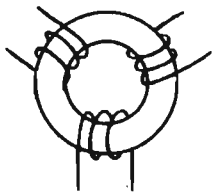
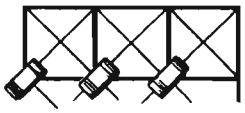
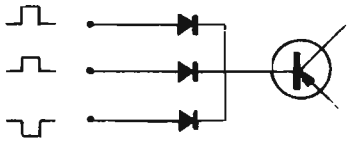


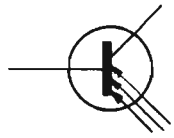


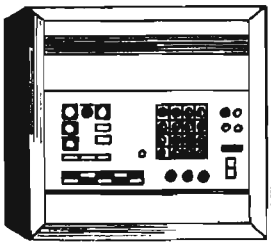


1 + 1 = 10
10 + 10 = 100
1000 - 100 = 100
11 × 11 = 1001

ET

OU





INITIATION AU CALCUL ELECTRONIQUE

BASIC • ALGOL • FORTRAN

(Suite : voir N° 1 338)

LES premiers projets de machines à calculer automatiques remontent à une époque où l'on ne connaissait guère que la mécanique : Charles Babbage semble en effet avoir été le premier, vers 1830, à concevoir, en Angleterre, la structure d'une machine à calculer automatique ; cette dernière était capable d'exécuter sans intervention humaine, une séquence importante d'opérations arithmétiques. Mieux encore, la machine, baptisée « Analytical Engine » devait pouvoir effectuer tous les calculs sans transformation importante.

La machine, pour être d'usage pratique, doit pouvoir passer rapidement d'un calcul à l'autre, sans immobilisation matérielle : elle suit donc un « programme », ensemble d'opérations à exécuter en séquences. Dans les projets de Babbage, le programme était matérialisé par des trous codifiés sur une bande de carton perforée.

Les principes découverts par Babbage sont encore utilisés, sans modification, par les machines les plus récentes.

Les systèmes de programmation, dits « automatiques » offrent un moyen de programmer des problèmes complexes, rapidement et aisément. Leur principe de base est l'utilisation de programmes universels, dont le but est très particulier : les données du calcul sont écrites avec une présentation conventionnelle.

Le langage Algol, dont on a pu lire les fondements dans les deux précédents numéros du *Haut-Parleur*, est l'un de ces langages universels.

L'INSTRUCTION « POUR »

Considérons le problème suivant ; soit la fonction :

$$F(x) = \frac{x^4 - 1}{x + 53}$$

On veut calculer la valeur de la fonction précédente pour trois valeurs de x, soit par exemple, pour $x = 2$, pour $x = 5$ et pour $x = 21$.

Ayant ainsi déterminé F(2), F(5) et F(21), on désire trouver la somme des trois valeurs précédentes :

S = F(2) + F(5) + F(21)

En Algol, une instruction, appelée instruction « POUR » permet d'écrire simplement ce problème :

```
DEBUT
REEL X, S ;
S := 0 ;
POUR X := 2, 5, 21 FAIRE
S := S + (X ↑ 2 − 1)/(X + 5) ;
FIN
```

(rappelons que le signe ↑ représente l'instruction de calcul d'une puissance : X ↑ 2 = X . X)

Plus généralement l'instruction « POUR » s'écrira en Algol :

POUR (variable) : = (liste d'expressions arithmétiques) FAIRE une instruction ;

Il arrive souvent que la variable prenne des valeurs en progression arithmétique. Par exemple, on donnera à la variable X toutes les valeurs entières comprises entre 0 et 10. L'instruction « POUR » s'écrirait alors :

POUR X := 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 FAIRE (J)

où (J) est une instruction.

Il existe une autre forme d'écriture plus souple dans ce cas :

POUR X := 0 PAS 1
JUSQUA 10 FAIRE (J)

On donne à la machine une valeur de départ (ici : 0) pour laquelle elle exécutera l'instruction (J). Puis la machine augmentera cette valeur de départ d'un incrément indiqué par : PAS (ici l'incrément vaut 1) et réexécutera l'instruction (J). Ensuite la machine augmentera de nouveau la dernière valeur trouvée de la variable d'un incrément égal au pas [et exécutera l'instruction (J)] jusqu'au moment où la variable atteint la valeur finale indiquée par : JUSQUA (ici la valeur finale est : 10). La machine exécute une dernière fois l'instruction (J) pour cette valeur finale de la variable, puis passe à l'instruction qui suit en séquence.

Il faut signaler que la valeur du pas et la valeur finale peuvent être calculées par une expression arithmétique.

Voici un exemple d'application.

Le « factoriel » d'un nombre entier N est défini comme étant le produit de tous les nombres entiers inférieurs ou égaux à N. Ce produit est noté : N !. Ainsi :

1 ! = 1
2 ! = 1 × 2
3 ! = 1 × 2 × 3
4 ! = 1 × 2 × 3 × 4
5 ! = 1 × 2 × 3 × 4 × 5

On désire écrire un programme en Algol qui fasse le calcul des 10 premiers factoriels. On écrira simplement :

```
DEBUT
ENTIER N, FACTORIEL N,
FACTORIEL N : = 1
POUR N : = 1 PAS 1
JUSQUA 10 FAIRE
DEBUT FACTORIEL N : =
FACTORIEL N ✳ N
IMPRIMER
(FACTORIEL N) ;
FIN ;
FIN
```

(le signe ✳ représente ici le signe de multiplication).

On constate que derrière le FAIRE, on a le droit de donner une instruction composée (qui commence par DEBUT et se termine par FIN). On aurait le droit d'écrire une autre instruction « POUR ».

Du programme précédent, la machine donnera les dix premiers factoriels :

1 ! = 1
2 ! = 2
3 ! = 6
4 ! = 24
5 ! = 120
6 ! = 720
7 ! = 5040
8 ! = 40320
9 ! = 362880
10 ! = 3628800

UNE TROISIÈME FORME

Dans certains problèmes, il arrive que l'on ne connaisse pas jusqu'où la variable doit varier. Par exemple, le problème du calcul de la racine carrée d'un nombre A positif quelconque, que nous avons traité de plusieurs façons le mois dernier, va pouvoir s'écrire avec une troisième forme de l'instruction « POUR ».

```
DEBUT
REEL V, U ;
U := 10 ;
POUR V := (U + 101300/U)
/2 TANT QUE ABS (V − U)
⩾ 1 FAIRE U := V ;
IMPRIMER (U) ;
FIN
```

La machine calcule ici la racine carrée du nombre 101300, en partant de la valeur initiale 10. On arrête le calcul de la racine carrée lorsque la valeur absolue de la différence V − U est inférieure à 1 (donc on itère le calcul tant que cette valeur absolue reste supérieure ou égale à 1).

Plus généralement, cette troisième forme s'écrira :

POUR (variable) : = (expression arithmétique) TANT QUE (expression booléenne)

FAIRE (instruction composée),

UNE PROCÉDURE

Il arrive que le catalogue des fonctions proposées par le compilateur Algol ne suffise pas et que l'on désire introduire de nouvelles fonctions. Par exemple, dans un problème traitant de vecteurs, on désire connaître la longueur (ou module) d'un vecteur dont les composantes scalaires sont connues. Soient X et Y les composantes du vecteur V. Le module de ce vecteur sera par définition, égal à la racine carrée de la somme $X^2 + Y^2$.

On va déclarer cette fonction nouvelle comme suit :

```
DEBUT
REEL PROCEDURE
MODULE (X, Y) ;
REEL A, X, Y ;
MODULE : = RAC2
(X ↑ 2 + Y ↑ 2) ;
```

dans le cours du programme, on pourra écrire, par exemple :

A : = MODULE (5, 2)

Il faut signaler que l'Algol prévoit beaucoup de possibilités de créer des fonctions nouvelles.

SAVEZ-VOUS PARLER ALGOL ?

Vous allez prendre connaissance d'un petit problème fort simple, aussi simple que celui du mois passé dont vous avez trouvé la solution ce mois-ci dans le cours de l'article. N'hésitez pas à nous écrire en cas de difficulté !

Aujourd'hui M. Martin, du service informatique de la Société Computex, veut apprendre à raisonner à son ordinateur. Pour cela, il rédige un programme en Algol, qui, une fois assimilé par la machine, lui permettra de trouver un nombre, compris entre 1 et 100, auquel vous pensez fortement.

Vous pensez donc à un nombre compris entre 1 et 100.

La machine vous demande :

« Votre nombre est-il inférieur à 50 ? Tapez 1 pour oui, 0 pour non sur le clavier du télétype ».

Vous tapez donc 0 ou 1, puis la machine continue à vous interroger :

« Votre nombre, est-il inférieur à 75 ? » si vous avez tapé 0, ou « Votre nombre est-il supérieur à 25 ? » si vous avez tapé 1, puis de nouveau :

« Tapez 1 pour oui, 0 pour non, sur le clavier du télétype. »

Et ainsi de suite...

Après un certain nombre de questions similaires, qui pourraient s'accompagner de questions du type :

« Avez-vous pensé à un nombre pair ? »

« Tapez 1 pour oui, 0 pour non sur le clavier du télétype », la machine indique le nombre auquel vous avez pensé.

Etes-vous en mesure de réécrire le programme de M. Martin, du service informatique de la Société Computex ?

SYNTAXE ALGOL

DEBUT

déclarations
- REEL . . . . . ;
- ENTIER . . . . . ;
- BOOLEEN . . . ;

} la spécification porte sur des variables, des tableaux ou des procédures

TABLEAU (identificateur de tableau)
REEL PROCEDURE (fonction procédure) ;
AIGUILLAGE (identificateur d'aiguillage).

affectation (variable) : = (expression arithmétique) ;
Instruction inconditionnelle : DEBUT (instruction 1) ; (instruction 2) ; (instruction 3) ; . . . ; FIN ;

Ordres de branchement
- inconditionnel : ALLERA (étiquette) ;
- conditionnels : SI (expression booléenne) ALORS (instruction inconditionnelle) SINON (instruction) ;

Instruction « POUR » :
- POUR (variable) : = ϵ_1, ϵ_2, ϵ_3, ϵ_4
- POUR (variable) : = ϵ_1 PAS ϵ_2 JUSQUA ϵ_3
- POUR (variable) : = ϵ_1 TANT QUE (expression booléenne)

} FAIRE (instruction)

FIN

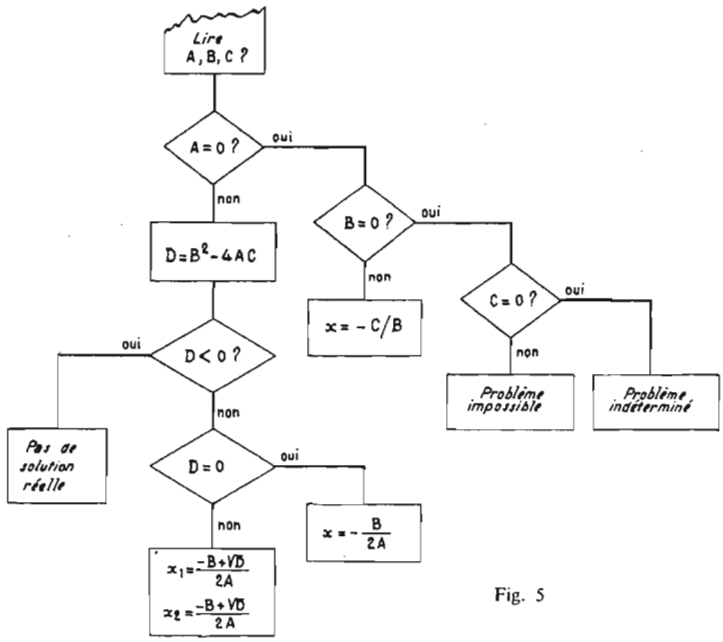


Fig. 5

Une procédure peut créer une fonction entière ou booléenne.

LES AIGUILLAGES

Nous allons faire un retour en arrière : l'instruction SI... ALORS ALLERA permet à la machine de prendre des décisions. Il arrivera qu'il y ait un choix à faire entre plusieurs solutions. Par exemple, dans le calcul des racines d'une équation du second degré, on doit calculer un nombre, appelé le « discriminant », puis voir si ce dernier est positif, nul ou négatif pour connaître l'existence ou l'absence d'une ou de deux racines à l'équation.

La solution classique reviendra donc à ce qui suit : soit $Ax^2 + Bx + C = 0$ l'équation du second degré pour laquelle on suppose que A n'est pas un nombre nul. Le discriminant est défini par : $D = B^2 - 4AC$.

Si D est négatif, il n'y a pas de racines à l'équation donnée.

Si D est nul, il y a une racine « double » :

$$x = -\frac{B}{2A}$$

Si D est positif, il y a deux racines x_1 et x_2 :

$$x_1 = \frac{-B + RAC2\ (D)}{2A}$$

$$x_2 = \frac{-B - RAC2\ (D)}{2A}$$

où RAC2 (D) est la racine carrée du discriminant D.

La déclaration d'aiguillage se place en tête du programme : AIGUILLAGE est suivi d'une expression de désignation (ici : CHOIX), du signe d'affectation et d'étiquettes (ici : ETI1, ETI2, ETI3).

On introduit un nombre N qui représentera le signe du discriminant : N = 1 s'il est négatif, 2 s'il est nul et 3 s'il est positif.

Dans le cours du programme, un ordre de branchement inconditionnel apparaît :

ALLERA CHOIX (N) ;

selon que N vaudra 1, 2 ou 3, la machine ira à l'étiquette ETI1, ETI2 ou ETI3.

EN RÉSUMÉ...

En résumé, l'Algol a pour syntaxe, principalement les phrases indiquées au tableau III.

Les langages Basic et Fortran ont de nombreux points de similitude avec l'Algol, ce qui nous permettra d'aller bien plus vite dans leur description...

LE LANGAGE BASIC

Le language Basic a été créé par le collège de Darmouth, aux U.S.A., pour les utilisateurs d'ordinateurs en « time-sharing », qu'ils n'aient aucune connaissance préalable des calculateurs, ou qu'ils aient déjà une expérience de la programmation. Ainsi, le Basic est aussi commode, pour l'homme d'affaires, que pour l'ingénieur ou le chercheur.

Pour son écriture, le Basic utilise les notations mathématiques standard. De plus, à cote de ses possibilités arithmetiques, il contient aussi des possibilités d'édition de textes, des procedures d'entrée et sortie, et un jeu complet de diagnostics et de messages d'erreurs.

TIME-SHARING ET BATCH-PROCESSING

Mis au point en 1965, le Basic (acronyme de « Beginner All-purpose Symbolic Instruction Code) est spécialement conçu pour l'emploi conversationnel et l'enseignement.

Plusieurs tendances caractérisent, à l'heure actuelle, le time-sharing : diversité des services, réseaux internationaux, manque de formation des utilisateurs.

Mais, qu'est-ce que le time-sharing ?

Que recouvre ce terme américain issu du jargon informatique ?

Le terme « time-sharing » désigne un mode d'exploitation d'un ordinateur. L'un des modes d'ailleurs, car ils sont nombreux : Remote Batch, Batch Processing, etc.

Comment travailler avec un ordinateur ?

La première méthode consiste à « enfourner » son programme dans la machine. On appuie sur le bouton « Marche » et l'on attend les résultats sur l'imprimante rapide. C'est le mode de travail en Batch. L'ordinateur avale les programmes et les traite les uns après les autres, quand un programme est terminé, on passe au suivant.

Le traitement en Batch (ou Batch-Processing) peut se faire à distance. C'est alors du Remote-Batch. Un terminal de télégestion relie l'utilisateur à l'ordinateur.

On peut encore concevoir une machine mettant tous les programmes dans sa mémoire, et qui traite une partie de chaque programme, qui place les résultats partiels dans la mémoire, et qui continue à traiter la suite de chacun des programmes, lorsqu'elle aura vu l'ensemble de chacun des programmes qu'on lui a donné à traiter. Ainsi, chaque programme est traité pendant quelques dizaines de millisecondes ; puis, c'est le programme suivant qui est traité pendant un incrément de temps identique, et ainsi de suite jusqu'au dernier programme ; la machine revient alors au premier programme et recommence sa « tournée » des programmes.

C'est du time-sharing. Le temps d'utilisation de l'ordinateur est partagé entre tous les utilisateurs. L'ensemble des « ressources » rassemblées autour d'un ordinateur est partagé : la puissance de calcul, les équipements de stockage d'informations, les langages de programmation, et même toute la matière grise enregistrée dans la « bibliothèque » sous forme de programmes élaborés, qui vont permettre au financier d'utiliser un modèle économique, ou à l'ingénieur de trouver les racines d'une équation complexe.

Le dialogue avec l'ordinateur doit rester très simple en time-sharing et ne plus être le fait de rares professionnels qui ont d'ailleurs suffisamment à faire avec l'écriture de nouveaux systèmes software. Il faut que celui qui a un problème puisse l'exprimer en un langage simple, accéder simplement à l'ordinateur et dialoguer avec lui.

Fig. 6. — *Le PICTUREPHONE permettra de communiquer plus vite en time sharing : la machine à écrire sera remplacée par un écran de télévision.*
(Cliché Bell Tel. Lab.)

BASIC

Tout d'abord, il faut savoir qu'en Basic :

● Contrairement à l'Algol, on commence le programme directement. Il n'y a pas de mot DEBUT en tête de programme ;

● Les seules variables utilisables sont : les lettres de l'alphabet et les lettres suivies d'un chiffre. Ainsi, alors qu'en Algol, les variables PRESSION, TEMPERATURE, INTENSITE étaient permises, en Basic, on devra se contenter des variables P. T. I.... et aussi P1, P2.... P9, T1,... T9... ;

● L'affectation se fait de la façon suivante :

LET X = X + 1

La variable X, si elle valait 5,5 avant de parvenir à cette instruction, prendra la valeur 6,5 après cette instruction.

Certains compilateurs bien réalisés permettent de s'affranchir du mot LET. On écrira alors simplement l'instruction d'affectation sous la forme :

X = X + 1

A gauche du signe « d'affectation » (qui est ici le signe « égal ») se trouvera toujours une variable ; à droite, on trouvera une expression arithmétique... comme en Algol.

Par exemple, les « phrases » ci-dessous sont correctes :

A = 5
B = 10
C = 15
D1 = B↑2 − 4 ✳ A ✳ C

On élève à la puissance par le signe ↑. La multiplication se fait par l'astérisque ✳.

● Enfin le point-virgule qui sépare en Algol, chaque phrase du programme, n'existe plus en Basic. Sur la machine à écrire, on fera simplement un « retour-chariot » après chaque phrase : on va, en d'autres termes, à la ligne après chaque instruction du programme.

On notera que les blancs sont ignorés. Par exemple, on pourra écrire :

X I = (− B + D)/2/A
ou X I = (− B + D)/ 2/A

Les blancs entre chaque variable ne sont pas pris en compte.

EXPRESSIONS ARITHMÉTIQUES EN BASIC

Le Basic accepte tout nombre décimal, positif ou négatif avec ou sans partie décimale ; la partie décimale, si elle existe, s'exprime au plus, avec 8 chiffres significatifs. En Basic, comme en Algol et en Fortran, la virgule séparant les parties entière et décimale d'un nombre est remplacée par un point : c'est une convention anglo-saxonne, reprise dans le monde de l'informatique. Ainsi, on n'écrira pas − 152,55, mais :

− 152.55

En Basic, un nombre pourra s'exprimer également sous la forme d'une mantisse (ce sera un nombre, entier ou réel), suivi d'un exposant (avec puissance entière de 10). Cette forme d'écriture convient pour les nombres trop grands ou trop petits pour être représentés par 8 chiffres significatifs. Ainsi, pour écrire 1 picofarad, on dira que :

$1\ pF = 10^{-12}\ F$

Ce qui s'écrira, en Basic I E − 12. La lettre E remplace symboliquement la valeur 10.

Autre exemple : − 152.55 pourra s'écrire :

− 0.15255E+3,
ou − 15255E3,
ou − 15255E−2,
ou encore : − 1.5255 E2

Le Basic autorise, comme les autres langages universels, l'utilisation d'un symbole pour représenter un nombre. Ce symbole est une variable simple (lettre, ou lettre suivie d'un chiffre) ou une variable indicée : on constitue un tableau, et chaque valeur du tableau est repérée par un indice. Par exemple A(4) fait référence au quatrième élément du tableau A. Ici, 4 est appelé l'indice.

Un tableau peut être à une dimension (c'est un « vecteur ») ou à deux dimensions (c'est alors une « matrice »). Dans ce dernier cas, un nombre indicé quelconque, d'une matrice s'écrira A(X,Y) : X représente le « numéro d'une ligne de la matrice » ; Y, le « numéro d'une colonne de la matrice ».

Par exemple, le tableau LV représente, sous forme de matrice, le détail de dépenses faites entre le 5 juin et le 8 juin. Appelons D ce tableau à deux dimensions. L'élément D(4,1) désigne la dépense inscrite au croisement de la **quatrième** ligne et de la **première** colonne. C'est donc une dépense de logement fait le 5 juin. Ici, donc, D(4,1) = 10.05.

Il est bien évident qu'il n'y a pas commutativité des indices : D(1,4) représente les dépenses en essence du 8 juin et D(1,4) = 6.08

Attention, un tableau se déclare en Basic :

— s'il s'agit d'une matrice ayant plus de 11 colonnes ou 11 lignes ;

— s'il s'agit d'un vecteur à plus de 11 composantes ;

par l'instruction DIM, placée avant la première instruction faisant appel au tableau.

La syntaxe de cette instruction de déclaration sera pour un vecteur :

DIM Q(58)

et l'ordinateur automatiquement réservera à l'utilisateur 59 positions de la mémoire, qui seront appelées Q(0), Q(1), Q(2), ..., Q(58). L'expression entre parenthèses doit être un nombre positif ou nul : c'est éventuellement une expression algébrique (mais tous les éléments et variables de cette expression doivent avoir une valeur).

Pour une matrice, la déclaration d'un tableau R, s'écrira :

DIM R(30,5)

L'ordinateur réservera un casier, dans la mémoire, contenant 31 lignes, numérotées 0 à 30, et 6 colonnes, numérotées 0 à 5.

On constate ainsi que :

● le nom d'un tableau est donné par une lettre de l'alphabet,

● la dimension du tableau T(I, J) est : (I + 1) . (J + 1).

Venons-en aux expressions arithmétiques : Elles sont formées de combinaisons de variables et de constantes, séparées par des opérateurs arithmétiques, comme

dans les formules mathématiques usuelles :

exponentiation ↑
multiplication ✳
division /
addition +
soustraction –

Les parenthèses sont, de plus, autorisées.

Ainsi, l'expression X/Y + 6 signifie que l'on divise X par Y, puis qu'on ajoute 6 au résultat de la division. Mais X/(Y + 6) est la division de X par la somme Y + 6.

Les fonctions mathématiques standard, disponibles en Basic, sont :

SIN (expression) : sinus de l'expression en radians.

COS (expression) : cosinus de l'expression en radians.

TAN (expression) : tangente de l'expression en radians.

ATN (expression) : arctangente d'une expression (résultat donné en radians).

EXP (expression) : exponentiation d'une expression.

ABS (expression) : valeur absolue d'une expression.

LOG (expression) : logarithme népérien d'une expression.

SQR (expression) : racine carrée d'une expression.

LGT (expression) : logarithme à base 10 d'une expression.

INT (expression) : partie entière d'une expression.

RND : nombre au hasard.

Lorsque l'on écrit Y = RND, la machine donne à Y une valeur absolument quelconque prise entre 0 et 1.

Enfin, les relations disponibles en Basic sont les suivantes :

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

La relation sera vraie ou fausse selon que la réponse donnée à la question posée sera OUI ou NON : en effet X > 5 signifie « la variable X a-t-elle une valeur supérieure à 5 ? ».

De même, on pourra rencontrer la relation Y = X, qui signifie ici : « la valeur de la variable Y est-elle égale à celle de la variable X ? Oui ou non ? » Il est important de bien comprendre la différence entre la relation « égal » et l'instruction « d'affectation ». La relation X = X + 1 sera toujours fausse. Mais l'instruction X = X + 1 affectera à X un point de plus à sa valeur précédente !

LA PHRASE BASIC

Une phrase Basic aura dans tous les cas, la syntaxe suivante :

– elle débute par un numéro de ligne ;

– le numéro de ligne est suivi d'une instruction ou d'une déclaration.

La présence d'un numéro de ligne entraîne qu'une instruction n'est pas exécutée immédiatement ; elle sera stockée en mémoire. Lorsque l'on frappera sur le clavier du télétype, l'instruction RUN, les lignes seront compilées, traduites en langage machine et exécutées dans l'ordre des numéros de ligne croissants.

Pour changer une ligne, il suffit de retaper, n'importe où dans le programme, la ligne correspondante. La machine prendra en compte exclusivement la dernière version de la ligne. Par exemple, supposez que pour le calcul du discriminant d'une équation du second degré, on ait écrit la phrase suivante :

```
100  D = B↑2 – 4 ✳ A ✳ B
```

On veut modifier cette phrase ; il suffira de retaper, après cette instruction, la phrase correcte :

```
100  D = B↑2 – 4 ✳ A ✳ C
```

Cette dernière instruction effacera l'instruction contenue précédemment à la ligne 100.

IMPRIMER ET LIRE

En Basic, l'impression des résultats se fait par la commande PRINT. La syntaxe de la phrase d'impression sera la suivante :

```
950  PRINT A, B, C,
       B↑2 – 4 ✳ A ✳ C
```

La commande PRINT est :

– précédée d'un numéro de ligne,

– suivie d'expressions arithmétiques, dont on veut imprimer la valeur,

– suivie éventuellement d'un texte en clair :

```
950  PRINT
       " DISCRIMINANT = ",
       B↑2 – 4 ✳ A ✳ C
```

Donnons un exemple de programme : soit à calculer l'hypoténuse C d'un triangle rectangle de côtés A, B, C ; si A et B ont les valeurs respectives 3 et 4, on écrira :

```
100  A = 3
110  B = 4
120  PRINT
       " HYPOTENUSE : "
       SQR(A↑2 + B↑2)
130  END
```

Si l'on ne met pas de virgule entre le texte et l'expression à calculer, le résultat du calcul apparaîtra accolé au texte :

HYPOTENUSE : 5

Si l'on désire imprimer les valeurs des 3 côtés du triangle rectangle, on n'a qu'à frapper sur le clavier du terminal :

```
115  C = SQR(A↑2 + B↑2)
120  PRINT " A =" A,
       " B = " B,
       " C = " C
```

La ligne 115 s'intercalera, en mémoire, entre les lignes 110 et 120, et la nouvelle ligne 120 remplacera l'ancienne.

A l'ordre RUN, tapé sur le terminal, la machine commencera le calcul et imprimera le texte suivant :

A = 3 B = 4 C = 5

On constate que le programme se termine par le mot END.

Pour donner des valeurs à des variables d'un programme, on peut, grâce à une instruction INPUT, procéder à des affectations au moment de l'exécution du programme.

Au moment de l'exécution, chaque fois qu'une instruction INPUT est rencontrée, le programme est interrompu : l'ordinateur attend que l'opérateur frappe les données correspondant aux variables de la liste suivant INPUT. Lorsque toutes les données ont été frappées au clavier et que l'opérateur a exécuté un « retour chariot » sur le clavier, l'exécution est reprise. La syntaxe de cette instruction est :

```
10  INPUT A, B, C, D, E, Z1
```

Quand la commande RUN est donnée, l'exécution s'interrompt au niveau de la ligne 10. L'opérateur doit alors inscrire 6 valeurs sur le clavier et procédera à un « retour chariot ».

Pour lire une matrice, on utilisera l'instruction MAT INPUT.

```
10  DIM A(20), B(15, 50)
50  MAT INPUT A, B
```

A signaler qu'il est possible d'imprimer tout un tableau (ligne par ligne, s'il s'agit d'une matrice) grâce à la seule instruction MAT PRINT. La syntaxe sera la suivante :

```
1000  MAT PRINT A
```

READ-DATA-RESTORE

Tout ce qui précède est classique. Une innovation importante est apportée par le langage Basic en matière de lecture de données grâce au « Bloc DATA ».

Dans certains problèmes, l'utilisateur a besoin de données qui seront bien spécifiques du programme : par exemple, on voudra insérer dans le programme les paramètres fondamentaux de transistors. Ce sont des valeurs immuables et il serait fastidieux d'écrire toutes ces valeurs dans le corps du programme, ou de demander à l'utilisateur du programme de les taper au clavier chaque fois que le programme passe en machine.

Le Basic a donc prévu l'établissement d'une sorte de bibliothèque de constantes numériques ; ces dernières sont rangés dans un « Bloc DATA », suivant la syntaxe suivante :

```
3000  DATA 5, 10, 15, 20
3010  DATA 100, 0, 4 E–2, 5.3
3020  DATA 1.1, 1.7, 34902,
        33.367 E 6
```

Chaque instruction DATA est précédée d'un numéro de ligne et suivie d'une suite de constantes séparées par des virgules.

Les constantes rangées dans le bloc DATA seront lues au moyen d'une instruction READ ou MAT READ, s'il s'agit d'une matrice.

```
 10  DIM A(2)
 40  READ A
 41  READ A1, A2, A3
 42  MAT READ A
 .
 .
 .
100  DATA 1, 2, 3, 4
101  DATA 0.01, 1, 10
```

Cette partie du programme est équivalente à :

```
10  DIM A(2)
40  A = 1
41  A1 = 2
42  A2 = 3
43  A3 = 4
44  A(0) = 0.01
45  A(1) = 1
46  A(2) = 10
```

Enfin, l'instruction RESTORE permet de reprendre la lecture de données au début du bloc DATA :

```
55  RESTORE
```

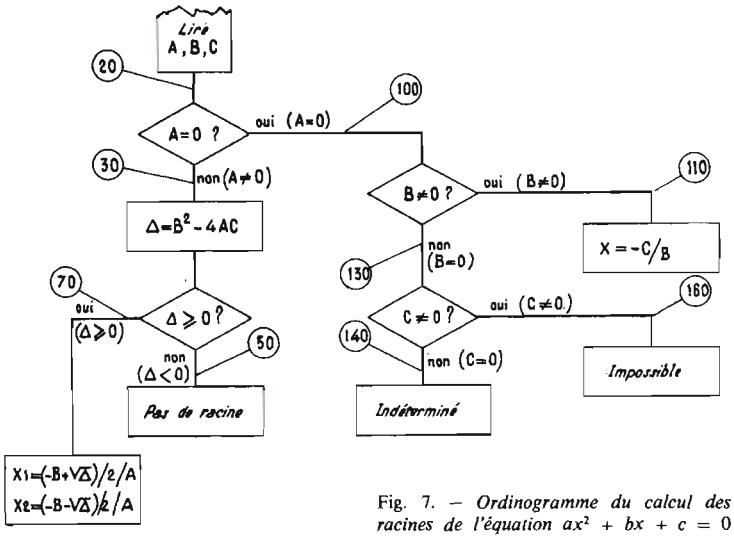


Fig. 7. – *Ordinogramme du calcul des racines de l'équation* $ax^2 + bx + c = 0$

Voici un exemple de programme où l'on rencontre les instructions DATA-READ-RESTORE :

```
10  READ A, B, C, D
15  RESTORE
20  READ E, F, G
30  PRINT " A = " A,
    " B = " B, " C = " C,
    " D = " D
40  PRINT " E = " E,
    " F = " F, " G = " G
50  DATA 1, 3, 5
60  DATA 7, 9, 11, 13
70  END
```

Le résultat apparaîtra ainsi lorsque l'opérateur aura tapé la commande RUN :

A = 1 B = 3 C = 5 D = 7
E = 1 F = 3 G = 5

Si l'on supprime la ligne n° 15 et si l'on tape RUN, la machine imprimera alors :

A = 1 B = 3 C = 5 D = 7
E = 9 F = 11 G = 13

RUPTURES DE SÉQUENCE

Les instructions d'un programme Basic sont exécutées dans l'ordre des numéros croissants de ligne. On peut cependant rompre cette séquence par un ordre de rupture inconditionnel de séquence GOTO.

```
150  GOTO 311
```

L'ordre GOTO est précédé du numéro de sa ligne. Il est suivi du numéro de la ligne où le calcul doit se poursuivre. Dans le cas présent, l'ordre est donné de sauter de la ligne 150 à la ligne 311.

Il est souvent utile de provoquer une rupture de séquence si une condition est réalisée : c'est l'ordre de rupture conditionnel IF... THEN..., dont la syntaxe commence toujours par un numéro de ligne ; si la relation placée entre IF et THEN est vraie, le calcul se poursuit à la ligne dont le numéro suit THEN. Si la relation n'est pas vraie, on passe à la ligne suivant cet ordre.

A titre d'exemple, examinons le calcul des racines d'une équation du second degré, dont l'ordinogramme est donné à la figure 7 :

```
 0  REM RACINES DE
    A X↑2 + BX + C = 0
10  INPUT A, B, C
20  IF A = 0 THEN 100
30  D = B↑2 − 4 ✳ A ✳ C
40  IF D >= 0 THEN 70
50  PRINT " PAS DE
    RACINE "
60  STOP
70  X1 =
    (− B + SQR(D) )/2/A
80  X2 =
    (− B − SQR(D) )/2/A
90  PRINT " X1 = " X1,
    " X2 = " X2
95  STOP
100 IF B = 0 THEN 130
110 PRINT " UNE SEULE
    RACINE : " − C/B
120 STOP
130 IF C < > 0 THEN 160
```

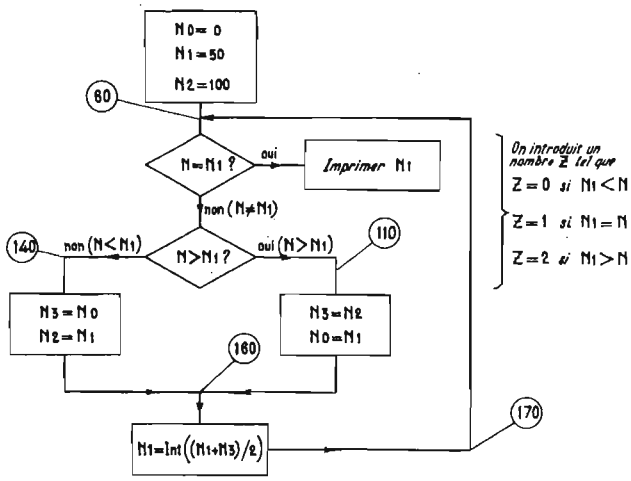


Fig. 8. — *Recherche d'un nombre compris entre 0 et 100.*

```
140 PRINT " PROBLEME
    INDETERMINE "
150 GOTO 170
160 PRINT " PROBLEME
    IMPOSSIBLE "
170 END
```

On constate la présence dans ce programme :

— d'ordres de rupture de séquence conditionnels aux lignes 20, 40, 100 et 130,

— d'un ordre inconditionnel de rupture de séquence à la ligne 150,

— d'un ordre d'arrêt inconditionnel STOP du programme aux lignes 60, 95, 120.

A la ligne 0, la présence du mot REM (abrégé de REMARQUE) permet à l'utilisateur de donner un titre à son programme. La machine ne s'occupera jamais de ce qui est inscrit derrière REM.

BOUCLES

Comment écrire en Basic un programme de calcul du factoriel des 100 premiers nombres ? Rappelons que le factoriel d'un nombre N s'écrit, par convention N ! et a pour valeur le produit de N par tous les nombres entiers inférieurs à N :

$N ! = 1 \times 2 \times 3 \times 4 \times 5 \times .. \times (N - 1) \times N$

On voit que $N ! = [1 \times 2 \times 3 \times ... \times (N - 1)] \times N = [(N - 1) !] \times N$

Le programme suivant effectue ce calcul ainsi :

```
10  REM FACTORIEL DES
    100 PREMIERS NOMBRES
    ENTIERS
20  N = 1
30  F = 1
40  N = N + 1
50  IF N > 100 THEN 90
60  F = F ✳ N
70  PRINT " N = " N,
    " N ! = " F
80  GOTO 40
90  END
```

Ici F représente le factoriel de N. L'ordinateur commence l'exécution du programme par N = 1. A la ligne 50, comme N est inférieur à 100, il calcule le factoriel de 2, puis, ligne 70, imprime 2 et 2 ! = 2. A la ligne 80, il y a rupture de sequence et retour à la ligne 40. N prend la valeur 3, la machine calcule 3 ! = 2 x 3 = 6, puis revient à la ligne 40 ; jusqu'à N = 100. La machine aura calculé la valeur de 100 !, puis on retourne a la ligne 40, ou N devient égal à 101. Mais à la ligne 50, il y a rupture de sequence, l'ordinateur va en ligne 90, fin du programme.

L'instruction FOR... THEN... permet de commander un nombre donné d'itérations d'une partie du programme de façon plus élégante. La syntaxe est la suivante :

```
 50  FOR N = 0 TO 100
     STEP 1
.
.
.
100  NEXT N
```

Les valeurs de départ de l'itération (ici 0), de fin d'itération (ici 100) et le pas de la variation de N (ici 1) peuvent être des expressions arithmétiques.

Une boucle itérative se termine toujours par NEXT...

Lorsque le pas vaut 1, on peut supprimer STEP 1.

Ainsi, le programme de calcul des factoriels peut s'écrire maintenant :

```
10  REM FACTORIEL N
20  F = 1
30  FOR N = 2 TO 100
40  F = F ✳ N
50  PRINT " N = " N,
    " N ! = " F
60  NEXT N
70  END
```

L'écriture est simplifiée et moins longue.

GOSUB/RETURN

Lorsqu'une partie du programme Basic fait appel à une fonction nouvelle ne faisant pas partie de la liste des fonctions standards, on fera une déclaration de fonction : DEF...

Le nom de la nouvelle fonction est formé de 3 lettres dont les 2 premières sont obligatoirement FN. La troisième lettre est laissée au choix du programmeur. Le nom de la fonction est suivi, entre parenthèses, du paramètre formel de la fonction. Enfin, on écrit sur une seule ligne l'expression de définition de la nouvelle fonction. Exemples :

```
25  DEF FNF (Z) =
    Z ✳ PI/180
40  DEF FNL (X) =
    LOG(X)/LOG(10)
60  DEF FNX (X) =
    SQR(X↑2 + Y↑2)
```

On notera que certains compilateurs Basic donnent la valeur de Pi (3.14159265).

Dans le cours du programme, si ces trois fonctions sont définies, on pourra écrire, par exemple :

```
150  Y = 30
160  S1 = FNX(40)
```

pour calculer la racine carrée de la somme $40^2 + 30^2$.

Pour utiliser une fonction déclarée dans une expression, il suffit donc d'écrire le nom de la fonction, suivi du paramètre effectif d'utilisation.

Souvent, on a affaire à des fonctions plus complexes ou a des parties de programmes qui ne sont pas des fonctions, mais qui seront utilisées à plusieurs endroits d'un même programme. On emploie alors un sous-programme.

L'appel à un sous-programme se fait par un GOSUB :

```
30  GOSUB 50
.
.
.
50  REM SOUS PROGRAMME
    N° 1
.
.
.
70  RETURN
```

L'effet d'un GOSUB est identique à celui d'un GOTO, mais de plus, intervient le fait que le Basic note le numéro de ligne du GOSUB qui provoque la rupture de séquence. Au premier RETURN rencontré, on revient à la ligne qui suit immédiatement ce GOSUB.

EXEMPLE

Par exemple, le programme ci-dessous résout le jeu suivant : « recherche par l'ordinateur d'un nombre compris entre 1 et 100 auquel vous pensez ».

La figure 8 en donne l'ordinogramme.

Vous pensez à un nombre N compris entre 0 et 100.

La machine prend trois nombres :

— un nombre N_0 qui sera toujours inférieur à N,

— un nombre N_2 qui sera toujours supérieur à N,

— un nombre N_1 égal à la partie entière de $(N_0 + N_2)/2$.

La machine vous demandera de comparer N_1 au nombre auquel vous avez pensé.

Si le nombre N_1 est inférieur au nombre auquel vous avez pensé, on donnera pour la suite du calcul cette valeur N_1 au minorant N_0.

Si N_1 est supérieur au nombre auquel vous avez pensé, on donnera cette valeur N_1 au majorant N_2.

Ainsi, à chaque itération, les bornes supérieures (N_2) et inférieure (N_0) se rapprochent du nombre auquel vous avez pensé. Il arrive un moment où les bornes inférieure et supérieure encadrent votre nombre de telle sorte que la valeur du nouveau nombre de comparaison N, est juste égale à celui auquel vous pensez. Le jeu est alors terminé. Voici donc le programme Basic de ce jeu :

```
  0  REM JEU DE
     RECHERCHE D'UN
     NOMBRE
 10  PRINT " PENSEZ A
     UN NOMBRE
     COMPRIS ENTRE 0
     ET 100"
 20  N0 = 0
 30  N1 = 50
 40  N2 = 100
 50  PRINT " VOTRE
     NOMBRE EST-IL : "
 60  PRINT " PLUS PETIT,
     EGAL OU PLUS GRAND
     QUE : " N1
 70  PRINT " TAPEZ
     0 POUR PLUS PETIT,
     1 POUR EGAL,
     2 POUR PLUS GRAND "
 80  INPUT Z
 90  IF Z = 1 THEN 180
100  IF Z = 0 THEN 140
110  N3 = N2
120  N0 = N1
130  GOTO 160
140  N3 = N0
150  N2 = N1
160  N1 = INT[(N1 + N3)/2]
170  GOTO 60
180  PRINT " VOUS AVEZ
     PENSÉ A " N1
190  END
```

MATRICES

Le Basic permet encore de nombreuses autres possibilités que celles indiquées. En particulier, le traitement de problèmes matriciels se fait très rapidement en Basic, beaucoup plus rapidement qu'en Algol, ou qu'en Fortran. On peut additionner, soustraire, multiplier des matrices, les transposer, ou les inverser à l'aide d'instructions très rudimentaires :

```
  5  INPUT I, J, K, L
 10  DIM  X(I,  J),  Y(J,  J),
     Z(J, K)
 20  DIM A(I, J)
 30  DIM S(I, J)
 40  DIM P(I, K)
 50  DIM M(I, J)
 60  MAT INPUT X, Y, Z
100  MAT A = X + Y
110  MAT S = X - Y
120  MAT P = Y * Z
130  MAT M = (L) * X
140  MAT PRINT A, S, P, M
150  END
```

Ici, la matrice A est la matrice addition des deux matrices X et Y ; S est la matrice soustraction ; P est le produit matriciel des deux matrices Y et Z ; enfin, M est la matrice produit de l'expression L par la matrice X.

Si on veut les transposer et les inverser, on écrira :

pour la transposée de la matrice A :

```
141  DIM T(J, I)
142  MAT T = TRN(A)
```

et pour l'inverse d'une matrice X(I, I) :

```
143  DIM MAT X(I, I)
     Y(I, I)
144  MAT INPUT X
145  MAT Y = INV(X)
```

Les lecteurs ayant l'expérience du calcul matriciel verront, dans ce bref exposé des possibilités matricielles du Basic, toutes les possibilités qui leur sont maintenant ouvertes : en particulier, la résolution d'une équation à 50 inconnues se fera très aisément en Basic, en jouant sur l'ensemble des opérations matricielles qui sont prévues.

Le mois prochain, il sera question du Fortran.

(A suivre.)

Marc FERRETTI.

SAVEZ-VOUS PARLER BASIC

Vous allez prendre connaissance d'un petit problème fort simple. Il n'est absolument pas nécessaire d'avoir un ordinateur pour le résoudre. Cet exercice est le troisième de la série, les deux premiers étant à rédiger en Algol.

Bonne initiation ! et n'hésitez pas à nous écrire en cas de difficulté !

Aujourd'hui, M. Martin, du service informatique de la Société Computex, doit déterminer le jour de votre naissance. Vous lui indiquez votre date de naissance, l'ordinateur vous répond : « C'était un lundi », ou « C'était un samedi »...

Comment, à votre avis, est écrit le programme Basic résolvant ce calcul ?