INITIATION AU LANGAGE BASIC

TOME 1

C. CHICOIX Professeur d'université à l'INSA de Rennes

J. DEWITTE Ingénieur à l'INSA de Rennes

M. OLLIVIER Ingénieur à l'INSA d Rennes

PREMIERE PARTIE

INTRODUCTION A LA MICRO-INFORMATIQUE

1 PRESENTATION DE L'INFORMATIQUE

1 - 1) Généralités

Dès qu'un certain travail doit être effectué de façon répétitive l'intérêt d'un système de traitement automatisé devient évident si l' on désire épargner à l'homme des taches fastieuses. Pour cela il faut en général disposer d'une machine adaptée au travail automatisé ; ce peut-être un automate spécialisé, tel que l'on rencontre dans les systèmes de régulation de chauffage ou également un ordinateur qui grâce a une programmation adaptée permet de réaliser la tache que l'homme lui a fixée.

Plus précisément dans le cas d'un ordinateur, il est nécessaire de fournir à la machine les valeurs de certains paramètres (Données), celle-ci effectue un travail de calcul conforme à la programmation pour aboutir à la fourniture de résultats.

En résumé l'homme n'interviendra que pour alimenter la machine en données et pour recueillir les résultats.

1 - 2) Classification des machines de traitement des informations

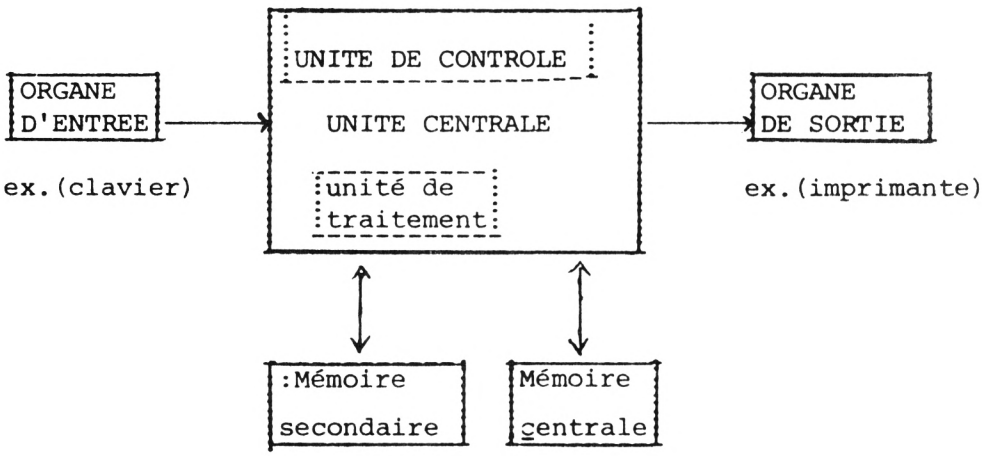
Il est fondamental de bien faire la distinction entre le traitement automatique lui-même, et les moyens mis en oeuvre pour le faire. Ainsi par exemple il est tout à fait concevable d'effectuer un traitement automatique par des moyens mécaniques, optiques, fluidiques aussi Bien que par des moyens électroniques.

Lorsque la machine ne peut faire qu'un seul type de traitement (ex. contrôle d'un ascenseur) on dira que la machine comporte un programme figé , par contre dès qu'il est possible de fournir au moyen d'un support externe le programme à la machine on dira que celle-ci comporte un programme enregistré, et c'est bien entendu le cas des ordinateurs.

Les ordinateurs, généralement considérés comme d'immense assemblage de composants, font appel à un concept extrèmement simple : celui de la machine de T Ü R I N G. Cette machine était composée d'un ruban perforé contenant le programme et une tête de lecture qui pouvait faire avancer ou reculer le ruban contenant les ordres à effectuer.

2 STRUCTURE GENERALE D'UN ORDINATEUR

Sans détailler le fonctionnement d'un ordinateur, il est important de connaitre l'outil que l'on va utiliser ; ce qui implique la présentation des différents organes constituant la machine de traitement de l'infor - tion.



Tout ordinateur est composé d'une unité centrale permettant d'effectuer par son unité de traitement les calculs prévus par le programme, ces calculs sont organisés logiquement dans un ordre imposé par ce même programme grace à l'unité de contrôle.

Le programme pouvant être stocké dans une mémoire secondaire, (ex. bande magnétique) avant d'être chargé en mémoire centrale ou l'on trouve également des données fournies par l'organe d'entrée. La visualisation des résultats s'effectuant sur l'organe de sortie.

3 ARCHITECTURE ET FONCTIONNEMENT DES MICRO-ORDINATEURS

3 - 1) Historique

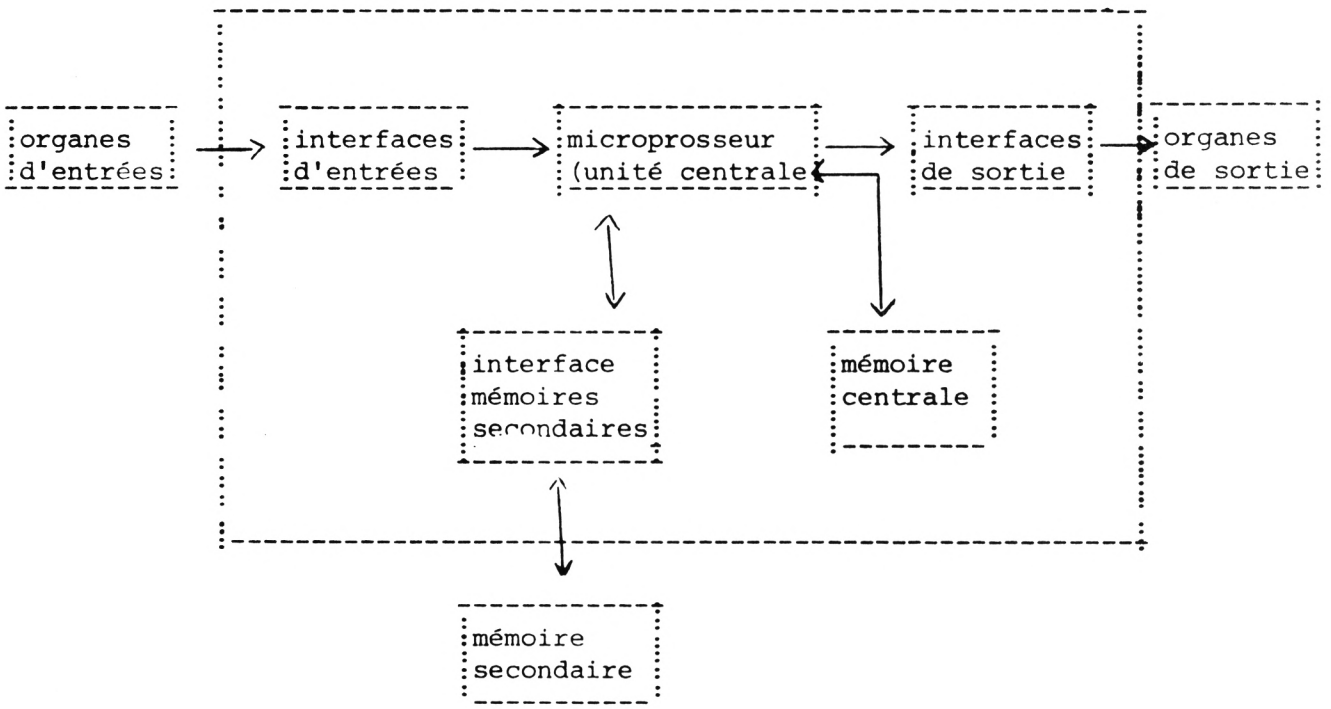
Tout à commencé comme dans un western : il était une fois dans l'Ouest... en 1971, une société américaine spécialisée dans la fabrication de terminaux informatiques à posé la question suivante à 2 grands fabricants de circuits intégrés : est-il possible de concevoir dans un même circuit intégré une petite unité centrale ? L'un de ces deux fabricant (INTEL) réussit à faire fonctionner un tel système en respectant tous les points du cahier des charges, sauf un : la vitesse de calcul était 10 fois trop lente ce qui évidemment ne pouvait pas satisfaire le demandeur.

La société INTEL fait alors une étude de marché pour voir si le produit qu'elle a conçu présentait un intérêt pour d'autres applications, et les résultats obtenus dépasseront et de loin toutes les espérances initiales.

Le microprocesseur était né en Californie.

3 2) Architecture d'un micro-ordinateur

Tout comme les ordinateurs classiques un micro-ordinateur comporte une architecture incluant les même organes principaux ; toutefois les progrès technologiques permettent d'effectuer un découpage différent de celui-ci



Ilest communément fait le découpage indiqué ci-dessus dans lequel on distingue :

- L'unité centrale ou microprocesseur
- Les circuits d'interfaces (entrées / sorties)
- Les circuits mémoire

Ces trois blocs fontionnels constituant le coeur du micro-ordinateur.

De plus on trouve également :

- Les organes d'entrée
- Les organes de sortie
- La mémoire secondaire

Analysons plus en détail ces différents éléments

3 3) Le microprocesseur ou unité de traitement

Actuellement les unités de traitement appelées également C P U (Central Processor Unit) sont en général contenu dans un même circuit intégré. Un C P U constitue la partie active de la machine puisque c'est lui qui traite les données et qui fourni les résultats. On trouve toujours - une unité de traitement arithmétique (addition, soustraction ...) et logique (comparaison et opérations logiques).

- Des mémoires spécialisées dans lesquelles sont effectués les traitements ; ces mémoires sont appelées REGISTRES.

L'unité de traitement comporte également un système de reconnaissance des instructions qui lui permet à chaque étape élémentaire de décoder, puis de faire exécuter l'opération demandée par le programme.

Les échanges entre le C P U et les circuits retirés à sa périphérie immédiate se font par des lignes organisées en B U S ; c'est à dire des lignes de données véhiculant toutes les informations logiques de données, d'adresse ou de contrôle;

3 4) Circuits mémoire principale

Bien qu'il soit possible de charger dans la mémoire centrale le programme et ses données il est courant de distinguer le mode de stockage de ces deux éléments.

Un programme, lorsqu'il est au point, n'a plus besoin d'être modifié ; il est donc possible de le conserver de manière permanente sur un support mémoire figé. Ceci garde donc l'information même en cas de coupure d'alimentation. Par contre les données sont des grandeurs essentiellement variables qui ne peuvent donc être stockées que dans des mémoires modifiables tout au long de l'exécution du programme.

Dan le premier cas (programme) il est courant d'utiliser des mémoires mortes (ROM pour Read Only Memoires) programmées en Usine par le fabricant de circuit intégré, en PROM (Programmable Read Only Memories) programmées par l'utilisateur sur un système spécial.

Dans le second cas (données) on utilise des mémoires (**RAM** on Random Acces Memoires) qui peuvent être lues ou écrites dans un ordre aléatoire au gré des désirs de l'utilisateur mais qui présentent l'inconvénient de perdre leur contenu lors des coupures d'alimentation.

On dit également que ces mémoires sont VOLATILES.

3 - 5) Circuits d'Interface

De manière à coupler le plus simplement possible les organes d'entrée ou de sortie on utilisera des circuits intégrés spéciaux appelés circuits d'interface. Ils permettent de relier au microprocesseur, en le déchargeant au maximum des taches d'interfaçage, tous les dispositifs externes courants. En général ils sontprogrammable, c'est à dire qu'ils peuvent être configurés par le programme dans un mode de fonctionnement adapté à l'utilisation particulière demandée.

3 - 6) Technologies

Bien qu'il existe de multiples technologies et plus d'une centaine de boitiers C P U différents, il apparaît que la grande majorité des C P U sont constituée autour d'une technologie M O S, ce qui veut dire que les circuits logiques sont constitués par des transistors (donc des semi-conducteurs) dont la commande se fait au moyen d'une électrode métallique (MOS = Métal - oxyde - semiconductor).
Ce type de transistors permet d'effectuer un intégration poussée (plus de 10 000 transistors dans un même circuit) tout en gardant une rapidité d'exécution convenable (jusqu'à 10 mégahertz pour le temps de cycle horloge).

On trouve dans cette technologie des microprocesseurs dont les mots de données sont de 4 ,8 et 16 bits mais les plus courants ont des mots de 8 bits.

4 NOTIONS DE NUMERATION BINAIRE

4 - 1) Généralités

La plus petite information que l'on puisse mémoriser est appelée B I T , elle représente un chiffre binaire 0 ou 1. On associe plusieurs bits pour former un mot binaire auquel peut-être donné une signification de nombre.

Ainsi par exemple en numération décimale le nombre 1392 est composé de 4 chiffres décimaux dont la signification est :

1	fois	1 000	(10^3)
3	fois	100	(10^2)
9	fois	10	(10^1)
2	fois	1	(10^0)

Donc $1392 = 1 \times 10^3 + 3 \times 10^2 + 9 \times 10 + 2 \times 10^0$

En binaire (base 2 au lieu de base 10)

Le nombre 10110 aura donc la signification suivante :

1	fois	2^4	(16)
0	fois	2^3	(8)
1	fois	2^2	(4)
1	fois	2^1	(2)
0	fois	2^0	(1)

Donc $10110 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$= 16 + 0 + 4 + 2 + 0$

$= 22$ en décimal

Remarque : on peut faire correspondre à chaque bit (de droite à gauche) d'un nombre binaire les "poids" décimaux 1, 2, 4, 8, 16, 32 ..... pour effectuer la conversion en décimal

Ainsi par exemple

1 1 1 1 1 1 1 1	⟷	255
0 1 1 1 1 1 1 1	⟷	127
0 0 0 0 1 1 1 1	⟷	15
BINAIRE		DECIMAL

4 - 2) Notion d'arithmétique binaire

Un nombre binaire composé de 4 bits représente un nombre décimal compris entre 0 et $2^4 - 1$ mais également un algébrique compris entre -2^{4-1} et $(2^{4-1} - 1)$ en considérant que le bit le plus à gauche est le bit de signe (0 si positif, 1 si négatif) et que le mode de représentation est en binaire en complément à 2.

Ex. pour un mot de 4 bits égal à 1101 la règle suivante peut-être employée pour connaitre sa valeur décimale :

Le bit situé le plus à gauche vaut 1 — c'est donc un nombre négatif.

Prenons le complément bit à bit de ce mot (les 1 deviennent des 0 et réciproquement) ce qui donne 0010 puis ajoutons 1

0010 + 1 = 0011

tout calcul fait c'est donc le nombre - 3

Remarque pourquoi introduire une représentation binaire en complément à 2 ? tout simplement parce que cette représentation permet de traiter sur ordinateur, de façon unique les nombres positifs et les nombres négatifs lors des opérations arithmétiques.

4 - 3) Addition binaire

Il suffit de donner la règle d'addition de 2 bits et les conditions sur les retenues pour comprendre la technique de l'addition binaire.

0 + 0 = 0 , retenue 0

0 + 1 = 1 , retenue 0

1 + 1 = 0 , retenue 1

Tout comme en décimal 9 + 1 = 0 , retenue 1

Exemple :

```
  0 0 1 1 0 1                    1 3
+ 0 0 1 1 1 1                  + 1 5
-------------                  -----
        0  retenue 1             2 8
      1
  0 0 1 1 0 1
  0 0 1 1 1 1
-------------
      0 0
```

RETENUE 1

etc

```
  0 0 1 1 0 1
  0 0 1 1 1 1
-------------
  0 1 1 1 1 1   ------>  28
```

Exercices : effectuer les additions binaires suivantes

0 1 0 1 0 0 + 0 1 1 0 = ...

0 1 0 0 1 1 + 0 1 1 1 = ...

Donner également la valeur décimale de ces 2 opérations.

4 - 4) Soustraction binaire

Une soustraction binaire est équivalente à une addition binaire entre le premier terme et le complément à 2 second terme

Ex. 3 - 2 = 3 + (- 2) en décimal

011 - 010 = 011 + (110)

```
  0 1 1
  1 1 0
-------
1 0 0 1      ------>  résultat
  ↑
```

retenue ignorée — signe

pour des mots de 3 bits (débordement)

Exercice pour les 2 mots de 4 bits suivants (3 bits + signe)

trouver le résultat de la soustraction binaire

```
  0 1 0 1          0 1 0 1          0 1 0 1
- 0 1 0 0        - 0 1 1 0        - 1 0 1 1
---------        ---------        -----------
.  . . .          .  . . .          .  . . .
```

Exercice trouver le nombre binaire représentant

+ 4, - 7 , 32, - 32, - 1, + 1, 0

4 - 5) Notion de base hexadécimale

Il est évident que le binaire comporte une suite très importante de bits, dès qu'un grand nombre doit être représenté (par exemple 65530 ⇔ 1111 1111 1111 1010) ; il est plus facile de définir un autre mode de représentation en prenant les bits par groupe de 4.

BINAIRE	DECIMAL	HEXADECIMAL
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	10	A
1 0 1 1	11	B
1 1 0 0	12	C
1 1 0 1	13	D
1 1 1 0	14	E
1 1 1 1	15	F

Le code hexadécimal utilise alors les chiffres 0 à 9 puis les lettres A à F pour représenter les nombres compris entre 0 et 15.

Exemples : 1 1 1 1 1 0 0 0 1 1 0 0 $\Longleftrightarrow$ F 8 C

1 0 0 1 1 1 0 1 0 1 1 1 $\Longleftrightarrow$ 9 D 7

Exercices : donner la valeur binaire des nombres hexadécimaux suivants :

7 F, F E, 80, C D, A C.

5 GENERALITE SUR LES LOGICIELS

5 - 1) Introduction aux logiciels

Un programme contrôlant un système informatique doit pouvoir fonctionner correctement quelquesoient les données que l'on peut lui fournir, il est donc indispensable qu'il soit écrit avec le plus grand soin.

Il est possible de disposer de langage de programmation plus ou moins évolués, c'est à dire que la traduction d'une opération quelconque est faite plus ou moins directement. Le premier niveau correspond au langage ASSEMBLEUR dans lequel toutes les opérations élémentaires sont explicitement décrites (il est indispensable de bien connaitre la struture de la machine) et de plus tout programme ainsi décrit ne pourra fonctionner que sur un type donné d'ordinateur.

Il est plus facile d'envisager l'utilisation de langages plus évolués dont les règles d'écriture seront pratiquement indépendantes de la machine (c'est le cas de BASIC) et pour lesquels des opérations complexes peuvent être exprimées très simplement.

Dans tous les langages évoluées, il est bien entendu nécessaire de disposer d'un organe de traduction entre le texte rentré par le programme et le programme réellement exécusable par la machine. Ces programmes de traductions appartiennent toujours à l'une des deux familles suivantes :

- Les langages interprètés

- Les langages compilés

En quoi consiste un langage interprèté : très schématiquement chaque ligne de programme introduit par l'utilisateur est traduite par la machine, puis exécutée sans préjuger des autres lignes de programme. Il est donc évident qu'une telle approche présente à la fois une grande simplesse de mise au point, et un temps d'exécution relativement long. (BASIC est en général un langage interprété).

Par contre un langage compilé comporte :

- Un programme source constitué par l'ensemble des lignes de l'utilisateur considérées comme un tout.

- Un programme objet qui est la traduction du programme source ; ce dernier programme est le seul exécutable par la machine mais le programmeur ignore en général sa structure fine, ce qui oblige dès qu'une erreur est décélée de revenir au programme source, de le modifier puis de refaire une compilation.

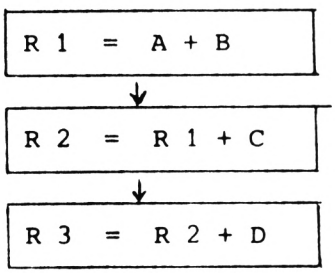
5 - 2) Notion d'algorithmes

L'art de traduire un problème posé en méthode programmable consiste à définir l'algorithme ou les algorithmes à programmes.

Prenons par exemple le cas de 4 nombres à additionner ; traduire en algorithme consiste à décrire finement la suite logique des opérations.

Soient A, B, C et D les nombres, il est possible de faire la démarche simple suivante :

- Additionner A et B
- Additionner le résultat avec C
- Additionner le résultat avec D

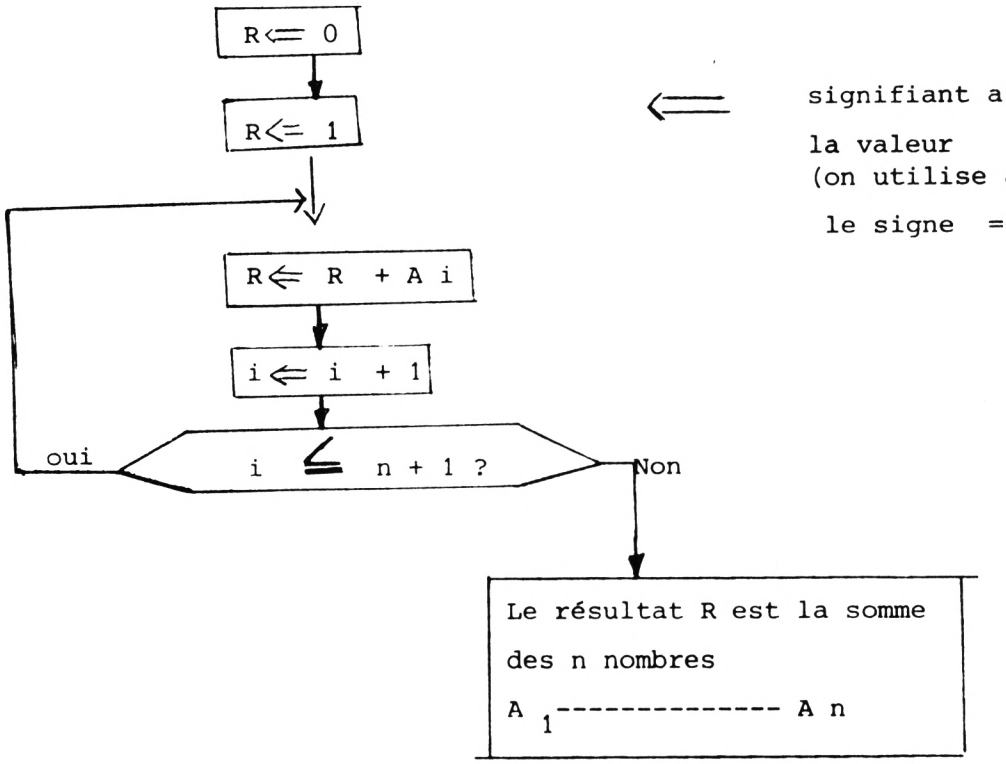


Que faire si on a à additionner n nombres ? Il est évident que la démarche précédente devient très lourde si n est grand ; il convient alors de faire des itérations.

Soient A_1 , A_2 , --------, A_n les n nombres à additionner.

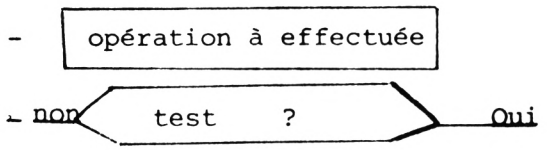
Posons R = 0 le résultat avant addition

Pour i venant de 1 à n il convient de faire R = A i + le résultat intermédiaire précédent.



⇐ signifiant affecter la valeur (on utilise aussi le signe =)

Remarque L'algorithme précédent a été établi sous forme d'un organigramme ou il apparait 2 types d'informations.



Avec ces deux éléments il est possible de définir tout algorithme sous leur forme programmable ; il faut toutefois prendre de grande précautions pour vérifier que tous les cas (même particuliers) ont été modélisés.

DEUXIEME PARTIE

ELEMENTS DE LANGAGE DE COMMUNICATION HOMME-MACHINE

LE LANGAGE BASIC

Les langages de haut niveau peuvent être classés en deux groupes : langages compilés et langages interprétés. Si ces derniers conduisent à des programmes plus lents à exécuter, par contre ils sont d'un apprentissage plus rapide. C'est le cas du langage BASIC, le plus facilement abordable par les non informaticiens.

BASIC (Beginners All purpose Symbolic Instruction Code) est un langage de haut niveau, c'est à dire théoriquement indépendant de la machine, qui a été défini dès 1960 au Dartmouth College (Angleterre). Langage interactif autrefois implanté principalement sur des systèmes en temps partagé, il est maintenant disponible sur pratiquement tous les microordinateurs. En effet l'utilisateur final d'un microordinateur préfèrera en général la simplicité de mise en oeuvre, critère directement relié au coût de développement d'une application, plutôt que la vitesse, qui n'est pas un facteur déterminant dans beaucoup de cas.

Nous avons vu que toute communication s'effectue à l'aide d'un support et que les relations entre l'utilisateur et un ordinateur nécessitent de même l'usage d'un langage. BASIC, comme tout autre langage est composé de phrases,les phrases de mots et les mots de caractères qui constituent l'alphabet du langage. L'usage d'un langage nécessite l'apprentissage d'un vocabulaire et de sa sémantique (signification associée aux mots du vocabulaire), de l'orthographe des mots et des règles d'écriture des phrases (syntaxe ou grammaire du langage). En fin, pour faire exécuter une opération, il sera nécessaire d'écrire un "texte" décrivant l'opération.

La tâche du programmeur sera donc d'exprimer l'opération à exécuter (calcul scientifique, gestion d'un stock, tri d'un fichier, ...) dans un langage compréhensible par la machine, cette description du traitement constituant un programme. Cette traduction s'effectue en deux temps.
Le premier consiste au passage du problème à l'énoncé d'une solution sous forme d'un algorithme.
Le deuxième temps consiste en la transcription de cet algorithme en un langage compréhensible par la machine utilisée. Le passage Problème → algorithme est relativement indépendant du langage utilisé, alors que le deuxième temps dépend le plus souvent non seulement du langage mais de la machine utilisée.

I ALGORITHMIE

I-1 Notion d'algorithme

Reprenant la définition de MARKOV, on appelle algorithme "un ensemble de règles précises qui définit un procédé de calcul destiné à obtenir un résultat déterminé à partir de certaines données initiales." Un algorithme est donc une méthode de traitement qui, pour une classe de problèmes donnée, permet d'obtenir la solution recherchée.

Pour qu'un algorithme soit valide, il faut qu'il puisse être réalisé en un nombre fini d'opérations et que chaque opération soit exécutable en un temps lui aussi fini. Il faut également que les traitements mis en oeuvre soient reproductibles, et donc le résultat invariant pour une même entrée.

Etant donné un problème concret, une certaine démarche doit être respectée pour aboutir à un algorithme. Tout d'abord le problème doit être exprimé de manière précise et en termes "informatiques", c'est à dire qu'au niveau de ses données (sources d'information), de ses résultats (solution recherchée) et des traitements à réaliser, les contraintes et les restrictions liées au matériel utilisé devront être prises en compte, au moins dans leurs grandes lignes. Ensuite le problème doit être, si possible, décomposé en un ensemble de sous problèmes. Il s'agit d'un découpage qui vise à l'obtention de modules dont la fonction doit être bien précisée ainsi que les liaisons entre les différents modules. On peut ainsi de manière iterative affiner l'analyse. Chaque élément de base pourra alors être exprimé sous forme d'un sous algorithme, l'ensemble des sous algorithmes constituant l'énoncé algorithmique du problème initial.

Un problème sera dit décidable s'il existe au moins un algorithme permettant de le résoudre. Il faut noter les points suivants :

- Le fait qu'il existe ou non une solution au problème est différent de la décidabilité du problème. Il peut exister une solution alors qu'il n'existe pas d'algorithme pour l'obtenir.

- Plusieurs algorithmes différents peuvent exister pour résoudre un même problème. Ce sont alors d'autres critères qui permettront de

choisir entre les différents algorithmes : temps de traitement, encombrement du programme résultant, complexité de la mise en oeuvre, ...

Si le respect d'une certaine démarche est souhaitable, il n'existe pas de méthode générale pour l'obtention d'un algorithme. aussi face à un problème particulier, une première étape peut consister à rechercher si un (ou plusieurs) algorithme à déjà été proposé. Dans le cas contraire, il s'agit de découvrir une solution au problème, en admettant que ce dernier soit décidable.

I 2 Réprésentation d'un algorithme

Il s'agit d'une transcription et d'une explicitation de l'algorithme qui peut se faire sous plusieurs formes : langage algorithmique ou organigramme.

Ces deux formes visent au même résultat. Cependant le passage au programme proprement dit sera d'autant plus facile que l'algorithme aura été exprimé selon une structure plus proche de celle du langage utilisé. Ici, nous nous limiterons à parler des organigrammes, forme d'expression des algorithmes bien adaptée au BASIC.

Un organigramme est un schéma logique qui analyse point par point les différentes phases du problème à traiter, ainsi que l'enchaine - ment à respecter dans l'exécution de ces phases.

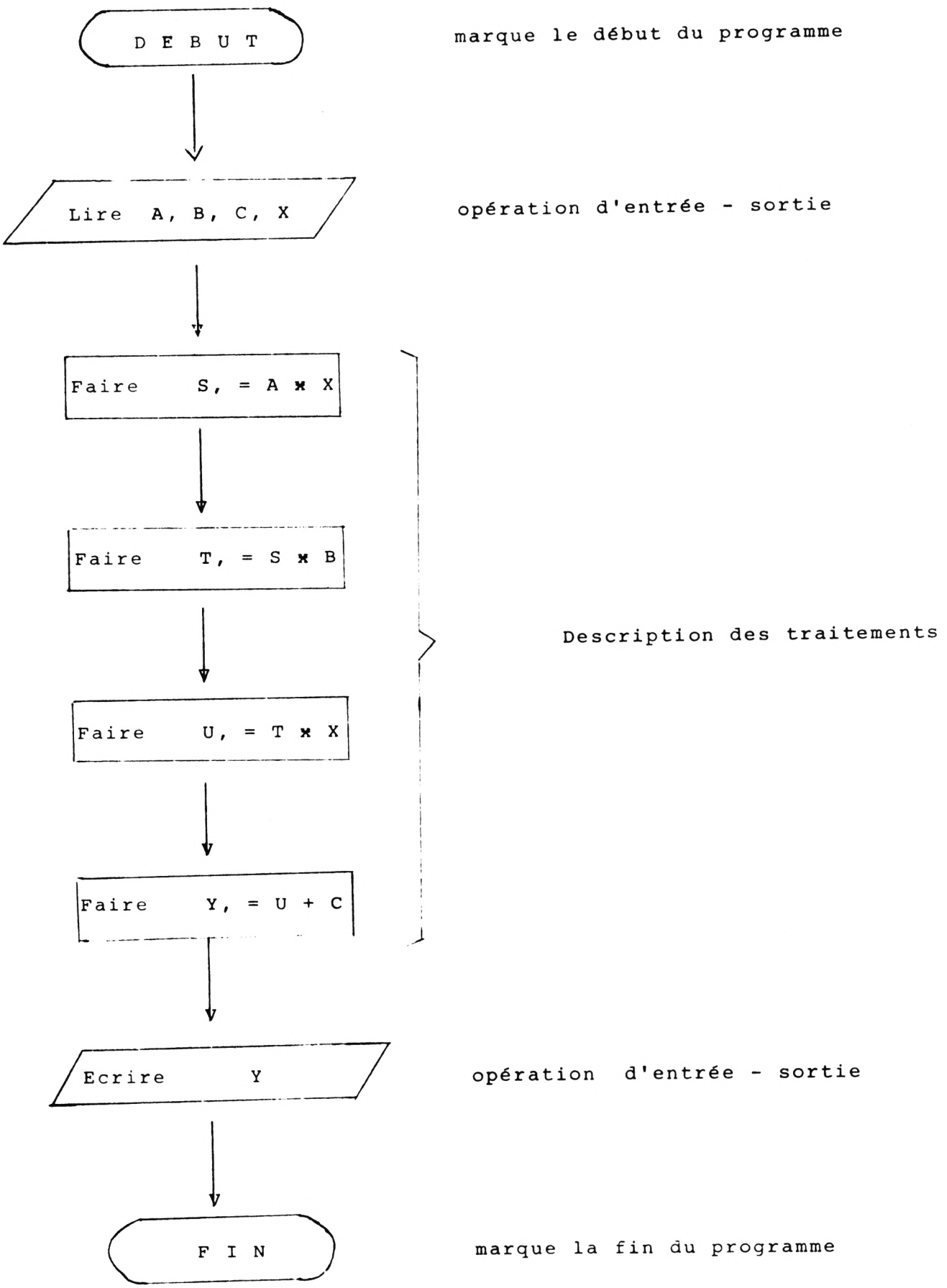
Prenons l'exemple du calcul de

$$Y = A X^2 + B X + C \qquad (I)$$

pour des valeurs données de A, B, C, X. L'algorithme utilisé ne correspond pas au calcul direct de (I), mais à

$$Y = (A X + B) X + C \qquad (2)$$

car utilisant une multiplication de moins que le calcul direct. Notons, au passage que l'algorithme le plus performant n'est pas la solution la plus immédiate. L'organigramme est le suivant :


D E B U T
marque le début du programme
Lire A, B, C, X
opération d'entrée - sortie
Faire S, = A * X
Faire T, = S * B
Faire U, = T * X
Faire Y, = U + C
Description des traitements
Ecrire Y
opération d'entrée - sortie
F I N
marque la fin du programme

Les macarons de début et de fin, peuvent préciser le nom de l'algorithme. Utilisés dans le corps de l'algorithme, ils permettent de rattacher une étiquette à un point de l'algorithme.

Pour les opérations d'entrée- sortie, on peut préciser où (à l'aide de quel dispositif d'entrée ou d'édition) et sous qu'elle forme (présentation de l'entrée ou de la sortie) se présentent les informations à échanger.

Les traitements correspondent à des traitements en unité centrale. dans cet exemple, le signe " = " correspond à une opération d'affection et non à une égalité au sens mathématique du terme. Le traitement à effectuer est alors :

- Estimation du résultat de l'expression à droite du signe égal, ce qui suppose qu'à ce point du traitement une valeur ait été affecté à tous les symboles intervenant dans l'expression.

- Faire correspondre au symbole à gauche du signe égal le résultat de l'expression.

Ainsi l'opération suivante est licite

```
Faire    I = I + 1
```

et signifie : ajouter 1 à la valeur de I et donner à I cette nouvelle valeur.

Ce premier exemple correspondait à un programme linéaire, la totalité du programme étant décrite une fois et ce dans un ordre figé.

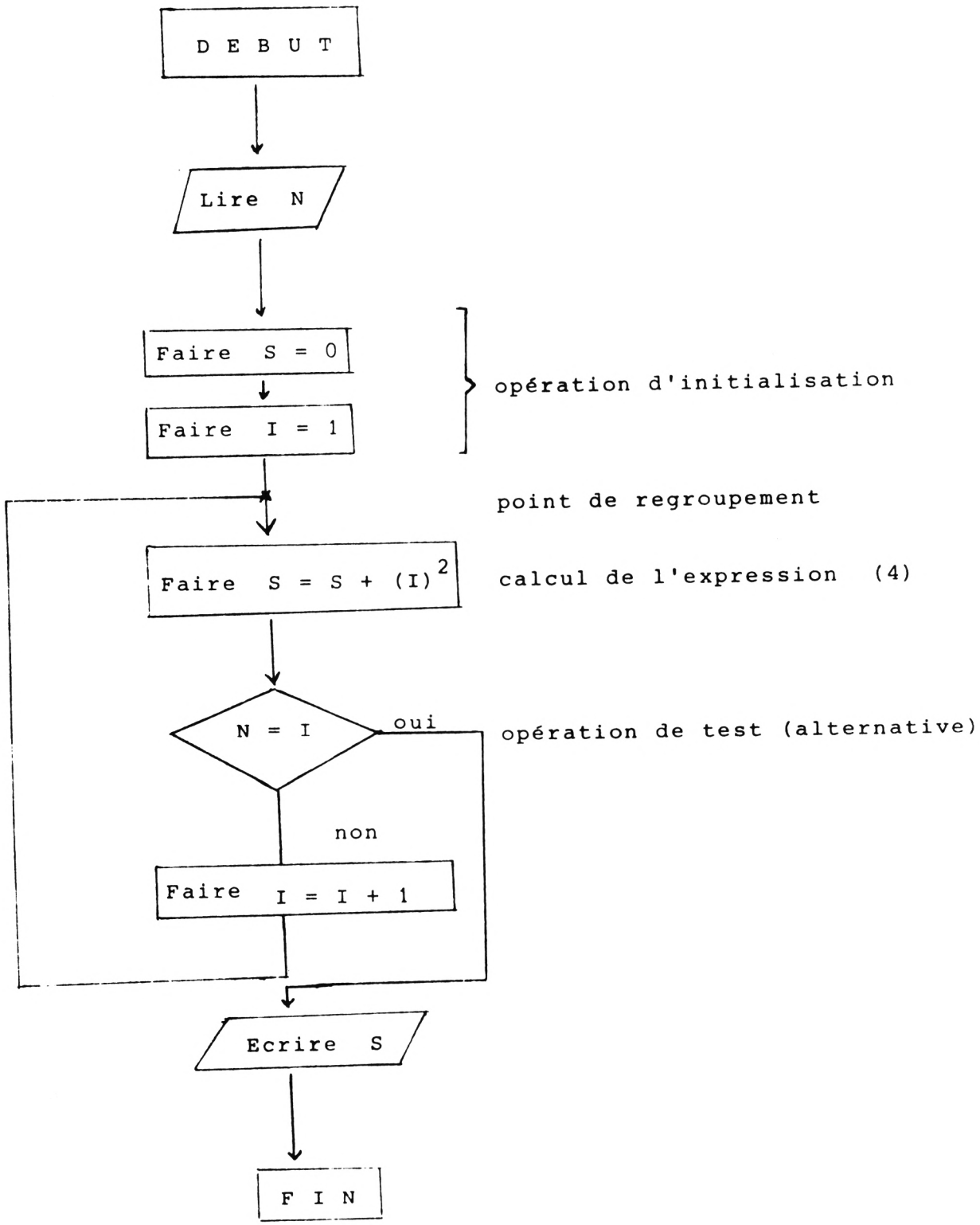
Considérons un deuxième problème, qui consiste à calculer la somme S_N des carrés des N premiers nombres entiers :

$$S_N = 1^2 + 2^2 + 3^2 + \ldots\ldots\ldots + (N-)^2 + (N)^2 \qquad (3)$$

Comme dans le premier exemple, l'algorithme utilisé ne correspond pas à la formulation du problème. On peut constater, en conservant les notations précédentes, que :

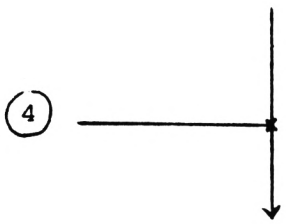
$$S_N = S_{N-1} + (N)^2 \qquad (4)$$

L'algorithme peut alors se représenter :

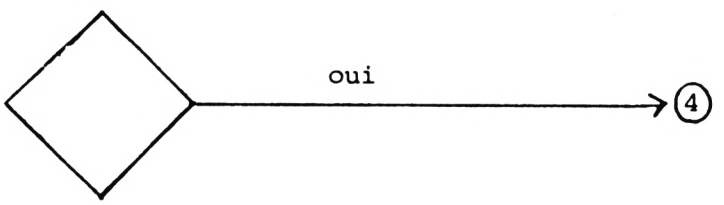


Dans l'opération de test, le losange ne contient pas une opération à effectuer, mais la description de la condition que l'on teste. Ainsi ici, on ne veut pas affecter la valeur de I à N, mais vérifier si I et N ont la même valeur. Selon le résultat de ce test, on aura branchement sur une partie ou l'autre de l'organigramme.

En cas d'organigramme important, il peut être nécessaire d'utiliser des points d'entrée



auquels correspondront des points de branchement



D'autres symboles sont utilisés, comme par exemple pour les aiguillages multiples ou bien les sous programmes, notions qui seront introduites dans l'étude du langage BASIC.

II LES ELEMENTS DU LANGAGE BASIC

II 1 Format d'une instruction

Une instruction BASIC est un texte constituant une phrase du du langage. Ce texte peut comprendre des mots réservés du langage, des opérateurs, des chiffres...

Il doit obéir à des règles de composition et de syntaxe.

La première étant qu'une instruction commence par un numéro d'identification. Elle se termine par le caractère "retour chariot" (return en anglais)

Exemple : 10 LET A = 25+3 (RC)

II 2 L'alphabet du langage BASIC

L'alphabet du langage BASIC est l'ensemble des caractères utilisable dans la composition des différentes instructions.

Cet alphabet comprend :

- Les caractères alphanumériques : de A à Z et de 0 à 9
- Le caractère blanc souvent représenté par ␣
- Les symboles spéciaux qui sont:
  - Les opérateurs arithmétiques + - / * ↑
  - Les opérateurs relationnels > < =
  - Les séparateurs % ( ) , $ "
  - Les signes de ponctuation . , :
  - plus le caractère "retour chariot (RC)" qui n'est pas visualisable mais qui délimite la fin de ligne.

II 3 Les mots du langage

Le langage BASIC comprend un certain nombre de mots qui sont des ordres pour l'interpréteur. Nous les appellerons MOTS CLES du langage.

Il est souvent utile d'avoir recours dans bien des domaines à des symboles pouvant représenter des variables susceptibles de prendre plusieurs valeurs. De la même manière en programmation nous ferons

appel à des IDENTIFICATEURS de variables dont la valeur pourra changer au cours de l'éxécution d'un programme.

D'autre part, avant d'utiliser dans un calcul une variable, il aura fallu lui affecter une valeur. Cette valeur initiale peut être figée. Une valeur écrite directement dans un programme s'appellera une CONSTANTE.

II 4 Les règles de constructions des mots du langage

a) Les identificateurs de variables

Les variables BASIC peuvent être de trois type distincts :

- les entiers qui sont des nombres généralement compris entre - 32 768 et + 32767.
- les réels pouvant habituellement varier de 10^{-38} à 10^{38} avec 9 chiffres significatifs
- les chaines de caractères.

A ces trois types, on peut parfois en ajouter un quatrième qui est le type réel double précision.

De manière standard, un identificateur en BASIC commencera par une lettre. Celle-ci pourra être suivie d'un chiffre. (Exemple : A 1, B, C O sont des identificateurs valides). Il à noter que de plus en plus, des versions de BASIC autorisent des identificateurs plus long donc plus explicites (exemples : PRIXDUPAIN, BENEFICE).

Pour définir une variable entière, on ajoutera à un identificateur le caractère %. Ainsi dans tout programme l'identificateur I % ne pourra contenir que desvaleurs entières positives ou négatives.

De même un identificateur de chaine de caractère se terminera par caractère $. Exemple : A 1 $

Cet identificateur désignera une chaine de 0 à 255 caractères.

Dans les versions de BASIC autorisant les réels en double précision, la distinction entre les réels se fera de la manière suivante :

- Les identifcateurs de réels simple précision se termineront par le caractère !
- Les identificateurs de réels double précision se termineront par le caractère #

b) Les constantes

Elles sont de deux types :

- les constantes numériques
- les constantes chaine de caractères

Les constantes numériques sont des nombres (réels ou entiers) positifs ou négatifs.

Si une constante est négative, elle sera précédée du signe - . Si elle est positive, le signe + pourra être omis.Ce signe sera suivi de la partie entière du nombre. Dans le cas d'un réel, nous trouverons de plus un point, puis si elle existe, la partie décimale.

Dans le cas des nombres réels, il est possible d'utiliser une notation exponentielle qui permet de simplifier l'écriture de nombres très grands ou très petits. Le nombre $3,15 \times 10^{-20}$ s'écrira 3. 15 E - 20

Les constantes chaines de caractères doivent être encadrées par des guillemets et avoir une longueur inférieur à 256 caractères.

Exemple : " IL FAIT BEAU "
est une constante chaine de caractères valide.

c) Les mots clés

Les mots clés de BASIC ne peuvent pas être utilisés comme identificateur de variables. Ils sont généralement en anglais. Nous les étudierons dans le détail au chapitre suivant.

d) Remarque - Les espaces

L'interpréteur BASIC ignore les espaces (ou blanc) séparant les composants d'une instruction. On les utilisera donc pour augmenter la lisibilité d'un programme.

Exemple : 4OIFK>1OTHEN6O
est équivalent à :
4O IF K > 1O THEN 6O

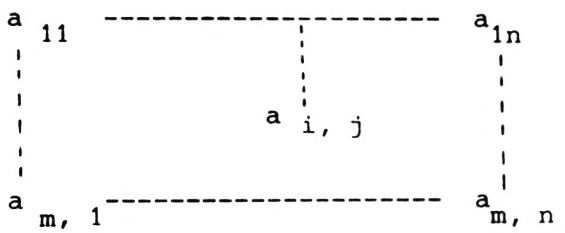
II 5 Les tableaux

a) définition

Les variables que nous avons étudié jusqu'à présent étaient des variables dites simples : c'est à dire qu'à un instant donné, elles ne pouvaient avoir qu'une valeur.

Or il est parfois utile de traiter des données structurées
Ainsi en mathématiques il est fréquent de manipuler des tableaux :

- à une dimension. Il s'agit dans ce cas de vecteurs ou de listes $(a_1 , a_2 \ \ldots \ a_i \ \ldots \ a_n)$
- à deux dimensions. Ce sont des matrices telles que



- à plusieurs dimensions.

i et j sont appelés indices. a_{ij} est un élément indicé. On l'appelle aussi élément générique, en effet, en faisant varier i entre 1 et m et j entre 1 et n, on décrit l'ensemble de la matrice.

b) Présentation

En BASIC, il est possible d'utiliser des tableaux à une ou deux dimensions.

Un élément de tableau sera représenté par une variable identifiant le tableau suivi par le (ou les) indice entre parenthèses.

Exemple - A (3) est le troisième élément de la liste A

-

- B (3,2) est l'élément situé dans la troisième ligne et la deuxième colonne du tableau B.

- C (I) représente l'élément générique d'un tableau à une dimension (ou liste) de valeurs numériques :

C (1) , C (2) , C (3) , ------ C (N).

Il est possible de définir également des listes ou tableaux de chaines de caractères. Pour cela, l'identificateur se terminera par le caractère $.

Exemple - D $ (J , K) est une variable indicée représentant l' élément générique d'un tableau de chaines de caractères.

III LES COMMANDES MONITEUR

Pour pouvoir mettre au point et exécuter un programme sous l'éditeur-interpréteur BASIC, il nous faut diposer de commandes permettant en particulier :

- d'insérer une instruction dans un programme
- de visualiser tout ou partie du programme déjà entré.
- de lancer l'exécution du programme
- de sauvegarder ou de charger un programme mémoire secondaire (cassette - disque - ruban perforé).

L'ensemble de ces ordres constitue les commandes moniteur du langage BASIC

Quand BASIC est prêt à acquérir une de ces commandes il affiche à la console un message l'indiquant :

Exemple : R E A D Y ou O K

III 1 Insertion - modification d'une ligne

Pour insérer ou modifier une ligne BASIC, il suffit de frapper au clavier son numéro puis son contenu. La fin de la ligne sera marquée par le caractère "Retour chariot" (ou Return).

Exemple :

```
O K
#  50  A = 15
O K
#
```

(Les caractères frappés par l'utilisateur sont soulignés).

La ligne crée sera alors insérée à sa place dans le programme, les lignes étant classées par numéros croissants.

Remarque : Pour supprimer une ligne, il suffit de frapper son numéro suivie d'u caractère "retour chariot", ceci ayant pour effet de remplacer une ligne existante par une ligne vide (c'est-à-dire rien).

2 Exécution d'un programme - Commande RUN

Cette commande permet d'exécuter un programme à partir de la première ligne.

La forme : R U N I permet de lancer le programme à partir de la ligne I.

Exemple : R U N 30

3 Visualisation du programme - Commande L I S T

Cette commande permet de visualiser sur l'écran tout ou partie d'un programme. Elle peut se présenter sous trois formes :

. L I S T	affichage de tout le programme.
. L I S T i - j	affichages de lignes du programmes depuis la ligne i jusque la ligne j.
. L I S T k	affichage de la ligne k.

Exemples : L I S T 10 - 50

L I S T 30

4 Sauvegarde d'un programme - Commande S A V E

Le programme résident peut-être sauvegardé sur support magnétique (cassette ou disque souple). Pour ce faire, il suffit de frapper au clavier la commande S A V E.

Remarques

1 Si le périphérique est une cassette, il faut avoir mis en route le lecteur / enregistreur de cassettes avant de frapper "S A V E".

2 Certaines versions plus évoluées de BASIC permettent d'associer au programme sauvegardé un nom de fichier. Dans ce cas, la commande s'écrit :

S A V E " < nom de fichier > "

Exemple : l'ordre S A V E " T O T O" permet de sauvegarder le programme résident dans un fichier de nom T O T O.

5 Chargement d'un programme - Commande L O A D

Cette commande permet de charger en mémoire un programme préalablement sauvegardé sur périphérique. Si à l'exécution de la commande L O A D , un autre programme est présent en mémoire centrale il sera détruit.

Remarques

1 Si le périphérique est une cassette, il est nécessaire avant de frapper la commande de rebobiner la cassette et de mettre le lecteur en route.

2 Dans le cas des interpréteurs BASIC associant un nom au programme sauvegardé, la syntaxe de la commande sera :

L O A D " nom du fichier "

6 Commande N E W

Dans le cas où un utilisateur, veut créer un nouveau programme, il devra au préalable frapper la commande N E W qui a pour effet de vider la mémoire centrale.
Un autre programme peut alors être élaboré.

IV LES INSTRUCTIONS DE BASIC

Dans ce chapitre, nous allons examiner les différentes instructions de BASIC.Celles-ci peuvent se décomposer en cinq groupes qui sont :

- les instructions d'entrée - sortie
- les instructions de calcul et traitement
- les instructions de test et de rupture de séquence
- les instructions d'arrêt
- les instructions non exécutables.

Dans un but de simplification, nous avons adopté un certain nombre de conventions pour la présentation de la syntaxe des diverses instructions :

- Les caractères majuscules doivent être frappés tels quels.
- Les rubriques entourées de < > contiennent des explica - tions.
- Les rubriques entre [ ] sont optionnelles.
- Les informations entre[ ]* peuvent être répétées.
- Toutes les ponctuations de la syntaxe doivent être respectées.

D'autre part, dans les exemples proposés, les caractères frappés par l'utilisateur sont soulignés.

IV 1) Les instructions d'entrée / sortie

Un programme s'exécutant à généralement besoin de communiquer avec l'extérieur, d'une part pour acquérir des données, d'autre part pour communiquer ses résultats. En BASIC, les instructions qui assurent ces fonctions sont :

- I N P U T (entrer)
- P R I N T (imprimer)
- R E A D (lire)

- R E S T O R E (relire)

- D A T A (données)

a) L'instruction I N P U T

Cette instruction permet à l'utilisateur d'introduire des données pendant l'exécution d'un programme.

Sa structure générale est :

I N P U T [<"chaine de caractères ">] ; <liste de variables>

La liste contient des variables numériques ou alphanumériques séparées par des virgules.

L'exécution de cette instruction provoque l'impression de la chaine de caractères puis d'un point d'interrogation. Le calculateur se met alors en attente. Les données frappées sont affectées aux variables de la liste suivant l'ordre d'entrée. Ces données doivent être séparées par des virgules. Si le type d'une donnée entrée n'est pas conforme à celui de la variable à affecter, un message d'erreur est visualisé et le calculateur attend une nouvelle donnée.

Exemples :

```
10 I N P U T  A, B $
R U N
  ? 10. 3, IL FAIT BEAU
```

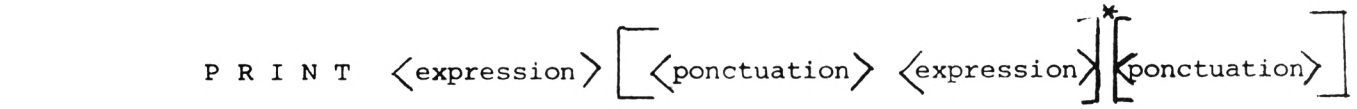
Cette exécution a permis d'affecter à la variable A la valeur 10. 3 et à variable B $ la chaine de caractères "IL FAIT BEAU"

```
. 10 I N P U T  "PRIX UNITAIRE ",  P 1
R U N
PRIX UNITAIRE ? TROIS        Erreur, BASIC attend un nombre
REDO FROM START              REDO FROM START signifie :
 ?  2.53                     Retour au début de l'entrée.
```

b) L'instruction P R I N T

Cet instruction a pour but de visualiser sur la console des messages ou le contenu de variables .

Sa syntaxe est :



L'expression peut être de type chaine de caractère ou numérique, (nous étudierons plus en détail les expressions au paragraphe n° 2)

La ponctuation permet de cadrer l'expression suivante. Un point-virgule la fera imprimer immédiatement après la précédente , une virgule la placera au début de la prochaine zone de 14 caractères. Une absence de ponctuation à la fin d'un ordre P R I N T placera la prochaine édition au début de la ligne suivante.

Exemples :

```
10 I N P U T "NOM" ; N$

20 I N P U T "DATE DE NAISSANCE" ; I, M, A

30 P R I N T N$ ; "EST NE LE" ;

40 P R I N T I ; " - " ; M ; " - " ; A

R U N

NOM ? DUPONT MARCEL

DATE DE NAISSANCE ? 10 , 3 , 1934

DUPONT MARCEL EST NE LE 10 - 3 - 1934
```

. 10 I N P U T A , B

20 P R I N T A ; B

30 P R I N T A, B

R U N

? 3 , 77

3 77

c) Les instructions R E A D - D A T A - R E S T O R E

L'instruction R E A D permet d'assigner à des variables des valeurs situées dans des instructions D A T A.

La syntaxe de cette instruction est :

R E A D <liste de variables>

Les variables peuvent être numériques ou alphanumériques et sont séparées par des virgules.

Les instructions D A T A associées ont pour syntaxe :

D A T A <liste de valeurs>

De même ces valeurs doivent être séparées par des virgules, les chaines de caractères pouvant être placées entre (").

A l'exécution du R E A D , les valeurs des constantes des instructions D A T A sont affectées aux variables de la liste de l'instruction R E A D.

Il peut y avoir plusieurs instructions D A T A dans un même programme. L'exploration des instructions D A T A se fait dans l'ordre où elles apparaissent dans le programme.

Si toutes les données déclarées ont été lues et qu'un nouvel ordre R E A D est rencontré un message d'erreur s'affiche sur la console : " OUT OF DATA" (Manque de données).

L'instruction R E S T O R E (relire) permet de préparer la relecture des mêmes données.

Exemples

- le programme

```
10 D A T A  1 , 3 , M O T
20 R E A D  A , B , C $
```

est équivalent à

```
10 D A T A  1 , 3
20 R E A D  A , B , C $
100 D A T A  M O T
```

et permet d'affecter à A et B respectivement les valeurs 1 et 3 ; à C $ la chaine de caractères "M O T "

```
10 R E A D  A , B
20 R E S T O R E
30 R E A D  C $
40 D A T A  136 , 7
```

Ce programme permet d'affecter à A la valeur numérique 136, à B la valeur 7 et à C $ la chaine de caractères "136"

IV 2 Les instructions d'affectation

Ces instructions correspondent aux opérations de calcul et de traitement. Elles permettent de modifier de manière dynamique la valeur d'une variable dans l'exécution d'un programme. En BASIC standard, ces instructions commencent par le mot "LET" (en français "Soit", qui correspond au "Faire" des organigrammes). La structure générale de ces instructions est la suivante. :

L E T ⟨variable⟩ = ⟨expression⟩

Cette opération consiste à estimer le résultat de l'expression située à droite du signe égal et affecter la valeur ainsi obtenue à la variable dont l'identificateur est situé à gauche du signe égal.

Dans la plupart des BASIC, le mot "LET" peut-être omis. Ainsi les lignes suivantes sont équivalentes :

```
10      LET     A = 1
```

et

```
10      A = 1
```

Dans la suite et afin d'alléger l'écriture, le mot "LET" est supprimé.

Pour que l'instruction soit correcte, il faudra que le résultat de l'expression soit du même type que la variable ; expression arithmétique si la variable est numérique et manipulation de chaines de caractères si la variable est de type chaine de caractères.

L'expression peut-être très simple :

- constante

```
10  A$ = "ABC"
```

- variable

```
10  B = C
```

Dans ce cas, la dernière valeur affectés à C sera affectée à B. Il apparait ici la nécessité que toute variable apparaissant dans une expression se soit vue affectée une valeur avant l'exécution de l'instruction où figure cette variable (par un "LET" , "INPUT" , "READ"). Par défaut, une valeur 0 sera prise pour une variable numérique, et une chaine de caractère de longueur nulle pour une variable de type chaine.

Dans le cas où l'expression fait intervenir plusieurs variables ou constantes, sa construction doit obéir à certaines règles.

a) les expressions arithmétiques

Une expression arithmétique consiste en une suite d'identificateurs de variable de constantes et de fonctions à résultat numérique séparés par des opérateurs arithmétiques.

Une expression peut faire appel aux cinq opérateurs suivants :

- addition : " + "
- soustraction : " - "
- multiplication : " * "
- division : " / "
- élévation a une puissance : "↑ "

Toute expression mathématique devra être traduite à l'aide de ces opérateurs, toutes les opérations devant être explicitées. Ainsi, l'expression

$$b^2 - 4 a c$$

se traduira

B↑2 - 4 * A * C

Pour que le résultat de cette expression puisse être calculé par l'ordinateur, il est nécessaire de définir dans quel ordre seront effectuées les opérations élémentaires, afin qu'une expression ait une traduction unique.

Considérons tout d'abord les expressions arithmétiques ne faisant pas intervenir de parenthèses. On applique alors les règles suivantes :

- il existe une hiérarchie entre les opérateurs arithmétiques. Les opérations d'exponentiation sont évaluées les premières (priorité de niveau 1), puis les opérations de multiplication et de division (priorité 2), enfin les opérations d'addition et de soustraction (priorité 3).

- dans un niveau de priorité donné, les opérations sont effectuées de manière successives, de la gauche vers la droite.

1er exemple. L'expression

```
10  S = B↑2 + C * D
```

sera donné comme suit (les minuscules correspondant à des variables intermédiaires)

```
a = B ↑ 2          priorité 1
b = C * D          priorité 2
c = a + b          priorité 3
S = c              affectation du résultat
```

soit $S = B^2 + C \cdot D$

2 ème exemple

```
10  s = A + B↑2↑C + D * E/F * G↑2 - H

        a = B ↑ 2

        b = a ↑ C

        C = G ↑ 2

        d = D * 2

        e = d / F

        f = e * C

        g = A + b

        h = g + F

        i = h - H

        S = i
```

soit $S = A + B^{2C} + \frac{D.\ E.\ G^2}{F} - H$

Dans certains cas, la complexité de l'instruction arithmétique nécessite un parenthésage. En effet une expression comme

$$S = \frac{A + \frac{B}{C}}{D + \frac{E}{F}}$$

n'est pas directement traduisible, en se contentant des deux règles précédentes. Aussi est-on obligé d'introduire des parenthèses, avec la règle suivante :

- dans une expression comportant des parenthèses, les expressions entre parentèses sont évaluées avec une priorité supérieure à tous les autres opérateurs.

Ainsi l'expression précédente pourra s'écrire :

```
10      S = (A + B / C)  / (D + E / F)
```

et sera exécutée :

```
a = B / C

b = A + a

c = E / F

d = D + C

e = b / d

S = e
```

Dans le cas où plusieurs niveaux de parenthèses seraient utilisés, c'est le niveau le plus intérieur qui sera le premier exécuté.

Exemple :

```
10  S = (A + B) * ((C + D) ↑ (1 + E (F + G )) + H ) / (I + J)
```

sera exécuté :

```
a  =  F + G                          niveau 3

b  =  C + D                          niveau 2

c  =  E * a

d  =  1 + C

e  =  A + B                          niveau 1
```

```
f = b ↑ d
g = f + H
h = I + J
i = e * g                                   niveau 0
j = i / h
S = j
```

soit $S = \dfrac{(A + B)\ ((C + D)^{(I + E\ (F + G))} + H)}{(I + J)}$

Il existe une exception aux règles définies précédemment. il s'agit du signe " - " utilisé en opérateur de négation, comme dans les expressions suivantes :

```
10   A = B ↑ - C
20   C = - A * B + C
```

ces lignes seront exécutées comme si elles étaient écrites :

```
10   A = B ↑ ( - C)
20   C = (- A) * B + C
```

ceci correspond à l'application des règles :

- l'opérateur de négation suivant un opérateur d'exponentiation est effectué avant ce dernier (cas de la ligne 10).

- dans les autres cas, l'opérateur de négation est effectué avec une priorité intermédiaire entre les niveaux 1 et 2 (après les exponentiations, mais avant les multiplications et divisions).

Remarque : dans certains cas, le résultat de l'expression arithmétique doit être un nombre entier. C'est le cas des indices de tableaux, ainsi que des paramètres de certaines fonctions. Dans ce cas, si le résultat de l'expression n'est pas un nombre entier, le résultat est ramené à sa partie entière.

Exemple :

```
10   A = 3.6

20   B (A) = 1

30   B (A + .5) = 2

40   PRINT B (3) , B (4)

RUN

 1            2
```

b) Manipulations de chaines de caractères

Les opérations que l'on peut définir sur des chaines de caractères sont très réduites. Elles se limitent à des opérations de concatenation, c'est à dire à la mise bout à bout des chaines de caractères. Le symbole de la concaténation est " + " . Cet opérateur ne peut-être confondu avec le symbole de l'addition, le type des variables opérandes étant différent.

Exemple

```
10      A$ = "BONJOUR "

20      B$ = "MONSIEUR"

30      C$ = "MADAME"

40      D$ = A$ + B$

50      PRINT D$

60      PRINT A$ + C$

RUN
```

BONJOUR MONSIEUR

BONJOUR MADAME

Comme il n'existe qu'un opérateur les règles d'exécution des instructions de manipulation de chaines sont peu nombreuses :

- s'il n'y a pas de parenthésage, les opérations sont effectuées de manière successive, de la gauche vers la droite.

- dans le cas où il y a un parenthésage, les opérations sont effectuées en commençant par le niveau le plus intérieur.

L'opération de concatenation étant associative, il faut noter que le parenthésage n'offre aucun intérêt, le résultat en étant indépendant (il ne dépend que de l'ordre dans lequel sont placées les chaines à concatener). La deuxième règle n'a en fait à intervenir que dans le cas d'utilisation de fonctions de manipulations de chaines de caractères.

IV 3) Les instructions de tests et de rupture de séquence

Jusqu'à présent, les programmes cités en exemples avaient un déroulement séquentiel, c'est à dire que les instructions étaient exécutées dans l'ordre croissant des numéros de ligne.

Dans ce paragraphe,nous étudierons les instructions permettant de rompre la séquentialité du programme.

a) L'instruction G O T O

GOTO signifie "aller à". La syntaxe de cette instruction est :

GOTO < numéro de ligne >

Son but est de transférer l'exécution du programme au numéro de ligne indiqué.

Exemple :

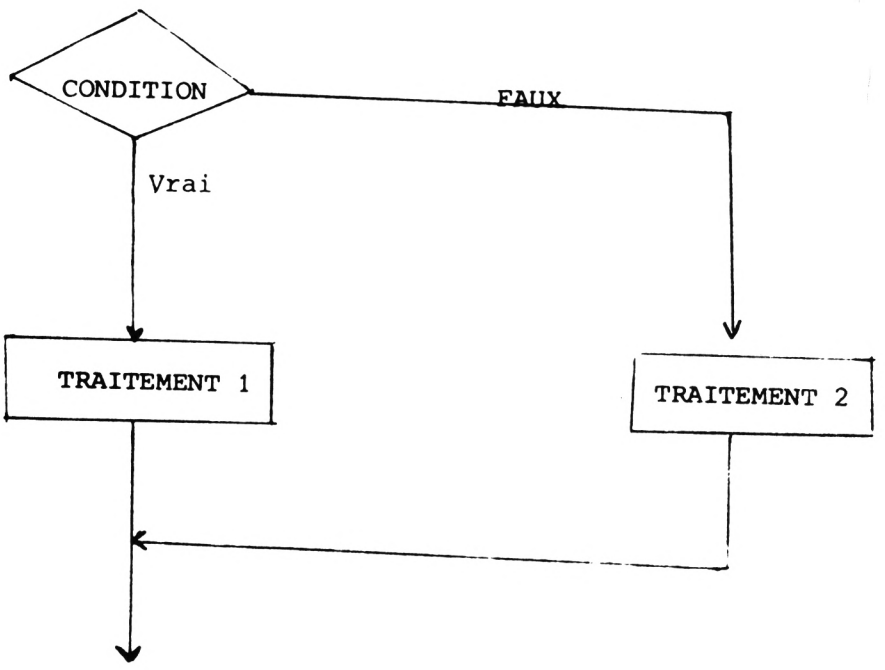
```
10   I = 1
20   I 2 = I * I
30   PRINT "LE CARRE DE" ; I ; "EST" ; I 2
40   I = I + 1
50   GOTO 20
```

Ce programme calcule et imprime le carré de tous les nombres entiers.

```
R U N

LE CARRE DE  1 EST 1
LE CARRE DE  2 EST 4
LE CARRE DE  3 EST 9
   --------
```

b) L'instruction IF... THEN...

Cette instruction permet de programmer les tests. Un test se représente par l'organigramme suivant :



Ceci peut s'écrire :

SI ⟨condition vraie⟩ ALORS ⟨faire traitment 1⟩ SINON ⟨faire traitement 2⟩

En BASIC standard, on ne dispose pas d'instruction de type : SI... ALORS ... SINON... Par contre, il est possible d'utiliser une instruction de type SI... ALORS... dont la syntaxe est :

I F ⟨expression conditionnele⟩ T H E N ⟨instruction⟩

Une expression conditionnelle peut avoir deux résultats : vrai ou faux. Elle peut-être simple ou complexe.

Si elle est simple, elle est composée de deux expressions du même type (arithmétique ou chaine de caractères) séparées par un opérateur relationnel. Ces opérateurs en BASIC sont les suivants :

= égalité

> supérieur

< inférieur

<> différent

>= supérieur ou égal

<= inférieur ou égal

Exemples A = B + 2 : A est égal à B + 2 (VRAI ou FAUX ?)
C >= 2 : C supérieur ou égal à 2

Une relation d'ordre entre deux chaines de caractères permet de les comparer suivant l'ordre alphabétique :

Exemples . A $ > B $ veut dire : la chaine A $ est elle classée alphabétiquement après la chaine B $

. C $ = D $ signifie : la chaine C $ est-elle identique à la chaine D $.

Remarque : La relation ordre utilisée consulte les caractères

numériques comme inférieurs aux caractères alphabétiques.

Une expression relationnelle complexe est composée d'expressions relationnelles simples séparées par des opérateurs logiques qui sont en BASIC

N O T (non)

O R (ou)

A N D (et)

L'expression conditionnelle (A > B AND C = D) sera considéré comme vraie si la relation A > B est vraie et la relation C = D est vraie.

L'expression (A>B OR C = D) sera vraie si au moins unes des deux conditions (A > B , C = D) est vraie.

L'opérateur logique NOT réalise la fonction suivante :
NOT <condition> est vrai quand <condition> est faux et inversement.

Exemple (NOT A > B) est identique à A = B)

Ces opérateurs logiques permettentdonc de tester des conditions complèxes telles que :

. S I condition 1 . E T condition 2 A L O R S

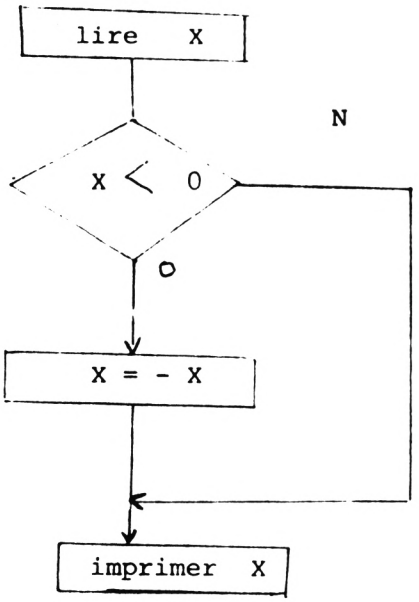
. S I condition 1 O U condition 2 A L O R S

. S I condition 1 E T N O N condition 2 A L O R S

Exemples d'utilisation de l'instruction I F ... T H E N :

- Lire un nombre et imprimer sa valeur absolue

L'organigramme de ce programme peut s'écrire :



Ceci se code en BASIC :

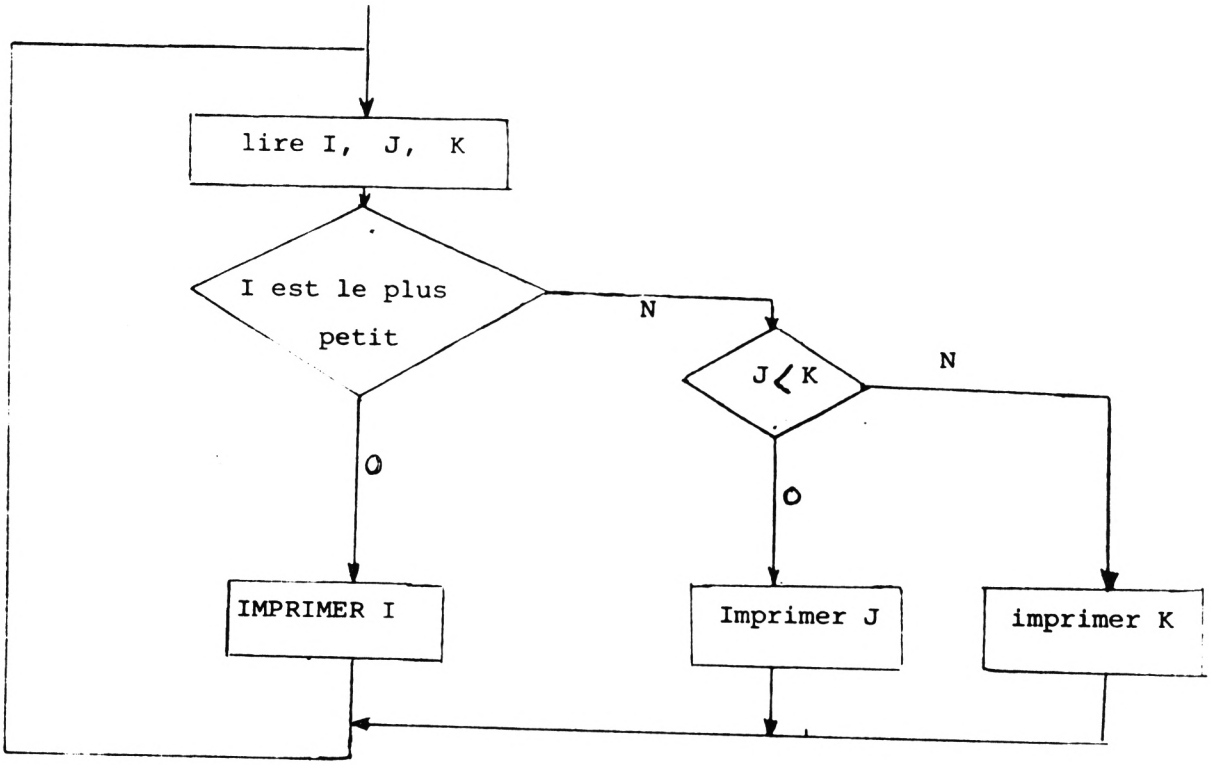
```
10 INPUT "NOMBRE" ; X

20 I F X< 0 THEN X = - X

30 PRINT "VALEUR ABSOLUE :" ; X

R U N

NOMBRE ? - 3. 27
VALEUR ABSOLUE : 3. 27

R U N

NOMBRE ? 3. 27
VALEUR ABSOLUE : 3. 27
```

● Imprimer la plus petite parmi trois valeurs

```
10 INPUT I, J, k
20 IF NOT (I< J AND I< K) THEN GOTO 50
30 PRINT I
40 GOTO 10
```

```
50  IF J<K  THEN GOTO 80
60  PRINT   K
70  GOTO  10
80  PRINT  J
90  GOTO  10
```

L'organigramme peut se faire de la manière suivante :



La condition de la ligne 20 est : I n'est pas le plus petit .

c) L'instruction FOR....NEXT

Soit à calculer la somme des carrés des dix premiers entiers. Le programme peut s'écrire :

```
10  S = 0
20  I = 1
30  S = S+ I↑2
40  I = I + 1
50  IF  I < = 10        THEN GOTO 30
```

Pour ce type de problème, où on connait d'avance le nombre d'itérations, l'instruction FOR offre une solution plus élégante.

Sa structure est la suivante :

FOR V=<expression arithmétique> TO <exp. arith.> STEP <exp. arith.>
:
N E X T V instructions de l'itération

L'exemple précédent peut alors se réecrire :

```
10   S  = 0
20   FOR I = 1  TO  10 STEP 1
30   S  =  S + I ↑ 2
40   NEXT  I
```

I est ici le compteur de boucle (ou variable de contrôle).

A la première exécution de l'itération, I prendra la valeur qui lui est affectée dans l'instruction "F O R".

Puis à chaque passage, sa valeur sera incrémentée de la valeur indiquée derrière le mot clé S T E P (=pas).

L'instruction N E X T suivie de la variable de contrôle permet de délimiter les instructions sur lesquelles doit porter la boucle.

Quand la variable de contrôle aura dépassé la valeur limite placée derrière le mot - clé T O = jusque), le contrôle sera donné à l'instruction qui suit l'instruction N E X T.

Exemples

. Calculer la somme des éléments d'un tableau - à deux dimensions de N lignes par M colonnes.

En mathématiques cette somme s'écrit sous la forme :

$$S = \sum_{I=1}^{N} \sum_{J=1}^{M} a_{ij}$$

ou

ou

$$S = \sum_{I=1}^{N} \left(\sum_{J=1}^{M} a_{ij} \right)$$

La partie située entre parenthèses représentant la somme des éléments d'une ligne.

Ceci peut aisément se programmer à l'aide de deux boucles FOR imbriquées:

```
100   S = 0
110   FOR I = 1 T O  N   STEP 1
120   FOR J = 1 TO   M   STEP 1
130   S = S + A (I,J)
140   NEXT J
150   NEXT I
```

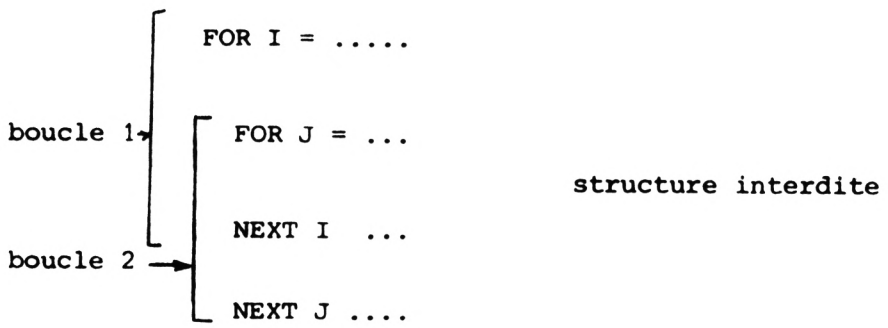
Remarques • Quand le pas (STEP) d'une instruction "FOR" est 1 il peut être omis. Ainsi la ligne 110 peut s'écrire plus simplement:

```
110  FOR I = 1 TO N
```

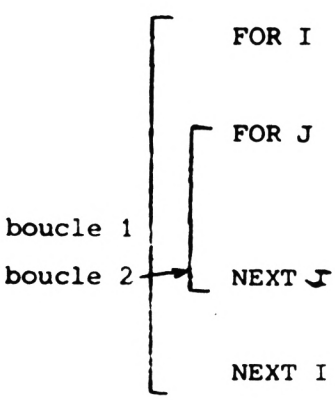
• Les lignes 140 et 150 peuvent être condensées de la manière suivante :

```
140  NEXT  J , I
```

• Deux boucles "FOR" peuvent être imbriquées, mais pas croisées.



structure interdite



structure permise en BASIC

IV 4) Les instructions d'arrêt

a) l'instruction **END**

L'exécution d'un programme BASIC s'arrête de lui même quand la dernière instruction interprétée portait le numéro le plus élevé (à condition bien sûr que celle-ci ne soit pas une instruction de branchement).

L'instruction END permet de terminer l'exécution d'un programme. Dans de nombreuses versions de BASIC, cette instruction doit être la dernière du programme.

Exemple d'utilisation :

Lire un nombre, indiquer s'il est positif ou négatif :

```
10  INPUT A
20  IF A> 0  THEN GOTO 50
30  PRINT  "NEGATIF"
40  GOTO 60
50  PRINT  "POSITIF"
60  END
```

Dans ce cas, l'instruction END est indispensable, elle permet en effet de créer une ligne sur laquelle le programme pourra s'arrêter et ne comportant aucune "action".

b) L'instruction STOP

Cette instruction placée n'importe où dans le programme provoque à sa rencontre, l'arrêt du porgramme et l'affichage d'un message du type :

BREAK I N < n° de ligne > (arrêt ligne...)

La commande moniteur CONT permet de relancer l'exécution du programme là où elle avait été arrêtée.

Exemple :

```
10  PRINT "TOTO"
20  STOP
30  PRINT "LULU"
```

R U N

TOTO
BREAK IN 20
CONT
LULU

IV 5) Les instructions non exécutables

a) L'instruction DIM

Il n'existe pas à proprement parler de déclaration de variables en BASIC. Nous avons vu que le type d'une variable dépendait uniquement des caractères composant son identificateur (Exemple : A $ représente toujours une chaine de caractères).

Cependant, il existe une instruction (DIM) permettant de dimensionner les tableaux. Elle permet de définir le nombre d'éléments maximum d'une liste ou d'un tableau. Celui-ci doit être déclaré avant tout utilisation. Par défaut, sa taille sera prise à 10 éléments pour une liste et à (10 X 10) éléments pour un tableau à deux dimentions.

Exemples

- 10 DIM A $ (20)

Cette déclaration réserve uneliste de 20 chaines de caractères et de nom A $, dont les éléments sont A $ (0) , A $ (1) .... A $ (19)

- 10 DIM B (13, 20)

Cette instruction déclare un tableau numérique B à deux dimensions de taille 13 X 20.

Remarques

Il est possible de définir la dimension d'un tableau par une variable numérique considérée comme une donnée qui sera lue au moment de l'exécution :

Exemple
```
10  INPUT (N)
20  DIM A (N)
```

Un indice de tableau doit être un entier positif ou nul, inférieur à la dimension déclarée. Il peut être une constante, une variable ou une expression arithmétique.

b) L'instruction REM

Cette instruction permet d'insérer des commentaires dans un programme BASIC. Sa syntaxe est la suivante :

REM < TEXTE >

A la rencontre de cet instruction, l'interprèteur passe immédiatement à l'instruction suivante.

Exemple:

```
10  REM  RANGER   " ABC" DANS  A $
20  REM    * * * * * * * * * * * *
30       A $  = "ABC"
```

IV 6 Les fonctions BASIC

Les fonctions en BASIC peuvent être classées en trois catégories :

- fonctions mathématiques standards. C'est leur existence qui fait de BASIC un langage scientifique.
- fonctions de manipulation de chaines de caractères, qui rendent le traitement de ces chaines possible en BASIC. L'opération de concaténation, si elle est nécessaire, n'est pas suffisante : ainsi elle ne permet pas d'extraire une chaine d'une chaine plus importante.
- fonctions définies par l'utilisateur. Ces fonctions, qui peuvent être nécessaires dans une application, seront alors utilisables dans tout le programme où elles ont été définies.

Les deux premières catégories correspondent aux fonctions dites

de bibliothèque. Le problème qui se pose est que cette bibliothèque est plus ou moins complète selon les interprèteurs : par rapport au standard, certaines fonctions peuventmanquer et d'autre avoir été rajoutées. Aussi, dans le cas où le programme doit être utilisé sur des machines différentes, il faudra faire très attention à la transportabilité des séquences utilisant des fonctions.

a) fonctions mathématiques standards.

La syntaxe est la suivante :

<nom de fonction>(<expression arithmétique

Ces fonctions pourront être utilisées dans toutes les expressions arithmétiques. Lors de l'exécution de l'expression, leur niveau de priorité sera supérieur à celui des opérateurs arithmétiques, la règle concernant le parenthésage restant inchangée. Ainsi l'expression suivante :

```
10  S = SIN ( X ) ↑ 2
```

sera équivalente à :

```
10  S =  (SIN  (X))↑ 2
```

Les fonctions couramment disponibles en BASIC sont :

- valeur absolue d'un argument

```
ABS (X)  = X   pour X ≥ 0
ABS (X)  = -X  pour X < 0
```

- racine carrée d'un argument : S Q R (X). Sous peine d'erreur, l'argument doit bien évidemment être positif.

- signe d'un argument

```
SGN (X) = 1     pour  X > 0
SGN (X) = 0     pour  X = 0
SGN (X) = 1     pour  X < 0
```

Exemple : transfert de signe

```
10  INPUT A, B
20  B = SGN (B)
30  IF B = 0 THEN B = 1
40  A = ABS (A) * B
50  PRINT A

R U N
? 3, - 2
- 3
R U N
? 3, 0
3
```

- partie entière d'un argument. INT (X). Le résultat est le nombre entier immédiatement inférieur ou égal à l'argument. Cette fonction est utilisée de manière implicite lorsque le résultat d'une expression devrait être entier et ne l'est pas. Ainsi le programme

```
10  A = 1. 6
20  B = 2. 15
30  C (A,B) = 3.2
```

est équivalent, quant au résultat, au programme dont la ligne 30 serait :

```
30 C (INT (A) , INT (B)) = 3. 2
```

Exemple

```
10 INPUT A
20 PRINT INT (A)
R U N
? 2. 15
2
R U N
? - 2. 15
- 3
```

- fonctions trigonométriques. Les fonctions standards sont

sinus	SIN (X)
cosinus	COS (X)
tangente	TAN (X)
arctangente	ATN (X)

Dans les trois premiers cas, l'argument est considéré exprimé en radians, et l'arctangente donne un résultat lui aussi en radians. Les autres fonctions trigonométriques peuvent être obtenues à partir des fonctions standard. Ainsi, par exemple : l'arcsinus :

```
10  S = ATN (X/SQR (1 - X↑2))
```

- fonction exponentielle EXP (X)

Cette fonction correspond à e^x, où représente la constante de NEPER. On a donc équivalence entre :

```
20  S = EXP (A)
```

et la séquence:

```
10  E = 2. 71828183
20  S = E↑A
```

La fonction exponentielle permet de calculer les fonctions hyperboliques. Ainsi le sinus hyperbolique est donné par :

```
10  S = (EXP (X) - EXP (-X)) / 2
```

- fonction logarithmique LOG (X)

Pour X positif strictement, cette fonction donne le logarithme dans la base e. On obtient le logarithme dans une base quelconque par la relation.

$$\text{Log}_a (X) = \frac{\text{Log}_e (X)}{\text{Log}_e (a)}$$

AInsi, on pourra calculer le logarithme décimal par

```
10   S = LOG (X) / LOG (10)
```

- génération d'un nombre aléatoire : RND (X)

Cette fonction permet l'obtention d'un nombre aléatoire compris entre O et 1. Le paramètre n'est pas obligatoire sur tous les BASIC. Si le paramètre est nul (ou omis lorsque cela est possible), on obtiend à chaque appel un nombre différent, de valeur aléatoire comprise entre

0 et 1. Si le paramètre X est différent de 0 (ou donné alors qu'il peut être omis) , alors le comportement de la fonction RND dépend de l'interprèteur.

Exemple

```
10 FOR I = 1 TO 5
20 PRINT RND
30 NEXT I

R U N

 .504388
 .733821
 .593819
 .723753
 .356224

20 PRINT RND (0)
R U N
 .811635
 .811635
 .811635
 .811635
 .811635
```

A partir d'une telle fonction, il est possible d'obtenir une loi de probabilité quelconque. Ainsi, si l'on veut un tirage aléatoire d'un nombre compris entre A et B, on pourra faire

```
10  N = INT ((B-A+1) * RND + A)
```

Ainsi un programme de tirage du LOTO pourrait s'écrite :

```
10 REM TIRAGE des 6 NUMEROS
20  FOR  I = I TO 6
30 PRINT INT (49 X RND + I) ;
40 NEXT I
50 PRINT
60 REM TIRAGE COMPLEMENTAIRE
50 PRINT INT (49 * RND + I)

R U N
```

```
25 36 30 35 18 40
34
```

```
R U N

25 36 30 35 18 40
34
```

Sur certains BASIC, si on relance le programme, on obtiendra toujours la même séquence comme si dessus. On dispose alors d'une fonction spéciale RANDOMIZE qui permet d'initialiser différemment le générateur de nombre aléatoire à chaque exécution.
Ainsi, si on rajoute au programme précédent, la ligne suivante, on obtiend :

```
5 RANDOMIZE

R U N

RANDOM NUMBER SEED ( - 32768 TO 32767) ? 7

26 49 13 46 2 38
40
```

Une caractéristique importante des fonctions mathématiques est la précision du calcul, qui dépend des interprèteurs. Le plus souvent, le résultat est donné avec 6 chiffres significatifs, avec une erreur de $\pm$ 1 unité sur le dernier. Il peut en résulter certianes erreurs :

```
10  S = ATN (TAN (.987654)) - . 987654
20 PRINT S

R U N

8. 9407E - 0 8
```

Cette erreur, bien que très faible, peut occasionner des problèmes en cas de test.

b) fonctions de manipulation de chaines de caractères

Ces fonctions peuvent être séparées en deux groupes

- Les fonctions à résultat "chaine de caractères", qui pourront figurer dans les expressions de type concaténation

- Les fonctions à résultat numérique, qui pourront figurer dans les expressions arithmétiques.

Avant d'étudier ces différentes fonctions, il est nécessaire de préciser comment s'effectue en machine la représentation d'un caractère. Nous avons vu dans la première partie qu'une machine informatique n'est capable de manipuler que des mots constitués d'information binaires.

Ces mots, selon le contexte où ils sont employés peuvent représenter des instructions, des données, des adresses. Les caractères seront représentés de la même manière, et chacun se verra attribuer un code. Le plus utilisé est le code ASCII (American Standard Communication Informations Interchange). Ce code utilise 7 éléments binaires et permet donc de représenter 128 caractèes (on trouvera la correspondance entre code ASCII et caractère en annexe). A chacun de ces codes, on peut également associer la valeur numérique (de 0 à 127) qu'il représente. On peut ainsi établir des comparaisons entre caractères.
Dans l'instruction :

```
10 IF A$ > B $  THEN ...
```

on ramènera les deux chaines à des longueurs identiques en complétant la plus courte par des blancs (caractère visualisable de code le plus petit) et on testera si la valeur numérique de la chaine A $ (en fait, pour des raisons de simplicité, le test se fera caractère par caractère, en commencant par le début de la chaine et s'arrêtera lorsque les caractères de même rang dans A $ et B $ seront différents).

Dans ces fonctions, on pourra voir apparaitre deux types d'arguments.

- arguments chaine de caractère. Dans la liste d'appel, ils pourront être remplacés par une expression detype concaténation, qui sera exécutée avant l'appel à la fonction. Cette dernière portera alors sur la chaine résultante.

- arguments numériques, Dans le cas où cet argument doit prendre une valeur entière (nombre de caractères par exemple), la fonction INT sera appliquée de manière implicite.

Les fonctions de manipulation de chaines que l'on trouve le plus

souvent sont les suivantes :

- L E F T $: extraction du début d'une chaine de caractères. La forme d'utilisation est

L E F T $ (X $, N)

Le résultat est une chaine de caractères formée des N premiers caractères (en partant de la gauche) de X $

```
10  A $ = "BONJOUR MONSIEUR"
20  PRINT LEFT$ (A $ , 7)

R U N

BONJOUR
```

-R I G H T $: extraction de la fin d'une chaine de caractères. Ce sont alors les N derniers caractères (en partant de la droite) qui sont extraits.

```
Exemple
10  A $ = "BONJOUR MONSIEUR"
20  PRINT RIGHT$  (A $, 8)

R U N

MONSIEUR
```

-M I D $ permet d'extraire une sous chaine située dans le milieu d'une chaine de caractèes. A l'exécution de l'instruction :

10 B $ = M I D $(A $, N , M)

on extraiera de A $ une chaine de M caractères, le premier étant le caractère de rang N. Si N est supérieur à la longueur de la chaine A $, le résultat de la fonction sera une chaine de longueur nulle. Si la chaine A $ comporte moins de M caractères à partir du caractère de rang N, la chaine résultat ne comportera que les caractères existants.

Exemple

```
10 A $ = "BONJOUR MONSIEUR"
20 PRINT MID$ (A$, 4, 4)
30 PRINT MID $ (A$, 20, 4)
40 B $ = MID$ (A$, 12, 29)
50 C $ = MID$ (A$, 1, 7)
60 PRINT B $ + C $
```

```
R U N

JOUR
.
SIEURBONJOUR
```

- SPACES $: fournit une chaine de caractères composée de caractères "espace" : SPACES $ (3) donnera une chaine de 3 blancs.
Si on modifie le porgramme précédent comme suit,

```
60 PRINT  B $  +  SPACE$ (2) + C $
```

on obtient

```
R U N

J O U R

S I E U R . . B O N J O U R
```

- LEN permet de connaitre la longueur d'une chaine de caractères. Tous les caractères sont comptés, y compris les blancs qui peuvent se trouver à la fin de la chaine.

Exemple

```
10 A $  =  "BONJOUR MONSIEUR"

20 PRINT  LEN (A  $)

30 PRINT  LEN (A $  + SPACE $ (4))
```

```
R U N

16

20
```

En plus de ces fonctions, qui permettent réellement de manipuler les chaines de caractères, il existe des fonctions dites de conversion :

- ASC : conversion d'un caractère en code ASCII. Elle fournit le code ASCII du premier caractère de la chaine constituant l'argument.

Exemple

```
10  A $  = "BONJOUR MONSIEUR"
20  PRINT ASC (A$)

R U N

66
```

Ceci correspond bien (cf annexe) au code ASCII du "B" exprimé en décimal.

- CHR$: cette fonction est l'inverse de la précédente. Son résultat est le caractère correspondant au code ASCII fourni.

Exemple :

```
10  A  =  66
20  B  =  65
30  PRINT CHR$ (A) + CHR$  (B) + CHR$ (A) + CHR$ (B)

R U N

B A B A
```

- VAL : elle convertit une chaine de caractères en la valeur numérique qu'elle représente. Cette valeur numérique doit se trouver en tête de la chaine qui ne peut donc débuter que par un des caractères " + " , " - ", ou un chiffre de 0 à 9.

Exemple :

```
10 INPUT A $
20 B = VAL (A $)
30 PRINT B ; "NOVEMBRE"

R U N

? 14 JUIN
14 NOVEMBRE
```

Pour les manipulations de chaines en sortie (édition), il existe enfin deux fonctions qui permettent d'améliorer la mise en page :

- TAB : fonction de tabulation. Elle permet de commencer l'édition d'une chaine de caractères ou de la valeur d'une variable à partir d'une position donnée dans la ligne. Sa forme générale est

TAB (<expression arithmétique>)

le résultat de l'expression étant automatiquement convertit en entier.

Exemple

```
10 FOR I = 1 TO 6
20 PRINT TAB (I) ; " X X "
30 NEXT I

R U N
```

```
 X X
   X X
     X X
       X X
         X X
           X X
```

Il est possible d'utiliser plusieurs tabulations dans la même impression. En cas de recouvrement entre deux chaines, la deuxième sera décalée et éditée à la fin de la première.

Exemple

```
10 PRINT  TAB (3) ; "CH" ; TAB (7) ; "AI" ; TAB (8) ; "NE" ;

R U N

.. CH.. AINE)
```

- POS : donne la position du curseur sur l'écran. Elle s'écrit

```
POS (N)
```

mais la valeur N n'est pas prise en compte. Les positions sont numérotées à partir de 0.
Si au programme précédent on rajoute la ligne suivante, on obtient :

```
20 PRINT  POS (0)

R U N

.. CH..AINE)

10
```

c) Les fonctions définies par l'utilisateur

L'utilisateur peut définir dans un programme des fonctions supplémentaires qui pourront être utilisées dans toute la suite du programme. La ligne où une fonction est définie doit être exécutée avant la première utilisation de la fonction. La forme générale est la suivante :

⟨Label⟩ (⟨liste de variables⟩) = ⟨expression⟩

Le label de toute fonction utilisateur doit commencer par les caractères "FN" , et se terminer par "$" si le résultat de la fonction est une chaine de caractères. Les variables indiquées dans la liste sont des paramètres formels. Lors de l'appel de la fonction,ils seront remplacé dans le calcul de l'expression par les arguments figurant dans la liste d'appel.

Il faut noter que certains BASIC n'autorisent pas la définition de fonctions à résultat chaine de caractères et que, parfois,

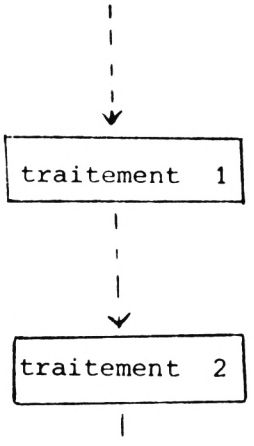
un seul paramètre formel est autorisé, la référence quant à ces limitations devant être le manuel du BASIC utilisé.

Exemple

```
10  DEF  FNPOL (X)  = X * X  + 2 * X  + 1

20  DEF  FNDEFI$ (A$ , B$ , N , M) = LEFT$ (A$ , N) + RIGHT$ (B$ , N )

30  DEF  FNCORAD (X)  =  3.1415927  *  X / 180

40  Y  = 4

50  Z  = 100

60  Z$ = "JEUDI 13 NOVEMBRE"

70  PRINT  FNPOL  (Y) ; FNCORAD  (Z)

80  PRINT  FNDEFi$ (Z$ ,  "MARDI"  , 5 , 2)

R U N

25  3.1415927

JEUDIDI
```

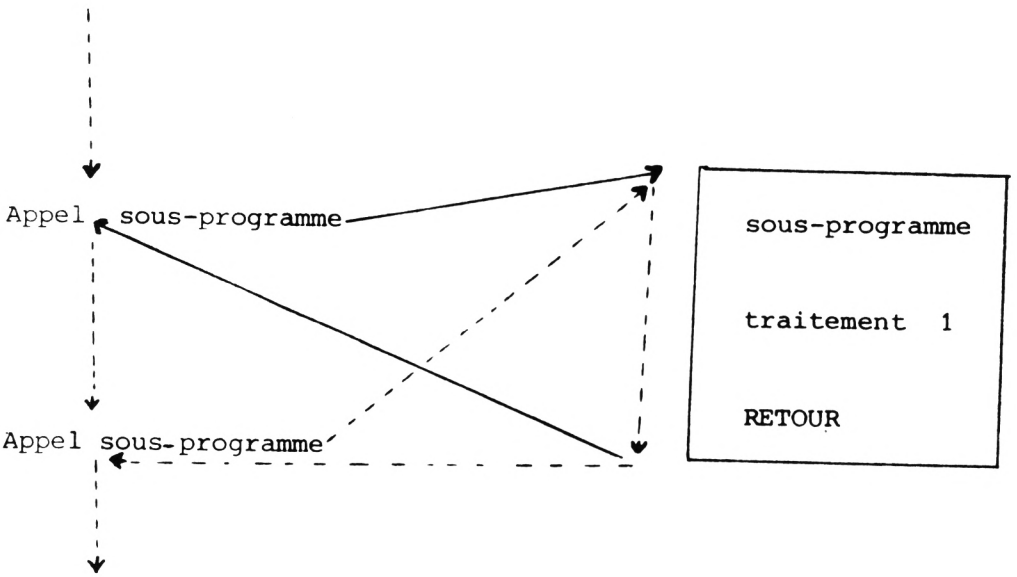
IV 7 Les sous programmes

Il est fréquent de devoir effectuer le même traitement en différents endroits d'un programme :



Dans une telle structure les instruction composant "traitement "1" devront être écrites plusieurs fois. On peut alors avoir recours à un sous-programme.

Un sous-programme est une suite d'instructions pouvant être appelé à partir d'un programme. Il se termine par une instruction de retour qui rend le contrôle au programme appelant. Cette structure peut se représenter de la manière suivante.



En BASIC, l'instruction d'appel de sous-programme est :

GOSUB numéro de ligne

SUBROUTINE veut dire sous-programme

Le numéro de ligne indiqué est celui de la première instruction du sous-programme.

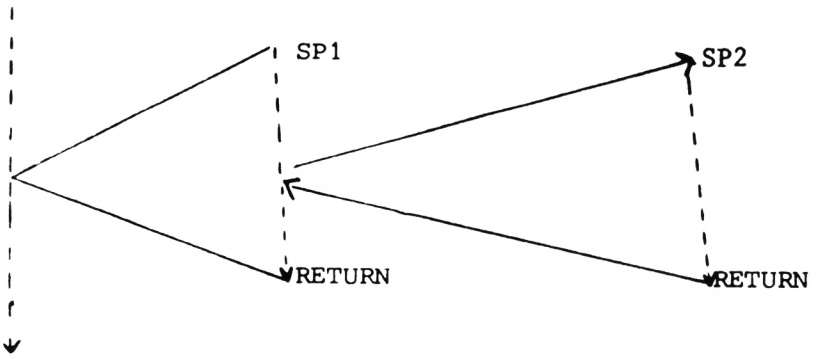
L'instruction de retour d'un sous-programme est : RETURN. Elle n'a aucun paramètre. A sa rencontre, l'interprèteur BASIC rend le contrôle à l'instruction qui suit l'instruction d'appel (GOSUB).

Remarques :

- Dans un même programme BASIC, il est possible d'insérer plusieurs sous-programme.

- A partir d'un sous-programme, il est possible d'en appeler un autre.

Nous pouvons donc nous trouver en face de structures telles que celle-ci :



programme principal

Le programme principal appelle SP1 qui, lui-même appelle SP2

Exemple :

Soit à écrire un programme calculant la somme des carrés des éléments d'une matrice A de N lignes par M colonnes.

Ce programme peut bien sûr s'écrire sans utiliser de sous-programme :

```
----

100  S  =  0
110  FOR  I  =  0  TO N - 1
120  FOR  J  =  0  TO M - 1
130  S = S + A (I , J) ↑ 2
140  NEXT  I , J
----
```

Utilisons pour l'écrire deux sous-programmes. Le premier calculera la somme des carrés des éléments d'une ligne. Le deuxième cumulera ces sommes partielles.

```
----
100    GOSUB     1000          programme   principal
----
```

```
1000      REM  SOMME DES CARRES
1010      S  =  0
1020      FOR  I  =  0 TO  N - 1
1030      GOSUB  1100
1040      S  =  S  +  S 1
1050      NEXT I
1060      RETURN
1100      REM
1110      REM SOMME DES CARRES  SUR LA LIGNE 1
1120      S 1  =  0
1130      FOR  J  = 0 TO  M - 1
1140      S 1  =  S 1 + A (I ,  J)↑ 2
1150      NEXT J
1160      RETURN
```

Dans ce cas précis, nous voyons que l'introduction d'un sous-programme augmente le nombre de lignes (ajout de GOSUB et RETURN). Cette structure prendra son intèrêt s'il faut utiliser les sous-programmes en plusieurs endroits du programme principal.

Il se peut, par exemple, qu'il soit nécessaire de connaître la somme des carrés des éléments de la ligne 3 de la matrice. Dans ce cas au lieu de réécrire :

```
----
200       I  =  3
210       S 1 = 0
220       FOR  J = 0 TO M - 1
230       S 1  =  S 1 + A (I , J)
240       NEXT  I
----
```

il suffira de faire :

```
----
200        I   =  3
210        GOSUB 1100
```

Remarque : Il faut noter qu'en BASIC, il n'est pas possible de passer au sous-programme des paramètres au moment de l'appel. Ceci est souvent gènant. En effet, dans l'exemple précédent, il aurait été commode que les