire la preuve d'un programme n'est pas une opération facile.
La sûreté d'un pragramme est renforcée par :

- le fait de le structurer clairement lors de l'établissement de l'algorithme
- l'utilisation de jeux d'essais conçus dès l'analyse du problème, pour couvrir, si possible, l'ensemble des cas de figure présentables par les données fournies en entrée.
  Pour ce faire, il convient d'analyser toutes les **conditions** (tests) contrôlant des blocs d'instructions et de constituer les diverses combinaisons de données d'entrée permettant de passer par toutes les branches de l'arbrescence et des différentes alternatives présentes.

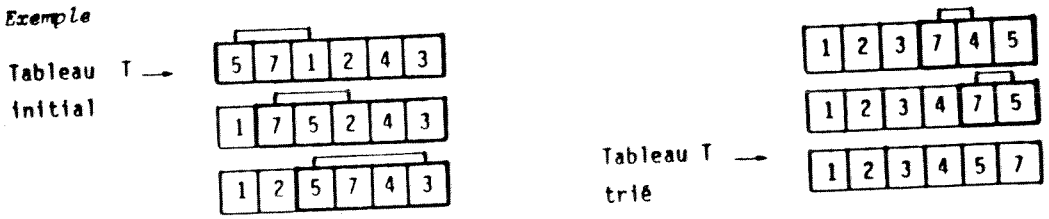
MINI T.P 4

1. Algorithmes permettant de trier (en ordre croissant par exemple) les éléments d'un tableau, en mémoire centrale.
Il existe plusieurs méthodes de tris, en voici quelques unes :

Tri par sélection

Classer les N éléments d'un tableau T par ordre croissant.
Le principe du tri par sélection est le suivant : Un parcours du tableau permet de rechercher le plus petit élément qui est alors permuté avec le premier élément. Le même traitement est effectué avec le tableau ayant le premier élément en moins. Cette recherche se fait jusqu'à ce que tous les éléments soient classés.

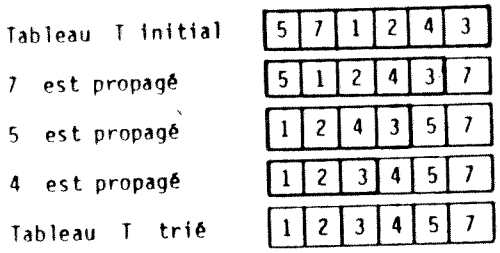
Exemple



. Tri par propagation ou méthode de la bulle

Classer les N éléments d'un tableau T par ordre croissant.

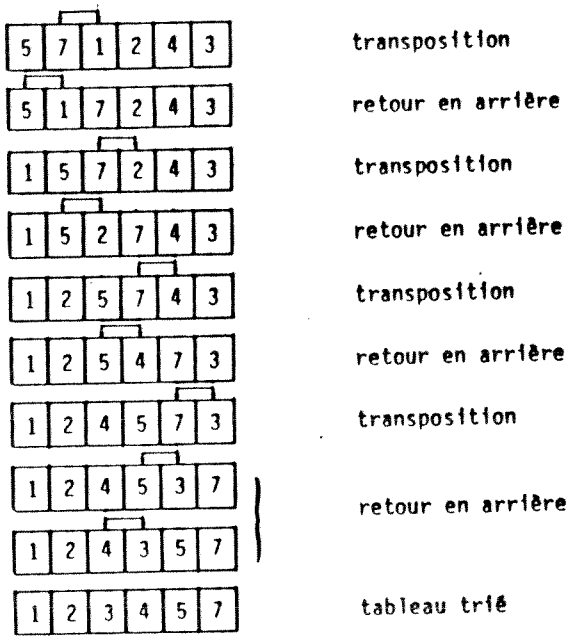
Le principe du tri par propagation est le suivant : Le tri est réalisé en propageant par permutations successives le plus grand élément du tableau à la fin de celui-ci (comme des bulles qui remontent à la surface d'un liquide). Ce principe est répété pour chacun des éléments.



. Tri par transposition

Classer les N éléments d'un tableau T par ordre croissant.

Le principe du tri par transposition est le suivant : Deux éléments consécutifs sont comparés et permutés, si nécessaire, puis un retour en arrière permet de vérifier si l'ordre déjà établi a été modifié ou non par cette permutation. Dans le premier cas l'ordre est rétabli.



Tri par comptage

Classer les N éléments d'un tableau T par ordre croissant.

Le principe du tri par comptage est le suivant :

1ère étape : création du tableau de compteurs.

Chaque élément est comparé à tous les autres afin de déterminer le nombre d'éléments qui lui sont inférieurs. Les résultats des N comptages sont rangés dans un tableau de compteurs possédant lui aussi N éléments :

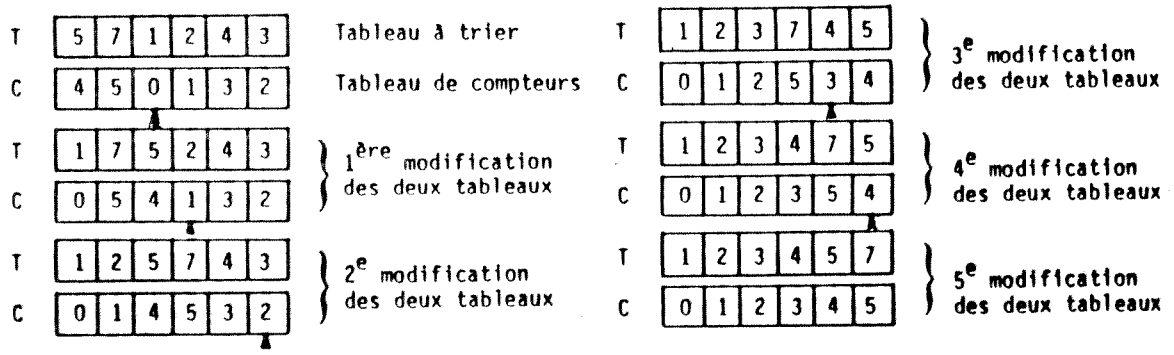
Tableau à trier	5	7	1	2	4	3
Tableau de compteurs	4	5	0	1	3	2

Le deuxième élément du tableau de compteurs indique qu'il y a 5 éléments inférieurs au deuxième élément (7) du tableau à trier.

2e étape : Utilisation du tableau de compteurs pour le tri.

Le premier élément du tableau T est permuté avec celui dont le compteur est nul (c'est bien le plus petit car il n'a aucun élément plus petit que lui-même). Les éléments correspondants dans le tableau de compteurs sont aussi permutés. Le processus est appliqué une deuxième fois avec l'élément dont le compteur est égal à 1 et ceci est répété jusqu'à la valeur N - 1 du compteur.

□ Exemple



2. Editer le triangle de PASCAL
   - en utilisant une matrice (n x p)
   - en utilisant uniquement un tableau à 1 dimension

3. Sur un échiquier, c.a.d. un tableau 8 x 8, à partir d'une case quelconque, marquer toutes les cases susceptibles d'être atteintes en un coup par une tour (une case atteinte contiendra la valeur 1, une case non atteinte la valeur Ø) - Même question pour un fou.

4. ontamination Sur une grille de 12x8 cases, placer au hasard 4 cases contaminées. Chaque jour , les cases adjacentes à une case contaminée sont elles-mêmes contaminées.
   Afficher l'évolution quotidienne de la contamination. On pourra commencer par mettre toutes les cases à zéro (jour Ø). Puis marquer avec "1" les cases contaminées le 1er jour, "2" celles du lendemain etc... Arrêter l'affichage lorsque toute la grille est marquée.

Variante :

Demander au clavier :

- le nombre initial de cases contaminées : n
- les coordonnées des n cases.