EXEMPLE D'EMPLOI DE OPEN ET CLOSE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COPIE.
REMARKS. CE PROGRAMME EST UN EXEMPLE D'EMPLOI
         DES INSTRUCTIONS OPEN ET CLOSE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT F1-BANDE   ASSIGN  TO TAPE F1.
    SELECT F2-BANDE   ASSIGN  TO TAPE F2.
    SELECT F-DISQUE   ASSIGN  TO DISC F-DISQUE.
DATA DIVISION.
FILE SECTION.
FD  F1-BANDE
    LABEL RECORD STANDARD.
01  REC-F1-BANDE.
    05  ..............
    05  ..............
FD  F2-BANDE
    LABEL RECORD STANDARD.
01  REC-F2-BANDE.
    05  ..............
    05  ..............
FD  F-DISQUE
    LABEL RECORD STANDARD.
01  REC-F-DISQUE.
    05  ..............
    05  ..............
*
 WORKING-STORAGE SECTION.
 01 REC-DISPONIBLE-F1          PIC X(...).
*
PROCEDURE DIVISION.
PREMIERE SECTION.
DEBUT.
    DISPLAY '** DEBUT COPIE **' UPON PRINTER.
    OPEN INPUT F1-BANDE REVERSED.
    PERFORM LIRE-DERNIER-REC-F1.
    CLOSE F1-BANDE.
    OPEN INPUT F1-BANDE.
    OPEN OUTPUT F-DISQUE.
    PERFORM LIRE-F1-ECRIRE-F.
    CLOSE F1-BANDE WITH NO REWIND.
    CLOSE F-DISQUE.
    OPEN  F2-BANDE WITH NO REWIND.
    OPEN INPUT F-DISQUE.
    PERFORM COPIE-DERNIER-REC-F1.
    PERFORM LIRE-F-ECRIRE-F2.
    CLOSE F-DISQUE.
    CLOSE F2-BANDE.
    DISPLAY '** COPIE TERMINEE **'
            UPON PRINTER.
    STOP RUN.
*
*  = = = = = = =    SECTIONS   = = = = = = = = = =
*
 LIRE-DERNIER-REC-F1 SECTION.
 DERNIER-REC.
        ..............
 DERNIER-REC-EX. EXIT.
*
 LIRE-F1-ECRIRE-F SECTION.
 F1-F.
        ..............
 F1-F-EX. EXIT.
*
 COPIE-DERNIER-REC-F1 SECTION.
 COPIE-DERNIER.
        ..............
COPIE-DERNIER-EX. EXIT.
*
 LIRE-F1-ECRIRE-F2 SECTION.
 F1-F2.
        ..............
 F1-F2-EX. EXIT.
```

dernier enregistrement logique et est prêt pour l'écriture d'autres enregistrement à partir de cette position. Le format est :

```
OPEN EXTEND nom-du-fichier
```

Notons l'importance de cette instruction pour effectuer des ajouts de données sur des fichiers déjà existants. Au lieu d'écrire :

```
OPEN E/S FICHIER-A.
PERFORM LIRE-TOUT-FICHIER-A.
PERFORM
ECRIRE-NOUVEAUX-RECORD.
CLOSE FICHIER-A.
```

Avec la clause EXTEND, on a recours à la séquence suivante qui effectue les mêmes opérations :

```
OPEN EXTEND FICHIER-A.
PERFORM
ECRIRE-NOUVEAUX-RECORD.
CLOSE FICHIER-A.
```

L'avantage de cette procédure est surtout évident quand le nombre d'enregistrements contenus dans les fichiers est grand. Cependant, la clause EXTEND n'est pas utilisable pour un fichier vide : le fichier ouvert avec la clause EXTEND contient au moins une donnée. Les formats complets des instructions OPEN et CLOSE sont donnés ci-dessous. En utilisant l'extension LOCK avec CLOSE, on ferme un fichier de manière à empêcher sa réouverture au cours du programme. Pour une plus grande lisibilité du programme, il est conseillé d'écrire les instructions en les disposant en colonne :

```
OPEN INPUT      FICHIER-1
                FICHIER-2
                FICHIER-3
       OUTPUT   FICHIER-4
                FICHIER-5

CLOSE  FICHIER-1
       FICHIER-2 WITH LOCK
                 WITH LOCK
       FICHIER-5
```

Lecture et écriture sur les fichiers : les verbes READ et WRITE

Dans les exemples donnés jusqu'à présent, une opération de lecture sur fichier s'effectue avec READ, l'écriture d'un enregistrement sur un fichier avec WRITE. Le format le plus simple de ces instructions est :

```
READ nom-du-fichier AT END
                        phrase-impérative.
WRITE nom de l'enregistrement.
```

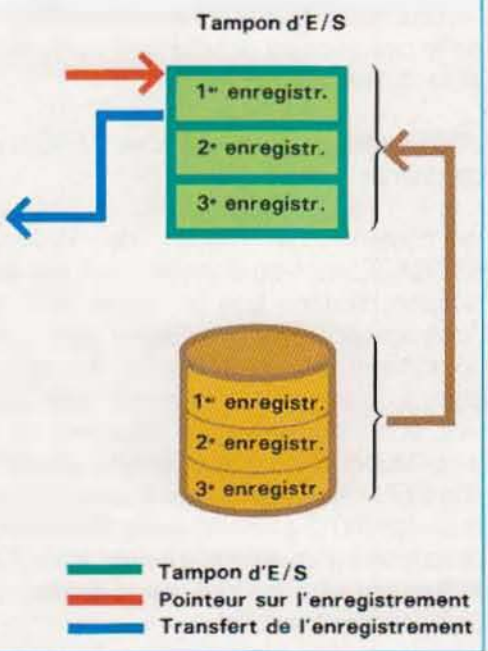
(Pour mémoire, on lit (READ) un fichier et on écrit (WRITE) un enregistrement).
Chaque instruction de lecture sur un fichier opère le transfert d'un enregistrement logique. En fichiers séquentiels, on effectue autant de lectures que d'enregistrements contenus dans le fichier, entre l'ouverture et la fermeture du fichier. Après chaque commande de lecture, un test de fin de fichier est automatiquement réalisé. Lorsque le système d'exploitation reconnait la fin du fichier, la phrase décrite dans la clause

FORMATS COMPLETS DES INSTRUTIONS OPEN ET CLOSE

```
{ INPUT   { nom-du-fichier } [ { REVERSED          } ] }
{ OUTPUT  nom-du-fichier     [ { WITH NO REWIND    } ] }
{ E-S     { nom-du-fichier                           } }
{ EXTEND  nom-du-fichier                             } }

CLOSE  nom-du-fichier  [ WITH { NO REWIND } ]
                              { LOCK      }
```

MECANISME DE LECTURE D'UN FICHIER

```
FD FICHIER-A
    LABEL ENREGISTREMENT STANDARD
    BLOCK CONTAINS 3 RECORDS
    ENREGISTREMENT CONTAINS 100 CHARACTERS
    01 REC-A                          PIC x (100)
    WORKING-STORAGE SECTION
    01 REC-A-W-S                      PI x (100)
    PROCEDURE DIVISION
    MAIN SECTION
    OUVRIR FICHIER
        OPEN INPUT FICHIER-A
        READ FICHIER-A INTO REC-A-W-S
            AT END
            DISPLAY 'FIN FICHIER-A'
                UPON PRINTER
            CLOSE FICHIER-A
```



AT END est exécutée. L'instruction READ est également utilisée avec le format :

READ nom-du-fichier INTO nom-de-donnée AT END phrase-impérative.

Dans le premier format, le contenu de l'enregistrement est transmis au programme par l'intermédiaire de la zone décrite dans la FILE-SECTION. Par contre, dans le second format, le même enregistrement est copié dans une zone définie dans WORKING-STORAGE SECTION.
Pour illustrer le mécanisme de lecture d'un fichier, considérons le cas simple (représenté dans le schéma ci-dessus) du fichier FICHIER-A constitué par six enregistrement de longueur fixe de 100 caractères avec un facteur de blocage égal à 3.
Le contenu de la zone REC-A est inconnu avant l'ouverture du fichier. Cette ouverture comporte l'allocation en mémoire d'une zone tampon dont la dimension permet de contenir trois enregistrement de 100 caractères chacun, c'est-à-dire le résultat de la lecture du premier bloc de trois enregistrements dans le tampon. A ce moment, le contenu de REC-A est encore indéfini. La première lecture (READ) n'effectue aucun transfert de données du tampon vers REC-A. Elle se borne à créer un pointeur entre REC-A et le premier enregistrement dans le tampon, rendant aussi disponible pour le programme le contenu de l'enregistrement. Si la clause INTO est spécifiée, le contenu du premier enregistrement est copié dans REC-A-W-S. La seconde lecture fait pointer REC-A sur ce second enregistrement et copie son contenu du tampon vers REC-A-W-S... La quatrième lecture, trouvant le tampon vide, va lire le second bloc du fichier. Elle « libère » le premier enregistrement du tampon, effectuant comme toujours le copie dans REC-A-W-S. La lecture suivante rencontre la marque de fin-fichier et exécute la phrase

```
DISPLAY 'FIN FICHIER-A'
            UPON PRINTER
CLOSE   FICHIER-A
```

La fermeture du fichier comporte l'effacement du tampon (le contenu de REC-A n'est plus disponible). Le processus est le même pour l'instruction WRITE, en sens inverse.

Dans le format

WRITE nom-de-l'enregistrement

l'instruction effectue les opérations d'écriture en se positionnant directement sur le tampon alors qu'avec la forme :

WRITE nom-de-l'enregistrement FROM nom-de-donnée

le contenu du champ de WORKING-STORAGE, nom-de-donnée, est copié dans le tampon. Notons que la clause INTO pour l'instruction READ et FROM pour WRITE constituent des transferts de données d'un point à un autre de la mémoire. Elles respectent donc les règles qui régissent ce type d'opération. Notons que, si dans une instruction READ, le nom-de-donnée accompagnant la clause INTO a une longueur inférieure à la description d'un enregistrement à lire, l'enregistrement sera tronqué sur la droite.
Soit l'exemple suivant :

```
DATA DIVISION.

FILE SECTION.
FD CARTES.
LABEL RECORD OMITTED.
01 CARTE                          PIC X(80).

WORKING-STORAGE SECTION.

01 CARTE-W-S.
   05 TYPE-CARTE                  PIC X(2).
   05 DATE-CARTE.
      10 JOUR                     PIC 9(2).
      10 MOIS                     PIC 9(2).
      10 ANNEE                    PIC 9(2).
   05 ARTICLE.
      10 SERIE                    PIC X(2).
      10 NUMERO                   PIC 9(4).
   05 FILLER                      PIC X(3).
```

(Notons que l'enregistrement CARTE du fichier CARTES est défini par 80 caractères alors que le champ de WORKING-STORAGE CARTE-W-S ne comporte que 17 caractères).

```
PROCEDURE DIVISION.

LIRE-FICHIER.
OPEN INPUT CARTES.
READ CARTES INTO-CARTE-W-S
                AT END...
```

Si le contenu de la carte lue (présente dans le tampon) est, par exemple,

AA101283XZ7411CABLETELEPHONIQUE

on a dans le champ CARTE-W-S

AA101283XZ7411CAB

Inversement, si le champ CARTE-W-S avait été

01 CARTE-W-S PIC X(150).

l'instruction

READ CARTES INTO CARTE-W-S
AT END...

aurait transféré le contenu entier de la carte lue, remplissant les positions de la 81e à la 150e avec des caractères blancs.
La règle énoncée demeure également valable pour l'instruction WRITE :

- Si la longueur du champ spécifié dans la clause FROM est inférieure à la description de l'enregistrement d'output, celui-ci chargera le champ entier de WORKING STORAGE, plus une série de blancs, jusqu'au remplissage de l'enregistrement.
- Dans le cas où le champ de FROM a, au contraire, une longueur supérieure à l'enregistrement, son contenu sera coupé aux dimensions maximales de l'enregistrement d'output.

Bien que les formats qui définissent directement les tampons entrée/sortie soient manifestement plus efficaces (car ils ne prévoient pas le transport des données d'un point à un autre de la mémoire), ils imposent une gestion des instructions plus rigoureuse. Il importe, en effet, de connaître exactement le moment où le contenu de l'enregistrement courant est fiable et quand il ne l'est plus. Voyons, à ce propos, une série d'instructions erronées qui illustrent les problèmes

auxquels nous avons fait allusion.

Premier cas :
WRITE RECORD-A.
MOVE RECORD-A TO DISPONIBLE-REC.
Chaque opération d'écriture rend le contenu de l'enregistrement courant inutilisable puisqu'elle l'écrase par une nouvelle valeur. Il en est de même pour READ.

Deuxième cas :
MOVE DISPONIBLE-REC TO RECORD-A
OPEN INPUT FICHIER-A.
L'opération d'ouverture d'un fichier ne rend pas disponible l'enregistrement qui, ailleurs, ne contient plus la valeur DISPONIBLE-REC.

Troisième cas :
READ FICHIER-A
AT END
MOVE RECORD-A TO DISPONIBLE-REC.
La condition de fin de fichier ne valide pas la valeur de l'enregistrement courant.

Ecriture sur fichiers d'imprimante. Les modalités d'utilisation de l'instruction WRITE pour les fichiers sur disque restent également valables pour les fichiers assignés à l'imprimante.
Pour ces derniers, il existe des clauses permettant de contrôler la tête d'impression. Rappelons cependant que les fichiers assignés à l'imprimante (PRINTER) doivent toujours être déclarés avec la clause LABEL RECORD OMITTED et que la longueur de l'enregistrement doit, au maximum, faire 132 caractères (taille du tampon de l'imprimante) plus 1, car une position est réservée au caractère de contrôle de la tête. Le format complet de l'instruction WRITE est développé dans le tableau ci-dessous.
Les états produits par un programme Cobol sont rarement de simples listings. En effet, on a souvent besoin de mettre en évidence les résultats des calculs sous forme de tableaux. De plus, un tableau peut aussi être très long ; aussi est-il indispensable que non seulement la première mais chaque page d'un état comporte une ou plusieurs lignes de marge expliquant la nature et la signification des informations reproduites. Examinons maintenant les clauses simples du WRITE assurant la gestion des fichiers d'imprimante.

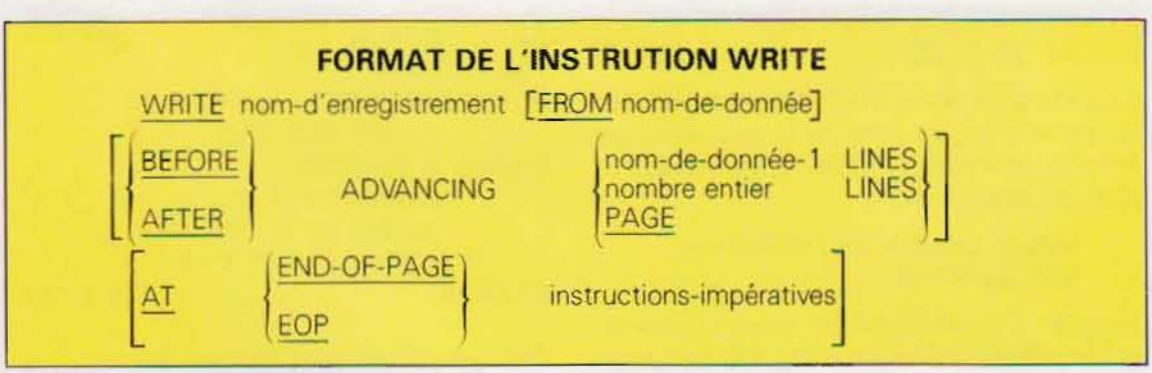
Clauses de saut de lignes. Si aucune clause n'est spécifiée dans l'instruction WRITE, celle-ci imprime le contenu de l'enregistrement dans la ligne qui suit immédiatement celle qui est imprimée. Si un autre type d'impression est demandé, le compilateur tient compte du nombre de lignes devant être sautées avant l'impression de la ligne contenue dans la mémoire.
En d'autres termes,

```
WRITE LIGNE FROM LIGNE-W-S
AFTER ADVANCING 3 LIGNES.
```

équivaut à dire : « après avoir sauté 3 lignes (AFTER ADVANCING 3 LINES) écrire (WRITE) l'enregistrement LIGNE en prélevant le contenu du champ de WORKING-STORAGE LIGNE-W-S ».
On peut d'ailleurs obtenir l'impression de la ligne en premier et seulement ensuite le saut d'un certain nombre de lignes en utilisant le format :

```
WRITE LIGNE FROM LIGNE-W-S
BEFORE ADVANCING 3 LINES.
```

FORMAT DE L'INSTRUTION WRITE



A savoir : « Ecrire l'enregistrement LIGNE en prélevant le contenu du champ LIGNE-W-S avant d'avancer de 3 lignes.

Le nombre de lignes dont on fait progresser la tête (en réalité c'est le papier qui avance) peut aussi être un nombre entier sans signe contenu dans un champ numérique élémentaire défini dans WORKING-STORAGE SECTION. Les exemples précédents sont donc équivalents à :

```
WRITE LIGNE FROM LIGNE-W-S
AFTER ADVANCING SAUT LIGNES.
WRITE LIGNE FROM LIGNE-W-S
BEFORE ADVANCING SAUT LINES.
```

où SAUT est défini dans WORKING-STORAGE SECTION ainsi :

```
01 SAUT PIC 9 (2) VALUE 3.
```

Indépendamment de la méthode utilisée pour annoncer les lignes à sauter, leur nombre ne peut dépasser 99.
Enfin, on observe que les mots ADVANCING et LINES ne sont pas obligatoires : de ce fait, les mêmes instructions peuvent s'écrire :

```
WRITE LIGNE FROM LIGNE-W-S
AFTER 3.
WRITE LIGNE FROM LIGNE-W-S
AFTER SAUT.
WRITE LIGNE FROM LIGNE-W-S
BEFORE 3.
WRITE LIGNE FROM LIGNE-W-S
BEFORE SAUT.
```

Clause de saut de pages. Comme pour le saut de lignes, l'instruction WRITE comporte une gestion de la page qui reçoit l'impression de la ligne. La clause PAGE permet de sauter à la page suivante :

```
WRITE LIGNE FROM LIGNE-W-S
AFTER PAGE.
```

équivaut à : « après (AFTER) s'être positionné sur la première ligne de la nouvelle page (PAGE), écrire (WRITE) LIGNE en utilisant le champ LIGNE-W-S » ;

```
WRITE LIGNE FROM LIGNE-W-S
BEFORE PAGE.
```

Signifie : « écrire LIGNE en utilisant le contenu de LIGNE-W-S avant (BEFORE) de sauter à la première ligne utile de la nouvelle page ». Si elle n'est pas explicitement modifiée par le programmeur avec des clauses spéciales, une telle ligne est directement gérée par le système d'exploitation.

La clause LINAGE. Pour organiser de manière personnalisée les pages d'un état, le programmeur a recours à la clause LINAGE dans la FILE SECTION. Cette clause signale au compilateur les points suivants :

- Le nombre total de lignes à imprimer dans le champ de la page (LINAGE) doit être plus grand que 0.
- Le nombre de lignes vides précédant les lignes à imprimer (TOP) peut être égal à 0.
- Le nombre de lignes vides suivant les lignes à imprimer (BOTTOM) peut être égal à 0.
- Le numéro de la ligne, où doit être imprimé l'enregistrement courant avant de sauter à unr nouvelle page (FOOTING), doit se situer entre les lignes pouvant être imprimées.

Le format complet de la clause LINAGE s'écrit comme indiqué dans le tableau ci-dessous.
Tous les paramètres annoncés doivent être des nombres entiers ou des noms de données numériques élémentaires sans signe.
Ecrire donc

FORMAT DE LA CLAUSE LINAGE

```
LINAGE IS nombre-ligne-pouvant être imprimées LINES
[WITH FOOTING AT ligne-de-saut]
[LINES AT TOP marge-supérieure]
[LINES AT BOTTON marge-inférieure]
```

```
DATA DIVISION.
FILE SECTION.
FD IMPRIMANTE
LABEL RECORD OMITTED
LINAGE IS 56 LINES
        LINES AT TOP 6
        LIGNES AT BOTTOM 4
        WITH FOOTING AT 56
01 LIGNE                    PIC X(133).
```

Ces instructions sont équivalentes à :

```
DATA DIVISION.

FILE SECTION.
FD IMPRIMANTE
LABEL RECORD OMITTED

LINAGE IS NOMBRE-LIGNES LINES
   LINES AT TOP MARGE-SUPERIEURE
   LINES AT BOTTOM MARGE-
   INFERIEURE
   WITH FOOTING LIGNE-FOOTING.
01 LIGNE                          PIC X(133).

WORKING-STORAGE SECTION.
01 NOMBRES-LIGNES  PIC 9(2) VALUE 56.
01 MARGE-SUPERIEURE    PIC 9 VALUE 6.
01 MARGE-INFERIEURE    PIC 9 VALUE 4.
01 LIGNE-FOOTING    PIC 9(2) VALUE 56.
```

Les clauses de LINAGE sont optionnelles. Dans le cas où elles ne sont pas utilisées, les valeurs suivantes sont prises par défaut :

marge-supérieure = 0
marge-inférieure = 0
Ligne-de-saut (footing) = nombres de lignes pouvant être imprimées.

La clause LINAGE accélère la gestion de l'imprimante. En effet, au moment de l'ouverture d'un fichier d'imprimante auquel est associée la cause LINAGE, on assigne également un registre spécial (LINAGE-COUNTER) géré automatiquement par le Cobol pendant l'exécution des instructions WRITE. Ceci est illustré par l'exemple ci-dessous. On en déduit que, lorsqu'on utilise la clause LINAGE dans la description d'un fichier d'imprimante, le Cobol est en mesure d'identifier, par l'intermédiaire du LINAGE-COUNTER, la fin de la page délimitée par les paramètres en FILE-SECTION. En utilisant cette facilité, il est possible de déclarer, dans l'instruction WRITE, une action que le programme doit accomplir immédiatement après la fin d'une page.

IDENTIFICATION DIVISION

```
—
FILE SECTION
FD IMPRIMANTE
    LABEL RECORD OMITTED
    LINAGE IS 50.
           TOP 10
           BOTTOM 6.
01 LIGNE                          PIC X(133).
WORKING-STORAGE SECTION.
01 LIGNE-W-S.
    05
    —
    —
PROCEDURE DIVISION.
OUVERTURE-FICHIER.
    OPEN OUTPUT IMPRIMANTE
    —
    —
    —
    —
    —
WRITE LIGNE FROM LIGNE W-S.
    —
    —
WRITE LIGNE FROM LIGNE W-S AFTER 3.
    —
WRITE LIGNE FROM LIGNE-W-S.
    —
    —
    —
    —
    —
    —
WRITE LIGNE FROM LIGNE-W-S AFTER
PAGE
```

Assigne le fichier et met à 0 le contenu du LINAGE-COUNTER.

Imprimer le contenu de LIGNE-W-S sur la 11e ligne de la feuille, c'est-à-dire après une marge supérieure à 10 lignes (TOP 10). Le LINAGE-COUNTER est incrémenté de 1.

Avancer le papier de 3 lignes, imprimer le contenu de LIGNE-W-S et incrémenter de 3 le LINAGE-COUNTER.

On suppose, qu'avec l'instruction WRITE utilisée en ce point, le chariot est positionné sur la 50e ligne, à savoir sur la dernière ligne susceptible d'écrire, en accord avec la clause LINAGE. Dans ce cas, le contenu de LIGNE est imprimé sur la première ligne utilisable à la page suivante. Le LINAGE-COUNTER est mis à 1, de façon à tenir compte uniquement des lignes imprimées dans la page commencée.

L'opération exécutée est analogue à la précédente pour le WRITE, mais elle est commandée par le programmeur qui, à ce point du programme, désire changer de page.

Supposons, par exemple, que nous voulions imprimer une marge au début de chaque page. Dans ce cas, si les instructions nécessaires à l'impression de la marge sont toutes contenues dans une SECTION appelée MARGE, on écrit :

```
WRITE LIGNE FROM LIGNE-W-S
AT END-OF-PACE PERFORM MARGE
```

ou, d'une manière plus concise

```
WRITE LIGNE FROM LIGNE-W-S
EOP PERFORM MARGE.
```

Les instructions ACCEPT et DISPLAY

Ces deux verbes d'E/S sont respectivement comparables à READ et à WRITE, mais ne sont opérationnels que sur de faibles quantités de données.
Avec ACCEPT, la programmeur est en mesure d'arrêter un calcul pour saisir des données sur la console système, le lecteur de cartes ou par certains registres particuliers du calculateur. Contrairement à ce qui se passe pour l'instruction READ, ACCEPT ne demande l'attribution d'aucune carte ni à l'intérieur ni à l'extérieur du programme.
L'instruction a le format décrit ci-dessous et permet de transférer, dans le champ de WORKING-STORAGE nom-de-donnée, l'entrée provenant d'une des zones systèmes (FROM) indiquées entre les accolades. De cette manière, le programme en cours reçoit de l'extérieur des paramètres variables.
Ces derniers ont des fonctions

- de documentation,
- de paramétrage,
- de communication avec l'opérateur.

On verra en détail la signification de cet ordre dans l'exemple étudié après la description du verbe DISPLAY. Pour le moment, on note que les éléments suivants sont transférés dans le champ, nom-de-donnée :

1 / la réponse numérisée à la console de l'opérateur quand la clause FROM CONSOLE est spécifiée ;
2 / le contenu d'une carte quand on emploie la clause FROM CARD-READER ;
3 / la date du jour avec le format JJMMAA quand on utilise la clause FROM DATE ;
4 / le numéro du jour avec le format JJJAA quand on utilise la clause FROM DAY;
5 / l'heure du jour exprimée avec 8 caractères avec le format HHMMSSCC quand la clause FROM TIME est utilisée.

Pour comprendre la signification des notations utilisées dans les trois derniers formats, on suppose que le programme a été exécuté le 15 janvier 1984 à 8 heures, 12 minutes, 35 secondes et 75 centièmes.
Les trois instructions suivantes

```
ACCEPT DATE-ACTUELLE FROM DATE.
ACCEPT DATE DU JOUR FROM DAY.
ACCEPT HEURE         FROM TIME.
```

imposent les champs respectifs d'arrivée aux valeurs ci-après reportées.

```
DATE-ACTUELLE = 150184
N°-DU-JOUR    = 01584
HEURE         = 08123575
```

On se souvient que les champs d'arrivée doivent avoir été définis avec une longueur respectivement de 6,5 et 8 caractères, numériques et sans signe.

Le verbe DISPLAY permet d'envoyer un message à la console système et à l'imprimante.

FORMAT DE L'INSTRUTION ACCEPT

ACCEPT nom-de-donnée FROM { CONSOLE | CARD-READER | DATE | DAY | TIME }

Un tel message peut être indifféremment composé de constantes alphanumériques ou du contenu des champs décrits dans le WORKING-STORAGE. Le format général de l'instruction figure dans le tableau. Un exemple d'emploi des instructions ACCEPT et DISPLAY est donné dans le tableau ci-dessous, relatif à un programme simple.

FORMAT DE L'INSTRUCTION DISPLAY

```
DISPLAY { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]  UPON { CONSOLE }
        { constante-1     }   { constante-2     }          { PRINTER }
```

EXEMPLE D'EMPLOI DES INSTRUCTIONS ACCEPT ET DISPLAY

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ACCEPT DISPLAY.
       REMARKS. CE PROGRAMME EST UN EXEMPLE D'EMPLOI
                DES INSTRUCTIONS ACCEPT ET DISPLAY.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.....................
       OBJCET-COMPUTER.....................
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT MOUVEMENTS ASSIGN TO DISC MOUVEMENTS.
           SELECT STATISTIQUES ASSIGN TO DISC STATISTIQUES.
       DATA DIVISION.
       FILE SECTION.
       FD  MOUVEMENTS.
           LABEL RECORD STANDARD.
       01  REC-MOUV                        PIC X(14).
       FD  STATISTIQUES
           LABEL RECORD STANDARD.
       01  REC-STA                         PIC X(10).
      *
       WORKING-STORAGE SECTION.
       01  REC-MOUV-WS.
           05  ART-MOUV                    PIC 9(3).
           05  DATE-MOUV.
               01 ANNEE-MOUV               PIC 9(2).
               01 MOIS-MOUV                PIC 9(2).
               01 JOUR-MOUV                PIC 9(2).
           05  QUANTITE                    PIC 9(5).
      *
       01  REC-STA-WS.
           05  ART-STA                     PIC 9(3).
           05  MOIS-STA                    PIC 9(2).
           05  VENDU                       PIC 9(5).
      *
       01  CARTE-PARAMETRE.
           05  TYPE-SK-PAR                 PIC XX.
           05  ART-SK-PAR                  PIC 9(3).
      *
       01  DEMANDER-MOIS                    PIC X(80).  VALUE
           'DIGITALISATION DU NOMBRE DE MOIS A CALCULER'.
      *
       01  MOIS-DE-CONSOLE                 PIC 9(2).
      *
       01  DATE-CALCUL                PIC 9(6).
       01  DATE-CA  REDEFINES DATE-CALCUL.
           05  ANNEE-CA                    PIC 9(2).
           05  MOIS-CA                     PIC 9(2).
           05  JOUR-CA                     PIC 9(2).
      *
       01  HEURE-CALCUL               PIC 9(8).
       01  HEURE-CA  REDEFINES HEURE-CALCUL.
           05  HEURE                       PIC 9(2).
           05  MINUTE                      PIC 9(2).
```

```
           05  SECONDES                        PIC 9(2).
           05  CENTIEMES                       PIC 9(2).
       *
       01  CALCUL.
           05  FILLER                          PIC X(17)  VALUE
               'CALCUL DU'.
           05  JOUR-DISPLAY                    PIC 99.
           05  FILLER                          PIC X      VALUE '/'.
           05  MOIS-DISPLAY                    PIC 99.
           05  FILLER                          PIC X      VALUE '/'.
           05  ANNEE-DISPLAY                   PIC 99.
           05  FILLER                          PIC X(10)  VALUE
               '    AUX'.
           05  HEURES-DISPLAY                  PIC 99.
           05  FILLER                          PIC X      VALUE ':'.
           05  MINUTES-DISPLAY                 PIC 99.
           05  FILLER                          PIC X      VALUE ':'.
           05  SECONDES-DISPLAY                PIC 99.
           05  FILLER                          PIC X      VALUE ':'.
           05  CENTIEMES-DISPLAY               PIC 99.
       *
       01  LECTURE.
           05  FILLER                          PIC X(42)  VALUE
               'LECTURE ET CODAGE DE L'ARTICLE'
           05  ART-DISPLAY                     PIC 9(3).
       PROCEDURE DIVISION.
       DEBUT SECTION.
       ACCEPTE.
           ACCEPT DATE-CALCUL FROM DATE.
           ACCEPT HEURE-CALCUL FROM TIME.
           DISPLAY DEMANDER-MOIS
                   UPON CONSOLE.
           ACCEPT MOIS-DE-CONSOLE FROM CONSOLE.
       DECIDE-OPEN.
           IF   MOIS-DE-CONSOLE = 1
                OPEN OUTPUT STATISTIQUES
           ELSE
                OPEN EXTEND STATISTIQUES.
       ACCEPTE-SK-PAR.
           ACCEPT CARTE-PARAMETRE FROM CARD-READER.
           OPEN INPUT MOUVEMENTS.
       LIS-MOUVEMENTS.
           READ MOUVEMENTS INTO REC-MOUV-WS
                AT END
                PERFORM FERMETURE
                STOP RUN.
           IF   ART-MOUV NOT = ART-SK-PAR
                GO TO LIRE-MOUVEMENTS.
           PERFORM CALCUL-RECORD.
           WRITE REC-STA FROM REC-STA-WS.
           GO TO LIRE-MOUVEMENTS.
       *
       CALCUL-RECORD SECTION.
       CA-REC.
           ......
       CA-REC-EX. EXIT.
       FERMETURE SECTION.
       FERME.
           CLOSE MOUVEMENTS.
           CLOSE STATISTIQUES.
           MOVE ART-SK-PAR TO ART-DISPLAY.
           MOVE ANNEE-CA   TO ANNEE-DISPLAY.
           MOVE MOIS-CA    TO MOIS-DISPLAY.
           MOVE JOUR-CA    TO JOUR-DISPLAY.
           MOVE HEURES     TO HEURES-DISPLAY.
           MOVE MINUTES    TO MINUTES-DISPLAY.
           MOVE SECONDES   TO SECONDES-DISPLAY.
           MOVE CENTIEMES  TO CENTIEMES-DISPLAY.
           DISPLAY CALCUL UPON PRINTER.
           DISPLAY LECTURE UPON PRINTER.
           DISPLAY '** FIN CALCUL **'
                   UPON PRINTER.
       FERMETURE-EX. EXIT.
       *
```

Le programme des instructions ACCEPT et DISPLAY doit :

1/ lire un fichier sur le disque qui stocke les mouvements mensuels de tous les articles vendus dans un magasin,

2/ extraire uniquement les données d'un article codé sur carte perforée,

3/ mettre en file d'attente les données traitées derrière le fichier de statistiques,

4/ créer ou ouvrir en extension le fichier STATISTIQUES et demander au pupitreur à quel mois (janvier ou autre) correspondent les données en cours de traitement,

5/ imprimer en fin de traitement le message suivant :

TRAITEMENT DU (date) A (heure)
EXTRACTION ET MISE EN FILE D'ATTENTE POUR L'ARTICLE (code-article)
FIN TRAITEMENT.

Ces diverses opérations sont schématisées dans l'organigramme de la page suivante.

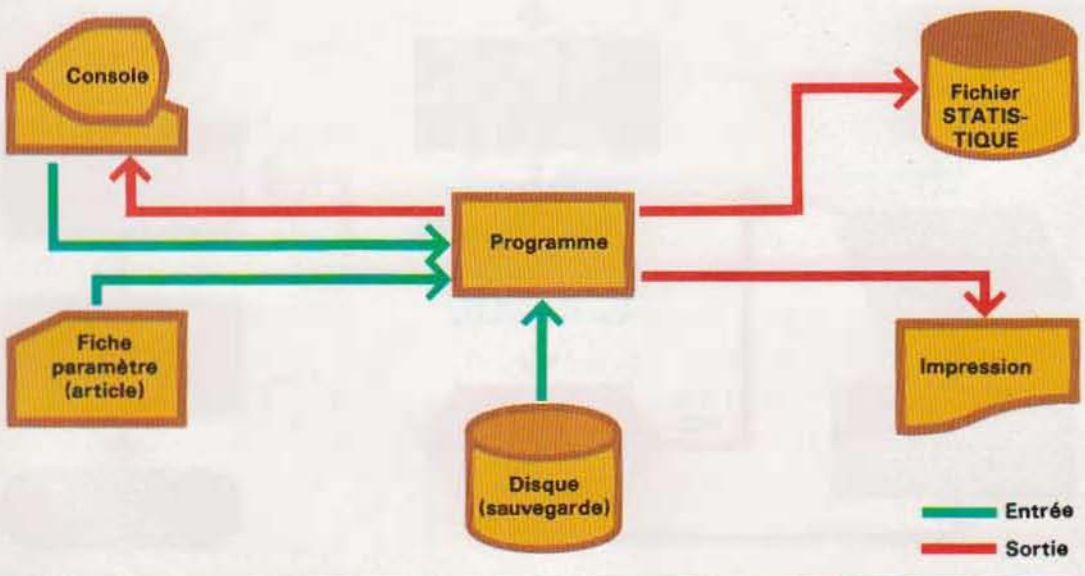
Verbes arithmétiques

Ils permettent de traiter les données rangées en mémoire pour le calcul de certaines formules mettant en jeu ces mêmes données.
Un langage de programmation doit être apte à traiter sans difficulté les informations provenant du type d'application pour lequel il a été conçu. S'agissant du Cobol, qui comporte un ensemble limité d'opérations mathématiques, ce critère est totalement pris en compte. Il répond parfaitement aux exigences de la gestion commerciale.
Avant d'examiner de près les instructions arithmétiques, il est indispensable de connaître la représentation des données numériques dans la mémoire. Nous décrirons ensuite certaines clauses qui permettent de choisir un type de représentation plutôt qu'un autre, de manière à structurer au mieux le traitement des données numériques.

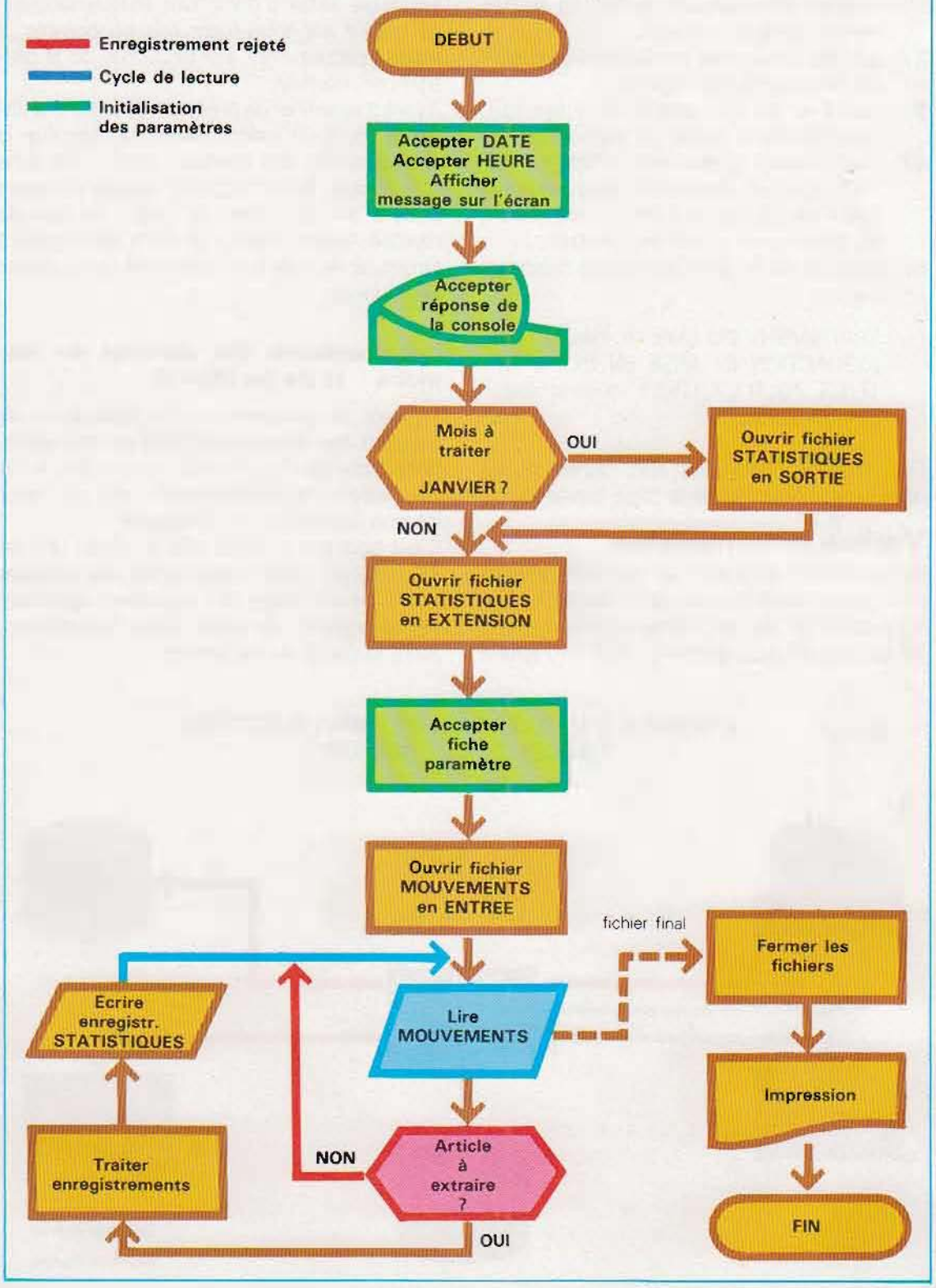
Représentation des données en mémoire : la clause USAGE

Pendant le déroulement des opérations de transfert des données (MOVE) ou des opérations arithmétiques, l'ordinateur procède à des conversions de représentation, afin de traiter les données de façon homogène.
C'est pourquoi, adopter dès le départ une représentation interne appropriée des données décharge le système des opérations répétitives de conversion, réduisant ainsi considérablement la durée du traitement.

EXEMPLE D'UTILISATION DES INSTRUCTIONS "ACCEPT" ET "DISPLAY"



EXEMPLE D'UTILISATION DES INSTRUCTIONS "ACCEPT" ET "DISPLAY"



En ANS COBOL, deux modes de représentation ont été prévus :

DISPLAY (affichage)
(codification hexadécimale)
COMPUTATIONAL (calcul)
(codification binaire)

Pour définir une donnée codée binaire, on peut écrire indifféremment :

```
01 CHAMP    PIC S9 (5)
            USAGE IS COMPUTATIONAL.
```

ou encore

```
01 CHAMP    PIC S9 (5) COMP.
```

Si la clause USAGE (dans sa forme entière ou abrégée) est omise, le compilateur opte par défaut pour une représentation type DISPLAY. Pour lui, les 3 descriptions :

```
01 CHAMP    PIC S9 (5)
01 CHAMP    PIC S9 (5)
            USAGE IS DISPLAY.
01 CHAMP    PIC S9 (5) DISPLAY.
```

sont totalement équivalentes.

USAGE DISPLAY. Dans ce mode, chaque chiffre occupe un caractère en mémoire.
Ainsi, la clause USAGE n'ayant pas été spécifiée dans la description :

```
01 CHAMP    PIC 9 (4) VALUE 6.
```

le compilateur choisit DISPLAY.
Le champ ayant été initialisé à 6, sa représentation interne est 0006.

USAGE COMPUTATIONAL. Cette clause (COMP) indique au compilateur que le contenu du champ examiné doit être codé en binaire. Le recours à ce code n'oblige pas le compilateur à effectuer des opérations de conversion puisqu'il reconnaît ce mode de représentation. La clause USAGE COMP réduit les temps de traitement et occupe un emplacement mémoire restreint. En effet, alors qu'un champ avec USAGE DISPLAY prend autant d'octets qu'il y a de caractères dans la PICTURE, avec la clause USAGE COMP, les mêmes nombres sont représentés mais occupent un espace mémoire inférieur.
Considérons la définition d'un champ devant contenir jusqu'à 10 chiffres.
Si sa description était

```
01 CHAMP    PIC S9 (10).
```

le compilateur réserverait 10 octets en mémoire.
Ce même champ décrit autrement :

```
01 CHAMP    PIC S9 (10)
            USAGE COMPUTATIONAL.
```

ne prendrait que 8 octets.
Cet avantage est plus parlant pour des champs d'une longueur supérieure à 10 chiffres. Dans les limites maximales autorisées par le Cobol (18 chiffres),

```
01 CHAMP    PIC S9 (18) COMP.
```

l'espace mémoire occupé sera toujours égal à 8 octets.
Le tableau des équivalences ci-dessous donne l'occupation effective en mémoire pour différents champs COMPUTATIONAL affichés dans la clause PICTURE

Nombre de chiffres (9) dans la PICTURE	Espace mémoire
de 1 à 4	2 octets
de 5 à 9	4 octets
de 10 à 18	8 octets

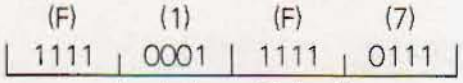
Pour bien saisir la différence entre les deux représentations analysées ici, considérons deux champs, CHAMP-1 et CHAMP-2, déclarés respectivement DISPLAY et COMPUTATIONAL :

```
01 CHAMP-1    PIC S9(2).
01 CHAMP-2    PIC S9(2) COMP.
```

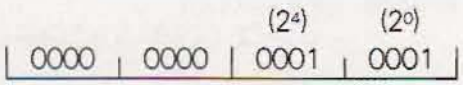
L'un et l'autre occupent 2 octets chacun : CHAMP-1 parce que la clause PICTURE le déclare expressément et CHAMP-2 d'après le tableau des équivalences.

Supposons que les deux champs contiennent le nombre 17, leur représentation s'écrit :

CHAMP-1



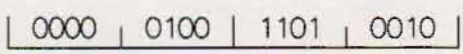
CHAMP-2



Comme on peut le constater, le contenu de CHAMP-2 est le codage binaire normal de 17 sur un mot de 16 bits.
Pour mémoire, le premier bit à gauche du champ est réservé au signe. Si ce bit est mis à 0, le nombre est positif (les nombres négatifs étant calculés par complémentation à 2). Dans le champ :

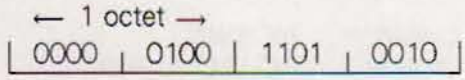
01 CHAMP-3 PIC S9(4) COMP.

le contenu vaudrait - 1234 à un certain moment. Si ce nombre avait été positif, sa représentation interne aurait été

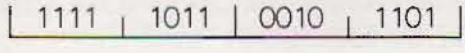


Le nombre - 1234 devant être rangé en mémoire, la machine se charge de représenter le même nombre que celui de la configuration précédente, comme complément à 2, en procédant selon différentes étapes :

1/ Représentation du nombre (+ 1234)

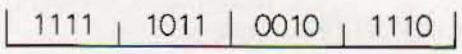


2/ Complément à 1



3/ Addition de 1

4/ Représentation du nombre (- 1234)



De nombreux compilateurs comportent d'autres modes de représentation numérique qui ne sont pas composés dans le Cobol standard. Ces modes sont affichés selon les formats suivants de la clause USAGE :

```
USAGE IS COMPUTATIONAL-3.
USAGE IS COMPUTATIONAL-1.
USAGE IS COMPUTATIONAL-2.
```

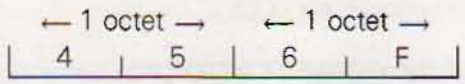
formats équivalents à :

```
COMP-3.
COMP-1.
COMP-2.
```

USAGE COMP-3. Cette représentation, dite aussi **packed** (abrégée), réserve un octet pour tous les deux chiffres du champ décrit, à l'exception du dernier octet à droite où un seul chiffre est mémorisé, plus l'emplacement du signe.
Considérons le champ :

01 CHAMP-4 PIC 09(3) COMP-3.

dont la valeur serait, à un moment donné, 456. Etant donné qu'avec USAGE COMP-3 chaque chiffre occupe un demi-octet (le dernier étant réservé au signe), on aura la représentation interne suivante :

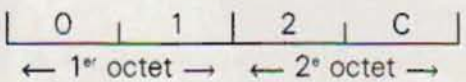


où F tient compte du fait que CHAMP-4 ne prévoit pas le signe (le symbole S manquant dans la PICTURE).
Observons que l'assignation en mémoire des octets nécessaires à la PICTURE est telle qu'elle occupe toujours un nombre entier d'octets. Autrement dit, un champ de deux chiffres avec le signe [(PIC S9(2)] nécessiterait l'utilisation de 3 demi-octets : 2 pour les chiffres et 1 pour

le signe. Dans ce cas, le compilateur assigne le 4e demi-octet, en remplissant la partie gauche non utilisée avec un 0.
Soit le champ :

```
01 CHAMP-5      PIC S9(2) COMP-3.
```

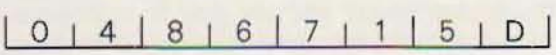
et supposons qu'à un certain moment CHAMP-5 contienne la valeur (+12). Sa représentation interne sera :



C indique que la valeur contenue dans le champ est positive.
Pour compléter la description de la clause COMP-3, considérons le champ :

```
01 CHAMP-6      PIC S9(4)V9(2)COMP-3.
```

L'occupation mémoire est de 6 demi-octets (soit le nombre de chiffres déclarés par le PICTURE) plus un demi-octet pour le signe, plus un demi-octet imposé par le compilateur pour rendre entier le nombre d'octets ; soit au total, 4 octets.
Prenons un exemple. Si une des valeurs temporaires de CHAMP-6 est (-4867,15), sa représentation sera :



D indiquant le signe négatif.

USAGE COMP-1 et **COMP-2.** Ces deux modes, dits aussi floating point (virgule flottante), sont rarement employés dans le cadre du Cobol, sauf lorsqu'on doit manipuler de très grands nombres, avec toutefois une précision relative.
Grâce à ces représentations internes, on peut traiter des nombres allant de 10^{-78} à 10^{+75}, avec un degré de précision allant jusqu'à la 6e décimale pour le COMP-1, et la 16e décimale pour le COMP-2.
Ceci explique que l'usage de tels nombres soit limité en informatique de gestion, terrain privilégié du Cobol.

Le format de l'USAGE COMP-1 ou COMP-2 est différent de celui des représentations étudiées précédemment. En effet, si un champ est déclaré en virgule flottante, le compilateur lui assigne automatiquement un mot (pour COMP-1) ou deux mots (pour COMP-2) en mémoire.
Par conséquent, un champ déclaré en virgule flottante et en simple précision est décrit par :

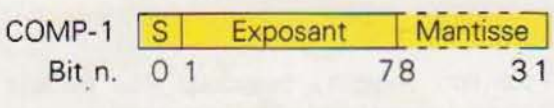
```
01 CHAMP-7      COMP-1.
```

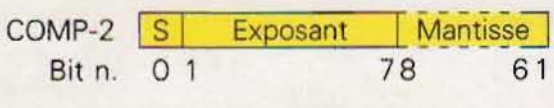
Cela donne, en double précision :

```
01 CHAMP-8      COMP-2.
```

Le codage interne d'un nombre en virgule flottante comporte :
- la **mantisse** ou partie entière pour les chiffres significatifs du nombre,
- l'**exposant,** partie qui détermine la valeur réelle du nombre.

Les ordinateurs sont généralement dotés d'un circuit spécial capable d'effectuer les opérations de « normalisation ». Nous ne mentionnerons donc que les deux modes de présentations COMP-1 et COMP-2 pour un 32 bits ou moins :





Une bonne connaissance des représentations numériques internes constitue une base solide pour la mise en place d'un programme rapide et fiable.
N'oublions pas que le Cobol permet d'utiliser dans le cadre d'une même opération des champs numériques ayant des représentations différentes. Cela est rendu possible par des systèmes de conversion directement gérés par le compilateur. Ils sont donc liés à une hiérarchie qui établit un ordre décroissant de priorité :

Litton PRC

Le microprocesseur se retrouve même dans une caisse enregistreuse.

COMP-2
COMP-1
COMP-3
DISPLAY
COMP

Tout ceci semblera beaucoup plus évident quand on parlera des verbes de type arithématique.

Les expressions arithmétiques

En arithématique, de telles formules permettent d'effectuer une série d'opérations généralement complexes sur des opérandes ou sur des constantes ; le résultat peut être associé à une variable.

$$A = (B + C)^2 + \frac{D}{(E + F)} - (G + H)^P + 100$$

constitue une expression où la valeur de la variable A (à gauche du signe égal) est équivalente au résultat du traitement de la seconde partie dont les valeurs sont, par définition, différentes de 0. En Cobol, le concept d'expression arithmétique est à peu près identique. Le programmeur qui désire calculer la valeur d'une expression dispose des 5 opérations mathématiques fondamentales de l'ordinateur plus les parenthèses, avec lesquelles il peut modifier l'ordre d'exécution des opérations

\+ addition
— soustraction
* multiplication
/ division
** élévation à une puissance.

A l'aide de ces signes, l'expression précédente sera réécrite en Cobol :

```
A = (B+C)**2+D/(E+F) — (G+H) ** P+100
```

Le calcul Cobol respecte toutes les règles de

l'arithmétique, à savoir l'ordre d'exécution des opérations : résolution des élévations à une puissance, ensuite multiplications et divisions et enfin additions et soustractions.
Prenons un exemple : le calcul de l'expression

```
A = 7 + 5*4 - 6**2
```

passe par les étapes suivantes :

```
A = 7 + 5 * 4 + 36
A = 7 + 20 + 36
A = 63
```

Les opérations de même niveau de priorité sont résolues en fonction de leur ordre d'arrivée dans l'expression, de gauche à droite s'il s'agit de + - * /, de droite à gauche pour les élévations à une puissance (**).
Dans l'expression :

```
A = B*C/D + E - F + G **2 + H **3
```

Le déroulement des opérations est le suivant :

1/ H ** 3
2/ G ** 2
3/ B * C
4/ (résultat de B * C) /D
5/ (résultat de B *C /D) + E
6/ (résultat de B * C/D + E) - F
7/ ...

Le programmeur dispose de 9 niveaux de parenthèses pour modifier l'ordre des calculs. De cette façon, il peut regrouper de n'importe quelle façon les données à traiter. Exemple :

```
A = (( B + (C*D)) ** 2 + E/ (F-G) ** 3
```

Les groupements de premier niveau (ici C*D et F-G) sont d'abord traités, en respectant à l'intérieur de ceux-ci les règles générales déjà décrites. Ainsi, dans ce même niveau, on calculera (C*D) avant (F-G).

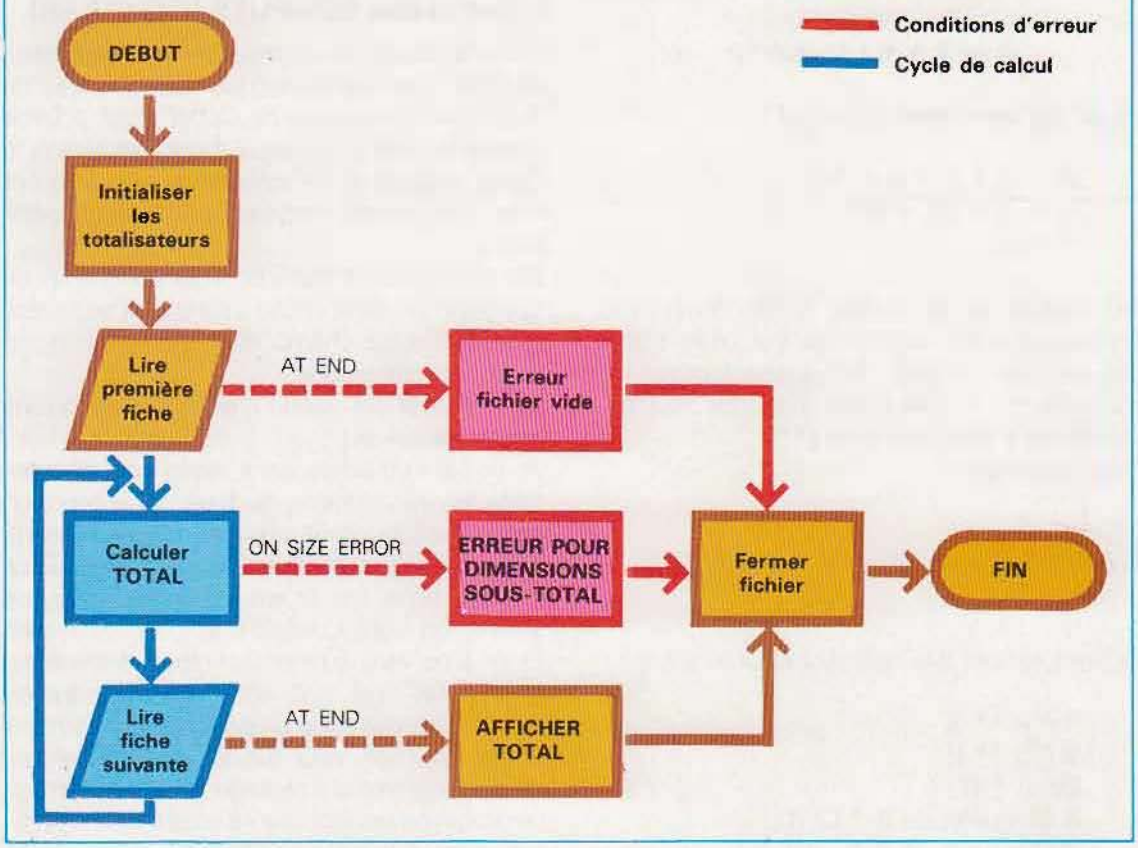
L'instruction COMPUTE (CALCULER)

Jusqu'à présent, les expressions ont été écrites dans leur forme arithmétique classique, tout en respectant la syntaxe du Cobol. Pour pouvoir obtenir le résultat du calcul d'une expression, le Cobol dispose d'une instruction prévue à cet effet (voir format ci-dessous en bas de cette page).
Elle permet de transférer, vers un champ de données, la valeur d'une expression arithmétique d'un autre champ de données ou d'une constante numérique.
Pour illustrer son mode d'emploi, considérons l'exemple suivant :
on doit lire un fichier dans lequel sont enregistrées les informations du type : quantité d'un certain article, coût unitaire, rabais consenti (pourcentage), frais d'expédition à l'unité. Pour chaque fiche lue (c'est-à-dire pour chaque achat), on veut connaître le coût du lot de façon à parvenir au montant global des achats. Le résultat final doit également prendre en compte les résultats précédents, directement entrés par l'opérateur sur console à la demande du programme (message affiché à l'écran). Le problème est représenté sous forme d'organigramme à la page suivante. On a également reproduit le flux des données et leur enregistrement sur une carte perforée (page 1017), ainsi que le programme correspondant (pages 1018 et 1019). Dans cet exemple, il faut remarquer l'insertion d'un compteur de fiches lues (COMPTE-FICHES), qui n'est pas explicitement requis par les spécifications Cobol. En effet, il est bon de toujours prévoir un compteur d'enregistrement (lus ou écrits) de manière à avoir une indication sur les données mouvementées par le programme.

FORMAT GENERAL DE L'INSTRUCTION

```
COMPUTE nom-de-donnée [ROUNDED] { expression arithmétique
                                  nom-de-donnée-1
                                  constante numérique }
    [ON SIZE ERROR phrase impérative]
```

EXEMPLE D'APPLICATION DE L'INSTRUCTION "COMPUTE"



Dans cet exemple, on a examiné tous les formats et toutes les clauses de l'instruction COMPUTE.

COMPUTE COMPTE-FICHES = 0

met à 0 (initialise) le compteur de fiches lues ;

COMPUTE TOTAL = TOTAL-PRECEDENT

range, dans le champ TOTAL, la valeur du champ TOTAL-PRECEDENT
(attention : le signe - est un séparateur inséré dans le nom du champ).

Les expressions :

```
COMPUTE ACHAT-NET ROUNDED =
      (QUANTITE * COUT-UNITAIRE) -
      (QUANTITE * COUT-UNITAIRE)*
      (RABAIS/100)+
      (QUANTITE * FRAIS-UNITAIRES)
```

calculent la valeur de l'expression à droite du signe et les rangent dans le champ ACHAT-NET en respectant l'alignement par rapport à la décimale dans la position prévue par la PICTURE.
Avec ce format, la valeur calculée n'est pas simplement tronquée à la 3e décimale ; si elle est supérieure à 5, elle est arrondie à la décimale immédiatement supérieure. En d'autres termes, si la valeur de l'expression était :

785 642, 3896 (notation européenne),

on lirait, dans ACHAT-NET : 785 642, 390 grâce à la clause ROUNDED. Sans elle, le contenu du champ aurait été :

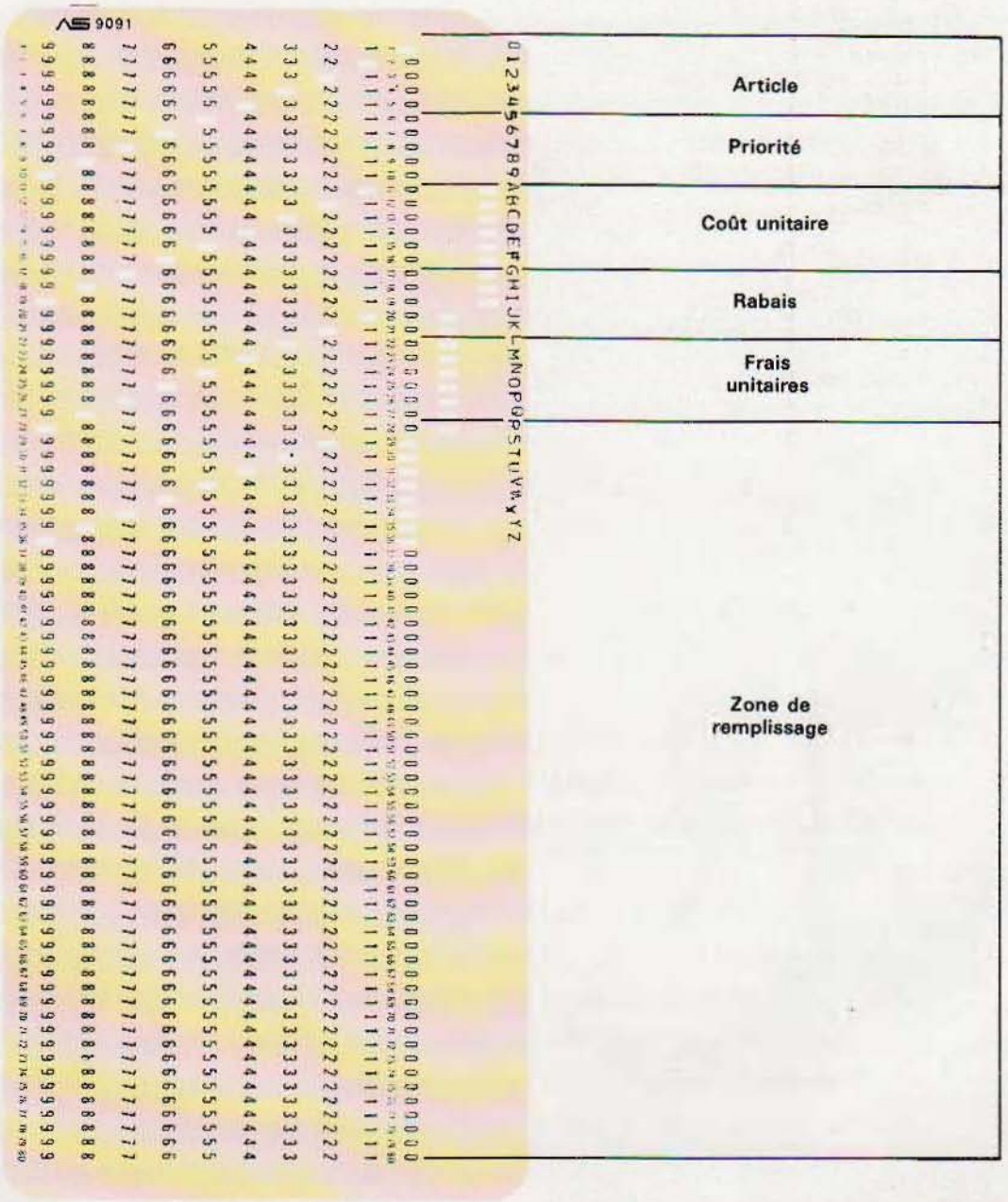
785 642, 389

(toujours en notation européenne).

Après avoir atteint le nombre maximum qu'il peut représenter (quelle qu'en soit la base), un compteur se remet automatiquement à 0.
On dit alors qu'il « sature », tel un compteur

FLUX DES DONNEES

LECTURE D'UNE CARTE



EXEMPLE D'APPLICATION DE COMPUTE

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CALCUL.
       REMARKS. CE PROGRAMME EST UN EXEMPLE
                D'APPLICATION DE L'INSTRUCTION
                COMPUTE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. ..............
       OBJECT-COMPUTER. ..............
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FICHES ASSIGN TO CARD-READER FICHES.
       DATA DIVISION.
       FILE SECTION.
       FD  FICHES
           LABEL RECORD OMITTED.
       01  FICHE                          PIC X(80).
      *
      *
      *
       WORKING-STORAGE SECTION.
       01  FICHE-WS.
           05  ARTICLE                    PIC X(5).
           05  QUANTITE                   PIC X9(5).
           05  COUT UNITAIRE              PIC S9(3)V9(3)  COMP.
           05  RABAIS                     PIC S9(2)V9(3)  COMP.
           05  FRAIS UNITAIRES            PIC S9(3)V9(3)  COMP.
           05  ZONE DE REMPLISSAGE        PIC X(53).
      *
       01  TOTAL PRECEDENT                PIC S9(15)      COMP.
      *
       01  TOTAL                          PIC S9(15)      COMP.
      *
       01  NET-ACHAT                      PIC S9(8)V9(3)  COMP.
      *
       01  SOUS-TOTAL                     PIC S9(15)V9(3) COMP.
      *
       01  COMPTE-FICHES                  PIC S9(5)       COMP.
      *
      *
      *
      *
      *
       PROCEDURE DIVISION.
       MAIN SECTION.
       INITIALISER.
           COMPUTE COMPTE-FICHES = 0
           COMPUTE SOUS-TOTAL = 0
           DISPLAY ' = CONVERTIR LE TOTAL PRECEDENT  :
                   UPON CONSOLE.
           ACCEPT TOTAL-PRECEDENT FROM CONSOLE.
           COMPUTE TOTAL = TOTAL-PRECEDENT.
      *
      *
      *
       OUVRIR-FICHIER.
            OPEN INPUT FICHES.
       PREMIERE-LECTURE.
            READ FICHES INTO FICHES-WS
                 AT END
                 DISPLAY '** FICHIER VIDE **'
                         UPON CONSOLE
                 PERFORM FERMETURE
       CALCUL.
            COMPUTE COMPTE-FICHES = COMPTE-FICHES+1.
            COMPUTE NET-ACHAT ROUNDED = (QUANTITE * COUT-UNITAIRE)-
                                        ((QUANTITE * COUT-UNITAIRE) *
```

```
                                        RABAIS / 100) +
                                       (QUANTITE * FRAIS-UNITAIRES).
        COMPUTE SOUS-TOTAL = SOUS-TOTAL + NET-ACHAT
                             ON SIZE ERROR
                             DISPLAY '* CHAMP SOUS-TOTAL INSUFFISANT *
                         'AVERTIR LE RESPONSABLE *'
                                    UPON CONSOLE
                             PERFORM FERMETURE.
    AUTRES-LECTURES.
        READ FICHES INTO FICHES-WS
                    AT END
                    PERFORM FERMETURE.
        GO TO CALCUL.
  *
  *
  *
  *
  *
    FERMER-SECTION.
    FERMER-FICHIER.
        CLOSE FICHES.
        DISPLAY '** FICHES LUES = ' COMPTE-FICHES
                UPON CONSOLE.
        COMPUTE TOTAL ROUNDED = TOTAL + SOUS-TOTAL.
        DISPLAY ' '
                UPON CONSOLE.
        DISPLAY '** TOTAL ACTUEL = ' TOTAL
                UPON CONSOLE.
        STOP RUN.

  *
```

tel un compteur kilométrique d'automobile. En général, ce type de compteur est prévu pour 5 chiffres ; il est donc en mesure de compter jusqu'à 99 999 kilomètres ; le 100 000e kilomètre sature le compteur qui se remet à 00000. Il en est de même des registres d'ordinateurs comme des champs décrits dans un programme Cobol.

Toujours dans le but d'assurer l'intégrité des informations contenues dans un compteur, le compilateur applique la clause ON SIZE ERROR à toutes les instructions arithmétiques. Grâce à elle, le programme est en mesure de signaler la saturation d'un compteur au moment où son contenu est incrémenté. S'agissant d'une condition anormale, de nature à fausser les résultats, une phrase impérative est associée à cet événement, permettant au programmeur de bien le gérer.

Dans l'exemple cité, la saturation du compteur SUBTOTAL (sous-total) entraîne un blocage de la situation ; il est donc nécessaire que le système signale cette anomalie à l'écran avant de procéder à l'interruption du traitement.

De l'exemple décrit, on peut tirer des règles générales valables aussi bien pour COMPUTE que pour toutes les instructions arithmétiques qui seront analysées par la suite.

1/ Tous les noms utilisés dans les expressions numériques doivent représenter des données numériques définies dans la division données (DATA DIVISION).

2/ Toutes les constantes utilisées dans les instructions de type arithmétique doivent être numériques.

3/ La longueur prévue pour chaque opérande d'une expression arithmétique est de 18 caractères au maximum.

4/ Les PICTURE et la clause USAGE des différentes opérandes d'une expression arithmétique ne doivent pas nécessairement être uniformes, dans la mesure où, au cours des opérations, aussi bien l'alignement sur le point décimal.(V) que la conversion du format de chaque opérande sont automatiquement réalisés. Cette conversion a lieu selon un ordre hiérarchique précis (voir page 1014).

Dans les champs :

```
01 CHAMP-1      PIC S9(8)V9(37).
01 CHAMP-2      COMP-1.
01 CHAMP-3      COMP-2.
```

l'instruction :

```
COMPUTE CHAMP-3 =
            CHAMP-1 * CHAMP-2
```

déclenche les opérations suivantes :

- CHAMP-1 est transformé en COMP-1 par DISPLAY (en fait COMP-1 est la représentation la plus prioritaire),
- calcul de CHAMP-1 * CHAMP-2 en codage COMP-1,
- conversion du résultat précédent de COMP-1 en COMP-1 en COMP-2.

5/ Le format des données utilisées dans une expression arithmétique ne contient aucun symbole associé à l'impression.

Le verbe COMPUTE peut lancer les 5 opérations sur une série d'opérandes groupées dans une expression unique. Les verbes qui seront ultérieurement analysés n'autorisent, en revanche, qu'un seul type d'opération sur les données décrites.

Le verbe ADD

Cette instruction additionne, selon troix modalités, deux ou plusieurs nombres et range le résultat dans un champ d'arrivée.
Malgré sa complexité apparente (voir ci-dessous) le premier format est très proche de la structure classique de l'addition. Considérons en effet les champs suivants :

```
01 CUMULATEUR-1  PIC S9(4)V9(2) COMP.
01 CUMULATEUR-2  PIC S9(4)V9(2) COMP.
01 CUMULATEUR-3  PIC S9(4)V9(2) COMP.
```

On veut additionner leur contenu, ainsi qu'une constante fixe, égale à 3 000, au contenu d'un champ somme défini par

```
01 SOMME      PIC S9(4)COMP.
```

L'opération demandée peut être ainsi codifiée :

```
ADD CUMULATEUR-1
    CUMULATEUR-2
    CUMULATEUR-3
    3000                    TO SOMME.
```

Les descriptions des différents cumulateurs (seconds termes de l'addition), comportent 2 décimales qui seront tronquées dans la dernière phase de l'addition dans le champ-SOMME (celui-ci ne prévoit pas de décimales). Il suffit, pour obtenir un résultat arrondi et non tronqué, de coder de la manière suivante :

```
ADD CUMULATEUR-1
    CUMULATEUR-2
    CUMULATEUR-3
    3000              TO SOMME ROUNDED
```

Comme le champ SOMME prévoit par ailleurs le même nombre de caractères que les cumulateurs, il faut s'attendre à une saturation rapide dans son contenu, avec perte consécutive de l'intégrité de la donnée.
La logique de l'instruction COMPUTE permet aussi de contrôler le dépassement de capacité (overflow) de SOMME : grâce à la clause ON SIZE ERROR et à une phrase impérative indiquant le type d'opération à entreprendre dans une telle éventualité.

Exemple :

```
ADD CUMULATEUR-1
    CUMULATEUR-2
    CUMULATEUR-3
    3000      TO SOMME ROUNDED
              ON SIZE ERROR
              DISPLAY « CHAMP SOMME
                  EN DEPASSEMENT DE
                  CAPACITE »
                  UPON CONSOLE
              GO TO FINE.
```

PREMIER FORMAT DE L'INSTRUCTION ADD

TO nom-de-donnée-n [ROUNDED] nom-de-donnée-p [ROUNDED]
[ON SIZE ERROR phrase impérative].

Précisons que le champ d'arrivée, SOMME, doit être un champ numérique ne contenant pas de symboles d'impression dans la PICTURE, à moins d'adopter le 2e format de l'instruction ADD.
Dans ce dernier cas, une fois la somme calculée, le résultat est transféré selon des modalités précises dans le champ d'arrivée nom-de donnée-n.
Ce champ est réservé aux opérations de transfert et ignore les calculs. Il peut donc contenir des symboles d'impression.
Le dernier format de l'instruction ADD exploite la possibilité qu'a le Cobol de décrire les sous-ensembles de deux champs différents à l'aide des mêmes noms.

Considérons les descriptions :

```
01 BLOC-1.
   05 MEMBRE-1.
      10 MEMBRE-11      PIC S9(3) COMP.
      10 MEMBRE-12      PIC S9(2) COMP.
   05 MEMBRE-2          PIC S9(4) COMP.
   05 MEMBRE-3          PIC S9(6) COMP.

01 BLOC-2.
   05 MEMBRE-1.
      10 MEMBRE-11      PIC S9(3) COMP.
      10 MEMBRE-12      PIC S9(2) COMP.
   05 MEMBRE-2          PIC S9(4) COMP.
   05 MEMBRE-3          PIC S9(6) COMP.
```

Pour additionner terme à terme les sous-champs de BLOC-1 et de BLOC-2 à l'aide du 1er format de l'instruction ADD, on doit écrire de la manière suivante :

```
ADD MEMBRE-11      OF BLOC-1
TO MEMBRE-11       OF BLOC-2.
ADD MEMBRE-12      OF BLOC-1
TO MEMBRE-12       OF BLOC-2.
ADD MEMBRE-2       OF BLOC-1
TO MEMBRE-2        OF BLOC-2.
ADD MEMBRE-3       OF BLOC-1
TO MEMBRE-3        OF BLOC-2.
```

En revanche, si on opte pour le 3e format, on obtient le même résultat en écrivant seulement :

```
ADD CORRESPONDING BLOC-1
TO BLOC-2.
```

Malgré son avantage dans ce cas particulier, il vaut mieux éviter de définir des champs avec des noms de sous-ensembles semblables.
Dans un souci de clarté, il est, de loin, préférable d'affecter des noms différents aux champs. On saisit mieux et plus facilement le déroulement de la fonction logique (d'après son nom), de même que ses anomalies en cours de traitement, grâce au message CROSS REFERENCE envoyé par le compilateur.

DEUXIEME FORMAT DE L'INSTRUCTION ADD

```
ADD { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]
    { constante-1     }   { constante-2     }

GIVING nom-de-donnée-n [ROUNDED] [ON SIZE ERROR phrase impérative].
```

TROISIEME FORMAT DE L'INSTRUCTION ADD

```
ADD { CORRESPONDING } nom-de-donnée-1 TO nom-de-donnée-2
    { CORR          }

[ROUNDED] [ON SIZE ERROR phrase-impérative]
```

Le verbe SUBTRACT

Très proche de l'addition, cette instruction comporte également 3 formats. Elle permet de soustraire, du contenu d'un champ, celui d'un ou de plusieurs autres. Les clauses respectent les mêmes règles déjà décrites dans ADD.
Pour illustrer le premier format, considérons les champs :

```
01 VALEUR-INITIALE-1      PIC S9(4) COMP.
   VALUE 3500.
01 VALEUR INITIALE-2       PIC S9(4) COMP.
   VALUE 1000.
01 VALEUR-INITIALE-3       PIC S9(4) COMP.
   VALUE 0.
01 EN SOUSTRAYANT-1        PIC S9(3) COMP.
   VALUE 630.
01 EN SOUSTRAYANT-2        PIC S9(3) COMP.
   VALUE 280.
01 EN SOUSTRAYANT-3        PIC S9(3) COMP.
   VALUE 2.
```

L'instruction :

```
SUBTRACT 180
         SOUSTRAIRE-1
         SOUSTRAIRE-2
         SOUSTRAIRE-3

    FROM VALEUR-INITIALE-1
         VALEUR-INITIALE-2.
```

effectue les sommes de 180, procède à SOUSTRAIRE-1, SOUSTRAIRE-2, SOUSTRAIRE-3 et enfin soustrait le résultat du contenu des champs VALEUR-INITIALE-1 et VALEUR-INITIALE-2.
Sur la base des valeurs données plus haut, le résultat de l'opération sera :

```
VALEUR-INITIALE-1 =  2408
VALEUR-INITIALE-2 =   -92
```

Si l'on souhaite ne pas modifier le contenu du champ dans lequel les soustractions sont effectuées et obtenir le résultat dans un autre champ, il faut adopter le 2e format.

PREMIER FORMAT DE L'INSTRUCTION SUBTRACT

```
SUBTRACT { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]
         { constante-1     }   [ { constante-2     } ]

    FROM nom-de-donnée-m [ROUNDED]
         nom-de-donnée-n [ROUNDED]

    [ON SIZE ERROR phrase impérative]
```

DEUXIEME FORMAT DE L'INSTRUCTION SUBTRACT

```
SUBTRACT { nom-de-donnée-1 } [ { nom-de-donnée-2 } ]
         { constante-1     }   [ { constante-2     } ]

    FROM { nom-de-donnée-m }
         { constante-m     }

   GIVING nom-de-donnée-n [ROUNDED]

    [ON SIZE ERROR phrase-impérative]
```

En reprenant les champs de l'exemple précédent, on peut écrire :

```
SUBTRACT 180
         SOUSTRAIRE-1
         SOUSTRAIRE-2
         SOUSTRAIRE-3

    FROM  VALEUR INITIALE-1
    GIVING VALEUR INITIALE-3.
```

Le résultat de l'opération sera :

```
VALEUR INITIALE-1 = 3500
VALEUR INITIALE-3 = 2048
```

Le 3e format est, en tous points, équivalent à celui de ADD. Si on se réfère aux groupes de données BLOC-1 et BLOC-2 déjà décrits dans l'addition, l'instruction :

```
SUBTRACT CORR BLOC-1
FROM BLOC-2
```

équivaut aux instructions suivantes :

```
SUBTRACT MEMBRE-11 OF BLOC-1
FROM MEMBRE-11 OF BLOC-2.

SUBTRACT MEMBRE-12 OF BLOC-1
FROM MEMBRE-12 OF BLOC-2.

SUBTRACT MEMBRE-2 OF BLOC-1
FROM MEMBRE-2 OF BLOC-2.

SUBTRACT MEMBRE-3 OF BLOC-1
FROM MEMBRE-3 OF BLOC-2.
```

Prenons comme valeurs des différents sous-ensembles avant l'exécution de l'instruction :

	BLOC-1	BLOC-2
MEMBRE-11	210	500
MEMBRE-12	31	63
MEMBRE-2	4780	1200
MEMBRE-3	370	25612

Les résultats seraient :

	BLOC-1	BLOC-2
MEMBRE-11	210	290
MEMBRE-12	31	32
MEMBRE-2	4780	-3580
MEMBRE-3	370	25242

Le verbe MULTIPLY

Dans le format le plus usité (voir page suivante), le résultat de la multiplicande (nom-de-champ-1 ou constante-1) et le multiplicateur (nom-de-champ-2 ou constante-2) est rangé dans le champ non-de-donnée-3.

Le verbe DIVIDE

Le premier format de la division est décrit dans la page suivante. Il agit comme MULTIPLY. Le quotient de la division, entre le dividende (nom-de-donnée 1 ou constante-1) et le diviseur (nom-de-donnée-2 ou constante-2), est rangé dans un champ d'arrivée (nom-de-donnée-3). Ce quotient est calculé dans un registre spécial de l'unité centrale avec un nombre constant de chiffres décimaux. De ce fait, si le résultat demandé est rond, l'opération, destinée à arrondir le quotient, se produit après le calcul proprement dit (avec un nombre fixe de chiffres décimaux).
N'oublions pas que la division d'un nombre par zéro donne lieu à un dépassement de capacité,

TROISIEME FORMAT DE L'INSTRUCTION SUBTRACT

```
SUBTRACT { CORRESPONDING } nom-de-donnée-1 FROM nom-de-donnée-2
         { CORR          }

[ROUNDED] [ON SIZE ERROR] phrase impérative]
```

FORMAT DE L'INSTRUCTION MULTIPLY

```
MULTIPLY { nom-de-donnée-1 }      BY { nom-de-donnée-2 }
         { constante-1     }         { constante-2     }
    GIVING nom-de-donnée-3 [ROUNDED]
        [ON SIZE ERROR phrase-impérative]
```

FORMAT DE L'INSTRUCTION DIVIDE

```
DIVIDE { nom-de-donnée-1 }      BY { nom-de-donnée-2 }
       { constante-1     }         { constante-2     }
    GIVING nom-de-donnée-3 [ROUNDED]
        [ON SIZE ERROR phrase-impérative]
```

géré par le programmeur avec l'introduction de la clause ON SIZE ERROR.
Il est superflu de noter que le champ qui recueille le résultat devra être décrit avec exactitude.
Si le résultat de l'opération est un nombre décimal, alors que le champ d'arrivée est défini en entier, il y aura perte de toutes les décimales.

Ainsi :

DIVIDE 10 BY 4 GIVING QUOTIENT

donnera comme résultat :

QUOTIENT = 2,5

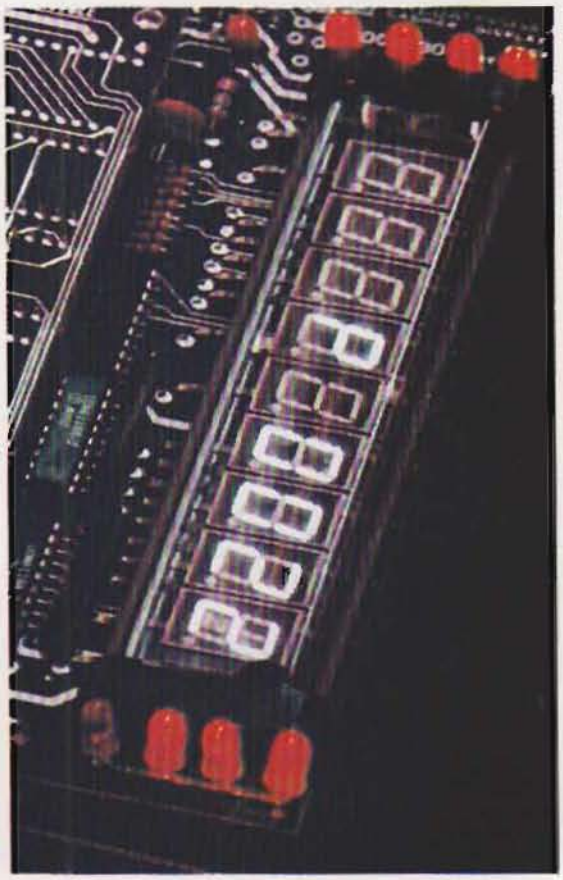
si le champ d'arrivée est décrit par

01 QUOTIENT PIC S9V9.

Le résultat sera 2 si le quotient est décrit par :

01 QUOTIENT PIC S9

Dans certaines applications, on peut trouver très utile d'avoir, pour résultat, les deux parties entière et décimale du quotient. Cela correspond au 2e format de DIVIDE.



Litton BEI

Affichage LED (Light Emission Diode) appliqué à un convertisseur analogique numérique.

DEUXIEME FORMAT DE L'INSTRUCTION DIVIDE

```
DIVIDE { nom-de-donnée-1 }  BY { nom-de-donnée-2 }
       { constante-1     }     { constante-2     }
    GIVING nom-de-donnée-3 [ROUNDED]
    REMAINDER nom-de-donnée 4
        [ON SIZE ERROR phrase-impérative]
```

A titre d'exemple, supposons que l'on veuille savoir si le nombre contenu dans le champ NOMBRE est pair ou impair. On sait que tous les nombres pairs sont divisibles par 2 avec un reste égal à 0, ce qui facilite l'opération de vérification :

```
DIVIDE NOMBRE BY 2
       GIVING QUOTIENT
       REMAINDER RESTE.

IF RESTE = 0
       DISPLAY « NOMBRE PAIR »
           UPON CONSOLE

ELSE
       DISPLAY « NOMBRE IMPAIR »
           UPON CONSOLE.
```

Verbes de transfert et de manipulation des données

Une des instructions les plus usitées en Cobol est MOVE. Elle procède au transfert des données d'un champ vers un autre parmi ceux décrits dans la division données du programme. Compte tenu de la signification explicite du verbe MOVE (déplacer), cette instruction a déjà été utilisée dans certaines exemples dans son format le plus simple.
Avant de l'examiner en détail, il convient de souligner que le compilateur détermine le type de transfert à effectuer d'après les caractéristiques du champ d'arrivée. En d'autres termes, selon qu'il est alphanumérique ou numérique, l'instruction MOVE sera également alphanumérique ou numérique. Il existe un 3e type de transfert dit **MOVE par état** destiné au transfert de données vers des champs dont les PICTURE renferment des caractères d'impression **(symboles d'édition)**. Ces derniers permettent d'insérer ou de masquer certains caractères du contenu du champ en fonction d'exigences esthétiques, fonctionnelles ou tout simplement de sécurité.
Considérons, par exemple, le cas où le champ TOTAL décrit par

```
01 TOTAL      PIC 9(5)V9(4).
```

doit être envoyé sur une ligne d'impression. Sachant que le caractère V ne représente qu'une position virtuelle de la virgule des décimales, si le champ en impression avait la même PICTURE, il ne serait pas en mesure d'établir la valeur réelle de TOTAL.
En effet, si cette valeur était 45 781,2478 (notation européenne) on aurait à l'impression :

```
457812478
```

En transférant TOTAL dans le champ TOTAL-IMPRESSION :

```
01 TOTAL-IMPRESSION      PIC 9(5).9(4).
```

on obtient la représentation réelle du nombre contenu dans TOTAL.
Le caractère . (point) est donc un caractère de préparation à l'impression. Notons au passage que, contrairement au symbole V, le point (la virgule pour nous) occupe une adresse mémoire, ce qui fait, pour TOTAL-IMPRESSION, 10 caractères.

Le verbe MOVE

L'instruction MOVE a deux formats valables tant pour les MOVE alphanumériques que pour

FORMAT DE L'INSTRUCTION MOVE

```
MOVE { nom-de-donnée-1 } TO nom-de-donnée-2 [nom de donnée-3] . . . . . .
     { constante       }
```

SECOND FORMAT DE L'INSTRUCTION MOVE

```
MOVE { CORRESPONDING } nom-de-donnée-1 TO nom-de-donnée-2.
     { CORR          }
```

les MOVE numériques ou par état. La nature de l'instruction est déterminée à partir de la description des champs d'arrivée.
D'une façon générale, MOVE transfère le contenu de nom-de-donnée-1 ou d'une constante vers un ou plusieurs champs d'arrivée.
Soit le champ :

```
01 DEPART      PIC X(20).
```

On désire transférer son contenu vers :

```
01 ARRIVEE-1   PIC X(20).
01 ARRIVEE-2   PIC X(20).
01 ARRIVEE-3   PIC X(20).
```

On a le choix entre 2 formulations équivalentes :

```
MOVE DEPART TO ARRIVEE-1.
MOVE DEPART TO ARRIVEE-2.
MOVE DEPART TO ARRIVEE-3.
```

ou bien

```
MOVE DEPART TO ARRIVEE-1
               ARRIVEE-2
               ARRIVEE-3.
```

La forme contractée est conseillée car elle rend plus évidente l'opération de MOVE simultanée du même champ vers plusieurs champs d'arrivée.

Dans le format :

```
MOVE constante TO  nom-de-donnée-2
                   nom-de-donnée-3
                   . . . . . .
```

« constante » doit être alphanumérique ou numérique selon le champ d'arrivée. Sur la base de ce qui a été dit à propos des MOVE multiples, il est évident que la compatibilité doit être respectée pour tout champ d'arrivée.

L'exemple suivant :

```
WORKING-STORAGE SECTION.
01 ARRIVEE-1                PIC X(20).
01 ARRIVEE-2.
   05 CHAMP-1               PIC 9(10).
   05 FILLER                PIC X(10).
PROCEDURE DIVISION.
DEBUT.
MOVE
'DEBUT TRAITEMENT' TO  ARRIVEE-1.
                       CHAMP-1.
```

est erroné, dans la mesure où la constante alphabétique DEBUT TRAITEMENT ne peut être déplacée dans un domaine numérique tel que CHAMP-1. En revanche il est correct de la déplacer vers ARRIVEE-1. Le transfert des données vers un ou plusieurs champs ne détruit pas le contenu du champ départ qui reste inchangé jusqu'à ce qu'on le modifie volontairement. En général, les champs d'arrivée et de départ peuvent être indifféremment élémentaires ou composés.
Rappelons à ce sujet qu'une donnée composée est prise dans son ensemble comme alphanumérique, indépendemment des descriptions des champs composants.

Analysons maintenant l'exemple suivant :

```
DATA DIVISION.
FILE SECTION.
FD FICHES
   LABEL RECORD OMITTED.
01 FICHE                          PIC X(3).
   05 TYPE-FICHE                  PIC X(3).
   05 CLASSEMENT NOMINATIF.
      10 PRENOM                   PIC X(10).
      10 NOM                      PIC X(15).
   05 DATE-DE-NAISSANCE.
      10 JOUR                     PIC 9(2).
      10 MOIS                     PIC 9(2).
      10 ANNEE                    PIC 9(2).
   05 SEXE                          PIC X.
   05 FILLER                     PIC X(45).
*
*
WORKING-STORAGE SECTION.
77 BAC-A-FICHE                    PIC X(80).
77 BAC-A-NOM                      PIC X(25).
01. . . . . .
   . . . . . .
*
*
PROCEDURE DIVISION
DEBUT.
   . . . . . .
   . . . . . .
*
*** EXEMPLES DE TRANSFERT
*
*
MOVE SPACES TO          TO BAC-A-FICHE.
MOVE FICHE              TO BAC-A-FICHE
MOVE BAC-A-NOM                TO NOM.
MOVE 'ABC'              TO FICHE-TYPE.
MOVE 83                       TO ANNEE.
   . . . . . .
   . . . . . .
```

Comme on peut le remarquer :

MOVE SPACES TO BAC-A-FICHE

déplace la constante figurative SPACES dans un champ élémentaire.

MOVE FICHE TO BAC-A-FICHE

déplace le champ composé FICHE dans un champ élémentaire.

MOVE BAC-A-NOM TO NOM

déplace un champ élémentaire dans un composé,

MOVE 'ABC' TO TYPE-FICHE

transfère la constante alphanumérique ABC dans un champ élémentaire.

MOVE 83 TO ANNEE

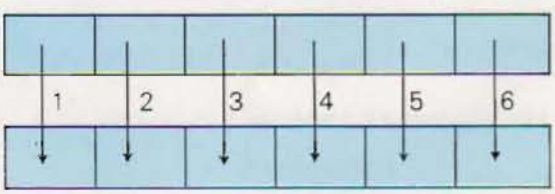
situe la constante numérique 83 dans le champ ANNEE.

Pour analyser en détail l'instruction MOVE, il est nécessaire d'apporter une précision importante quant aux constantes figuratives. Il s'agit de :

SPACE	(SPACES)
ZERO	(ZEROS ou ZEROES)
LOW-VALUE	(LOW-VALUES)
HIGH-VALUE	(HIGH-VALUES)
QUOTE	(QUOTES)
ALL	

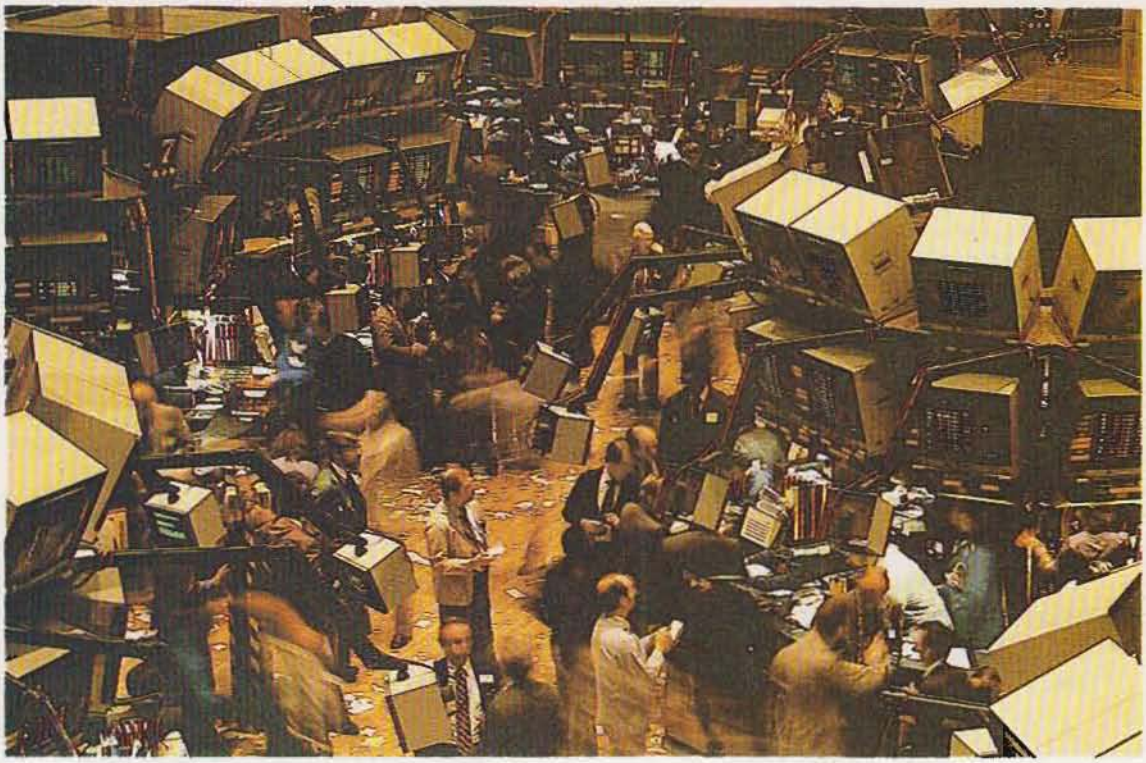
SPACE (ou SPACES) est considéré comme alphabétique, ZERO (ou ZEROS, OU ZEROES) comme numérique, alors que LOW-VALUE et HIGH-VALUE sont assimilés à des données alphanumérique.

MOVE alphanumérique. Il effectue le transfert des données dans la séquence indiquée par le schéma suivant :



Le champ d'arrivée est rempli de gauche à droite en copiant le contenu du champ de départ, caractère par caractère.

Si la longueur du champ départ est inférieure à celle de l'arrivée, il est comblé avec des blancs.



A la Bourse, les cambistes sont équipés de terminaux.

Par exemple :

```
01 DEPART      PIC X (3) VALUE 'ABC
01 ARRIVEE     PIC X (6) VALUE 'UVXYW'.
PROCEDURE DIVISION.
A.
      MOVE DEPART TO ARRIVEE.
```

Avant l'instruction MOVE, la situation était :

DEPART | A | B | C |

ARRIVEE | U | V | Z | X | Y | W |

Immédiatement après, on a :

DEPART | A | B | C |

ARRIVEE | A | B | C | | | |

Comme on peut le constater, le transfert de caractères a provoqué le remplissage du champ avec des blancs sur la droite.
Pour inverser cette modalité de transfert (c'est-à-dire remplir le champ ARRIVEE avec le contenu de DEPART et par conséquent être amené à combler à gauche avec des blancs) il faut insérer, dans la propre description du champ d'arrivée, la clause JUSTIFIED RIGHT (JUST).

Toujours en utilisant les champs de l'exemple précédent, on a :

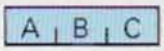
```
01 DEPART      PIC X (3) VALUE 'ABC'.
01 ARRIVEE     PIC X (6) JUST VALUE
               'UVZXYW'.
PROCEDURE DIVISION.
A.
      MOVE DEPART TO ARRIVEE.
```

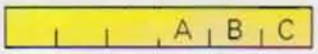
Situation avant MOVE

DEPART | A | B | C |

ARRIVEE | U | V | Z | X | Y | W |

Situation après MOVE :

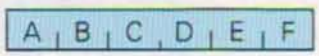
DEPART 

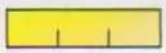
ARRIVEE 

Considérons maintenant le cas où le champ départ aurait une longueur supérieure à celle du champ d'arrivée :

```
01 DEPART     PIC X (6) VALUE 'ABCDEF'.
01 ARRIVEE    PIC X (3) VALUE SPACES.
```

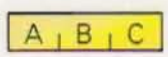
La situation précédant l'exécution de l'instruction MOVE DEPART TO ARRIVEE est alors :

DEPART 

ARRIVEE 

Situation après :

DEPART 

ARRIVEE 

Ainsi, tous les caractères en trop situés à droite du champ ARRIVEE ont été tronqués.

Inversement, si la définition du champ ARRIVEE avait été :

```
01 ARRIVEE PIC X (3) JUST VALUE SPACES.
```

La situation finale aurait été :

DEPART 

ARRIVEE 

En d'autres termes, la réduction des caractères en trop se serait produite à gauche, après le remplissage (à partir de la droite) de l'espace disponible. Le tableau ci-dessous présente une synthèse des opérations MOVE de type alphanumérique décrites jusqu'ici.
On y constate qu'un champ composé est en mesure d'accueillir des données en provenance de tout autre type de champ.
Exception à cette règle générale : les nombres en virgule flottante (USAGE COMP-1 et COMP-2). Cette exception n'apparaît pas dans le tableau ; il convient donc d'illustrer le cas à l'aide d'un exemple.

Considérons deux champs :

```
01 NOMBRE                    COMP-1.
01 CHAMP.
   05 SOUS-CHAMP-1           PIC 9 (4).
   05 SOUS-CHAMP-2           PIC X (3).
   05 FILLER                 PIC X.
```

CHAMP-DEPART	CHAMP-ARRIVEE			
Type	Composé	Elément alphanumérique	Elément alphabétique	Elément numérique
Composé	oui	oui	oui	NON
Elément alphanulérique	oui	oui	oui	NON
Elément alphabétique	oui	oui	oui	NON
Elément numérique	oui	oui*	NON	oui

*Seulement si le champ de départ est entier.

L'instruction

```
MOVE NOMBRE TO CHAMP
```

est erronée dans la mesure où le champ NOMBRE a un codage interne en virgule flottante en simple précision. On observera, en outre, que l'astérisque apparaissant sur le tableau indique que le transfert d'un champ élémentaire numérique vers un champ alphanumérique n'est possible que si celui de DEPART est entier.
L'instruction USAGE de ce champ peut être indifféremment COMP ou DISPLAY ; cependant, même alors il reste impossible d'utiliser un champ en virgule flottante comme départ.
Au cours du transfert d'un champ numérique vers un champ alphanumérique, la conversion se passe dans le champ ARRIVEE, selon les modalités décrites pour ce type de champ. L'exemple suivant permettra de mieux comprendre cette règle.
Considérons le champ :

```
01 ARRIVEE          PIC X (3).
```

et les instructions :

```
MOVE 38 TO ARRIVEE.
MOVE 7425 TO ARRIVEE.
```

Après le premier MOVE, le contenu d'arrivée sera :

3	8	

alors qu'après le deuxième MOVE, le champ aura la configuration suivante :

7	4	2

Remarquons que, si l'on souhaite vérifier, par exemple, que le champ ARRIVEE contient le nombre 38, cette opération devra être réalisée sur les trois caractères du champ.
En dépit de son apparence correcte, la séquence d'instructions :

```
MOVE 38 TO ARRIVEE.
IF ARRIVEE = '38'
   PERFORM EGAL-38.
```

n'aboutira pas à l'exécution de la SECTION

J. Pickerell/Marka

Test des plaques du circuit de commande pour disques souples.

EGAL-38. En effet, après le déroulement de l'instruction MOVE 38 TO ARRIVEE, le contenu du champ ARRIVEE est la configuration de caractères

3	8	b

et non pas 38 ; la forme correcte de l'exemple précédent est :

```
MOVE 38 TO ARRIVEE.
IF ARRIVEE = '38'
   PERFORM EGAL-38.
```

Notons par ailleurs que ce qui vient d'être exposé demeure valable quand le champ ARRIVEE est justifié à droite (JUST RIGHT) ; en d'autres termes, ' 38' n'est pas égal à '38'.

La clause ALL. Si on doit remplir tout un champ avec une série de caractères, il existe un format de l'instruction MOVE :

MOVE ALL caractères TO nom-champ.

Le diagnostic assisté par ordinateur (3)

L'échographie est une méthode de diagnostic fondée sur la production d'images des détails anatomiques au moyen de sondes spéciales qui émettent des ultrasons. Les échos émis par les différentes structures anatomiques sont traités par un microprocesseur et projetés sur un écran. L'image visualisée reproduit fidèlement les caractéristiques morphologiques et structurelles de la partie ou de l'organe examiné.

Les caractéristiques particulières qui différencient l'échographie des autres méthodes d'investigation la rendent pratiquement irremplaçable, surtout du fait de son inocuité absolue. Comme elle ne fait pas appel aux radiations ionisantes, on peut la répéter indéfiniment sans aucun risque. Elle fit son apparition aux alentours des années 70 mais ce n'est que vers la moitié de la dernière décennie que ce procédé se développa réellement en Europe et aux Etats-Unis, pour aboutir à nos installations actuelles. Celles-ci sont capables de traiter en ***temps réel*** *des images d'organes en mouvement. Autre avantage : son coût bien moindre en comparaison des autres systèmes. Enfin, ses dimensions réduites et sa facilité d'emploi le font largement apprécier.*

Les unités échographiques multidisciplinaires du type SDU 3000 sont aujourd'hui dotées d'une vaste gamme de transducteurs destinés à des examens statiques et dynamiques en pédiatrie, en ophtalmologie, en obstétrique, et à l'examen des organes profonds ou superficiels en général.

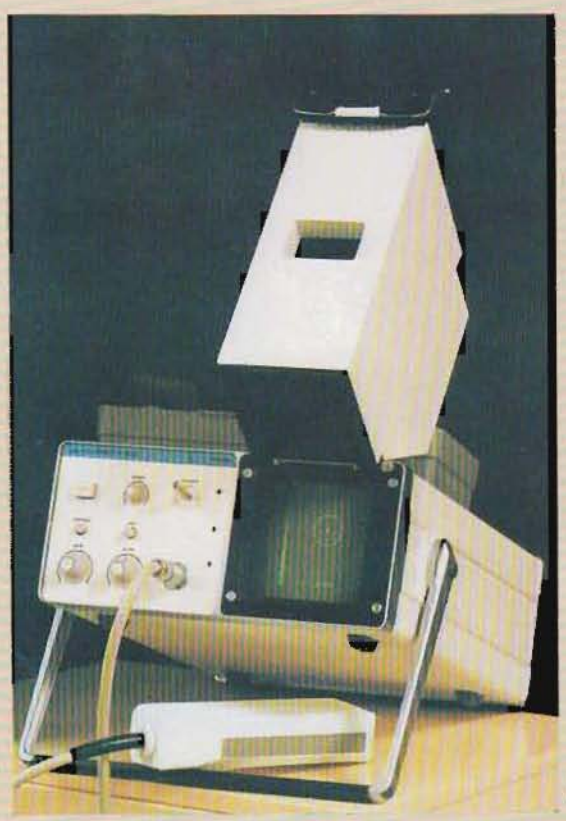
Il existe fondamentalement deux types de sondes pour l'investigation dynamique : la sonde linéaire à double focale (géométrique et électronique) pour des examens panoramiques de l'abdomen et pour l'obstétrique et la sonde à secteur qui permet d'éliminer les difficultés d'accès à certains organes.

Les systèmes multidisciplinaires de base sont dotés d'une vidéo incorporée, d'objectifs pour la production de documents imprimés, de sondes bioptiques et de perforateurs pour l'enregistrement des données sur papier. Le tout est intégré dans un petit meuble de la taille d'un réfrigérateur, sur roulettes, facile à transporter. Ces appareils présentent en outre des caractéristiques de résolution et de pénétration qui paraissaient utopiques au regard des imposantes installations que l'on construisait il y a moins d'une décennie. Une sonde sectorielle a une pénétration pouvant atteindre 30 cm et une résolutation angulaire d'à peine 12 pouces (soit 30 cm).

Malgré tous ces résultats déjà brillants, il ne faudrait pas croire que la ***miniaturisation*** *aurait déjà atteint ses limites. Le système Sono Diagnost R 1000 en est la démonstration la plus évidente : il pèse moins de 12 kg, caméra comprise (photo ci-dessous). L'écran de contrôle fait apparaître des images 10 × 8 cm produites par le traitement des sondages prélevés 25 fois à la seconde par une sonde linéaire qui émet un faisceau* ***d'ultrasons*** *à 2,2-3 MHz. Cette installation peut fonctionner en pleine campagne en se branchant sur une batterie de voiture. Leur prix et leur simplicité d'emploi la met à la portée de tous les cabinets médicaux.*

Les techniques de sondage aux rayons X et celles qui utilisent les ultrasons donnent une

Le Sono Diagnost R 1000, équipement portable pour échographies.



Philips

Philips

Appareil d'échographie transportable : le Sono Diagnost Universal 3000 (SDU 3000).

image morphologique des organes examinés se limitant à la reproduction, certes précise, d'un détail anatomique mais sans pour autant fournir la moindre information sur son état fonctionnel.

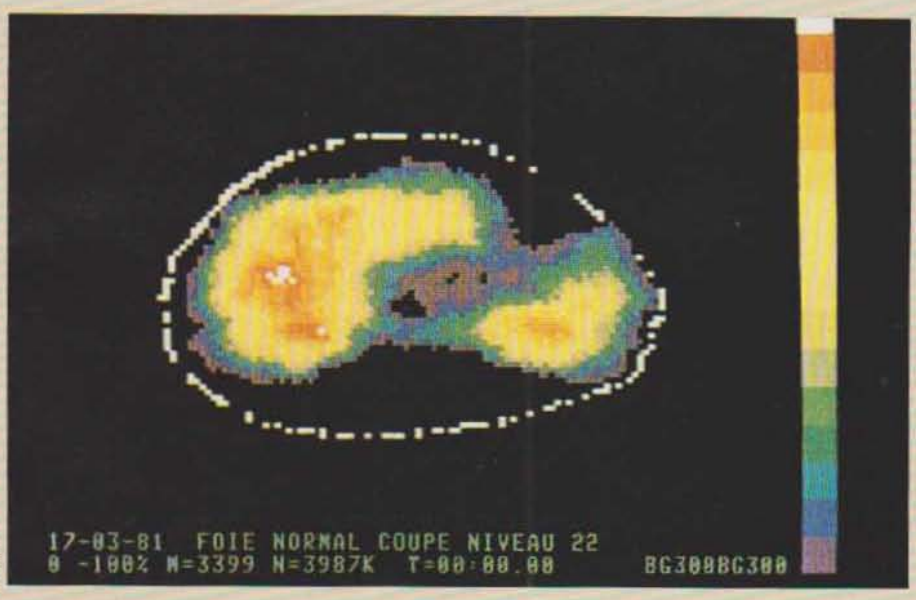
C'est justement pour répondre à cette exigence qu'est née la technique de diagnostic utilisant la radioactivité, c'est-à-dire les isotopes radioactifs de certains éléments naturellement présents dans le corps humain. (Attention ! Ceci n'a rien à voir avec la résonance magnétique nucléaire, la RMN, dont il a été question précédemment). Le but principal des ***isotopes radioactifs*** *est de permettre l'étude de la fonction physiologique des organes sans se borner à la production d'une image anatomique statique.*

Les techniques utilisant les isotopes radioactifs nécessitent l'injection, dans l'organisme, de produits émettant un ***rayonnement gamma*** *(c'est le cas des produits radioactifs) et qui, de plus, sont métabolisés dans l'organe ou dans le système anatomique à examiner. On procède ensuite au relevé et à la mesure d'intensité des radiations, ce qui permet de suivre le processus de métabolisation et d'établir ainsi une carte des concentrations du produit traçant dans les différents organes.*

Dans ce secteur, les nouveautés doivent beaucoup aux possibilités offertes par ***l'automatisation*** *des appareils. Depuis vingt-cinq ans, l'instrument de base d'un laboratoire de médecine nucléaire est la gamma-caméra, mais c'est seulement au cours des cinq dernières années que l'introduction du microprocesseur, pour piloter l'exploration, a transformé cet instrument en un puissant système de diagnostic, grâce auquel on peut relever jusqu'aux images tomographiques axiales.*

Dans le système Gamma Diagnost Tomo, grâce auquel on obtient l'image complète du corps par la tomographie axiale digitalisée, le révélateur gamma proprement dit est suspendu à un bras articulé. Il est, en outre, muni d'un verre à scintillation de 40 cm de diamètre et de 9 mm d'épaisseur, derrière lequel se trouve une batterie de 61 photo-amplificateurs transformant les signaux lumineux (produits dans le verre par la révélation des rayons gamma), en signaux électriques qui sont traités pour fournir l'image désirée.

Philips



Philips

Ci-dessus : positionnement du révélateur du « Gamma Diagnost Tomo ».
Ci-contre : image tomographique digitalisée de la coupe d'un thorax (l'échelle de concentration du traçant est représenté à droite de l'image).

Ceux-ci sont, à leur tour, traités pour fournir l'image désirée. La vitesse d'acquisition des données est de 40 images à la seconde, avec possibilité de coupler deux Gamma-caméras. La console de contrôle et de traitement des données est munie d'un moniteur d'une résolution de 512 × 512 pixels, (extensible à 640 × 256 pour le corps entier) et 256 niveaux de couleurs sont disponibles. Le système de calcul comporte 2 microprocesseurs spécialisés fonctionnant en parallèle. Ils sont dotés de 400 à 528 Koctets de mémoire vive, programmables en Fortran, en RTL 2, en Assembleur et en PMCL (un langage dédié aux applications biomédicales). Le système est complété par un disque dur de 24 Moctets et une disquette de 1,2 Moctets.

Le calculateur rend donc possible la construction de ***l'image tomographique sur un écran couleur.*** *La possibilité de préciser les contours de la section examinée permet également de corriger les données en tenant compte de l'absorption des rayons Gamma par les tissus. Les images antérieures peuvent être rappelées simultanément pour, par exemple, vérifier d'une manière simple l'évolution du processus de concentration de la* ***substance traçante*** *dans un organe. En outre, on peut réaliser des copies d'images sur papier ou sur transparent.*

De tels systèmes ont profondément contribué à éliminer les risques liés aux procédures classiques, en particulier dans le domaine du diagnostic des affections pulmonaires rénales, cardiaques, du foie, du pancréas, du système osseux et du système endocrinien.

Philips

Tableau de commande et de contrôle d'un thermographe à image digitalisée. A droite : manettes de positionnement et curseurs de délimitation du champ d'observation.

D'un usage aujourd'hui courant, la thermographie est pourtant une technique d'investigation relativement récente. Les thermographes de première génération utilisaient des plaques photographiques sensibles à l'infrarouge directement exposées aux radiations corporelles du patient. On exploitait ainsi la propriété, commune à de nombreux phénomènes inflammatoires et pathologiques, d'altérer localement la température du corps. L'application correcte de cette technique nécessitait, en ce temps-là, la production de plaques photographiques ***hautement sensibles,*** *même à des variations de quelques fractions de degré.*

Aujourd'hui, le procédé de balayage a recours à des cellules photovoltaïques sensibles à l'infra-rouge et à des circuits de traitement numérique des signaux émis par elles. Des images en fausses couleurs (c.a.d. en couleurs symboliques) sur grand écran, qu'on peut également obtenir en sortie d'imprimante ou stocker sous forme numérique dans une mémoire de masse, remplacent les anciennes photographies.

Le document ci-dessus reproduit le tableau de contrôle d'un ***thermographe à affichage numérique.*** *Le tube thermographique est doté d'une cellule photovoltaïque à indio-antimoine baignant dans de l'azote liquide. Des moteurs électriques contrôlent les mouvements transversaux (+ 25°), latéraux (+ 100°) et de niveau (50 ÷ 180 cm).*

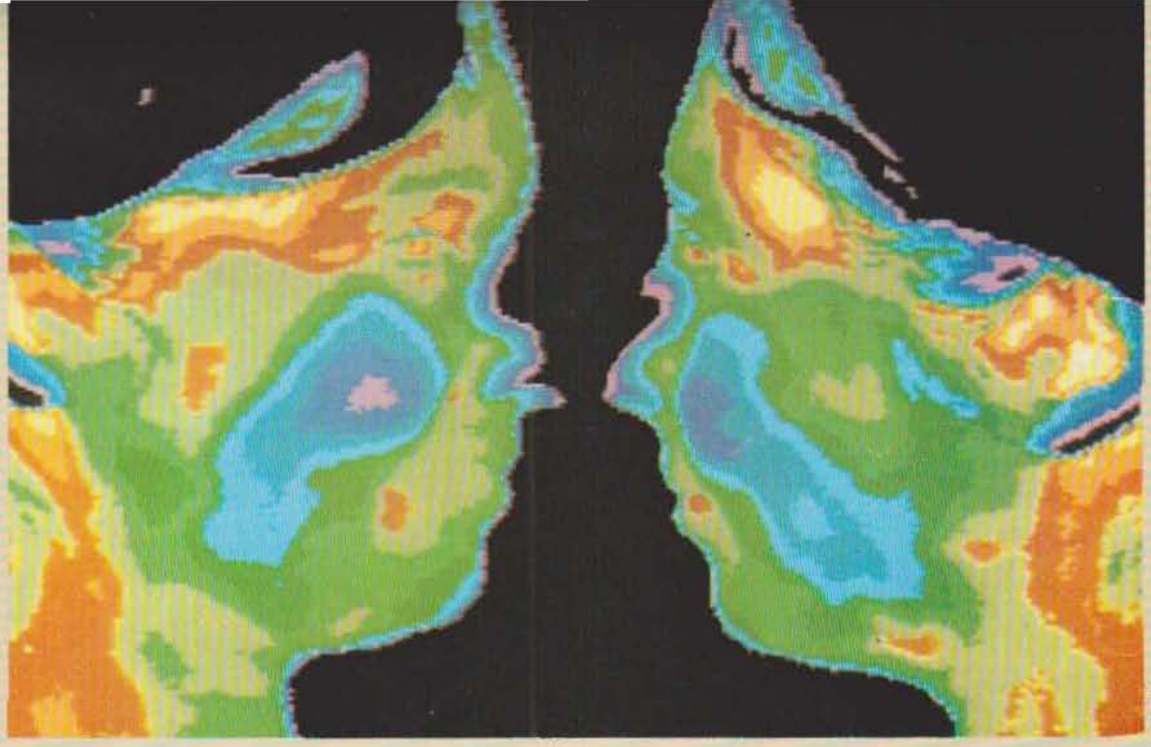
Grâce à sa fine résolution, la cellule permet de distinguer des détails sous un angle inférieur à 0,002 radians, alors qu'en résolution thermique la température varie seulement de 0,08°C. Le temps d'exposition nécessaire, pour obtenir une réponse complète, est d'une seconde.

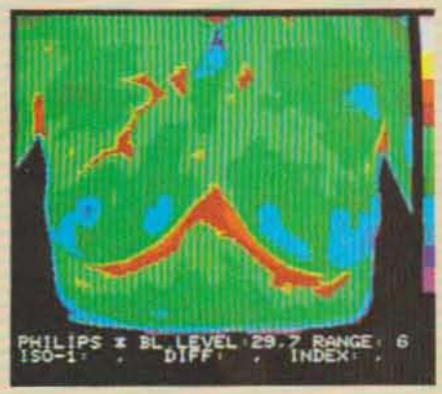
Les signaux émis sont enregistrés par le système qui élabore une matrice numérique du champ d'investigation qui est envoyée sur un moniteur à dix couleurs de 30 cm. Sur ce dernier, une matrice de 312 × 240 pixels, bien en évidence, complète par une échelle des couleurs certaines indications, permettant de repérer avec précision l'échelle des températures.

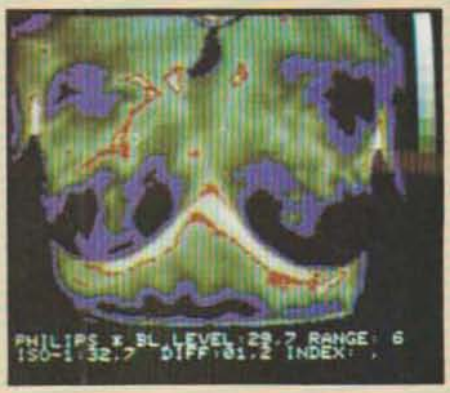
La température du seuil est réglée entre 15 et 40°C. Toutes les données indiquant que la température est inférieure à ce niveau sont visualisées en noir, d'où son nom de « black level ». L'étendue de la zone au-dessus du black level (range) est, quant à elle, réglée aux valeurs, 2, 3, 4, 5, 8, 10, 15°C.

Une fois visualisée, l'image thermographique est ***archivée*** *et, si besoin,* ***rappelée*** *sur une partie de l'écran. Ce qui permet de mettre en évidence simultanément des zones anatomiques particulières qui ne pourraient entrer en même temps dans le champ visuel de la caméra, comme par exemple les deux côtés d'une tête ou les deux côtés d'une main (photos de la page 1035).*

Il existe plusieurs manières de numériser l'image avant de la transcrire en fausses couleurs. Grâce à la première fonction du circuit électronique, on met en évidence deux isothermes d'une amplitude égale à 3% de la gamme sur deux couleurs. Une fois le niveau de la première isotherme choisi (ISO-1), celui de la seconde isotherme peut varier à volonté dans la gamme supérieure ; la différence de niveau est alors automatiquement affichée à l'écran (DIFF). Des curseurs du tableau de







Après calculs, l'image thermographique d'une comparaison opérée entre les deux côtés du visage est visualisée sur écran vidéo et, le cas échéant, enregistrée sur support magnétique. Ci-contre : deux représentations d'un thorax féminin et, dessous, la comparaison thermographique des mains d'un patient. La vascularisation réduite de la main gauche peut être quantifiée par l'intermédiaire du profil thermique visible à droite de l'écran.

contrôle permettent de délimiter, sur le moniteur, une fenêtre sur une région avec calculs en ***temps réel*** *de l'indice thermographique (INDEX). Mais surtout, l'ordinateur est en mesure de mettre en évidence le profil thermique d'une ligne horizontale sélectionnée par l'opérateur sur le moniteur (photo ci-dessus). La courbe de température est établie d'après les signaux thermiques traités en fonction du nombre de pixels « allumés ».*

Outre la traditionnelle application au ***diagnostic du cancer*** *du sein, la possibilité d'obtenir des thermogrammes, en temps réel dans un intervalle d'une seconde, permet accessoirement de quantifier les variations thermiques associées à l'administration de remèdes.*

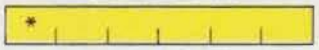
Soit le champ :

```
01 ARRIVEE          PIC X(6).
```

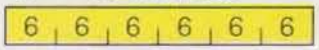
Dans les exemples suivants, on a mis en évidence les zones ARRIVEE après chacun des deux transferts (MOVE ALL et MOVE simple) :

MOVE ALL ' * ' TO ARRIVEE

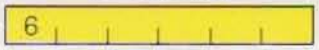
MOVE ' * ' TO ARRIVEE



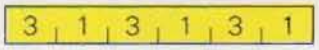
MOVE ALL ' 6 ' TO ARRIVEE



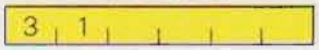
MOVE ' * ' TO ARRIVEE



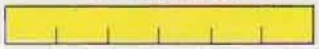
MOVE ALL ' 31 ' TO ARRIVEE



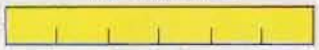
MOVE ' 31 ' TO ARRIVEE



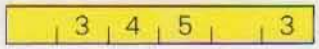
MOVE ALL ' ' TO ARRIVEE



MOVE ' ' TO ARRIVEE



MOVE ALL ' 345 TO ARRIVEE



La dernière instruction montre comment le compilateur respecte la répétition de la rangée de caractères indiquée jusqu'au remplissage du champ, tout en retranchant les caractères en trop.
De plus, les espaces sont traités comme des caractères à part entière, quelle que soit leur position.

MOVE numérique. L'opération de transfert des données nécessite, comme on l'a vu précédemment, une bonne connaissance des mécanismes internes. La validité des résultats est donc directement liée à l'utilisation correcte des instructions et au dimensionnement des champs. Ainsi, le MOVE alphanumérique ne se préoccupe pas de la signification des données, alors que le MOVE numérique tient compte de l'USAGE du champ d'arrivée et d'autres facteurs comme le signe et la position du point décimal.
Le transfert de type numérique est effectué en 5 étapes :

1/ Positionnement du point décimal
2/ Remplissage du champ ARRIVEE
3/ Troncature des chiffres excédentaires ou complémentation du champ d'arrivée,
4/ Conversion du format de départ en format d'arrivée
5/ Traitement du signe.

1/ Positionnement du point décimal.
Pour transférer un nombre décimal, les chiffres entiers devront être correctement positionnés à gauche et les décimaux à droite du point.
Dans le transfert suivant :

NOM-DEPART vers NOM-ARRIVEE, où :

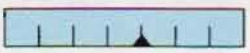
```
01 NOM-DEPART       PIC S9(4)V9(3).
01 NOM-ARRIVEE      PIC S9(6)V9(3).
```

L'instruction :

```
MOVE NOM-DEPART TO NOM-ARRIVEE.
```

effectue le positionnement de la façon suivante :

NOM-DEPART



NOM-ARRIVEE

(A noter que le symbole ▲ représente la position virtuelle en mémoire du point décimal).
C'est une opération qui est réalisée même quand la PICTURE des champs départ et arrivée ne comporte par le symbole V. Pour le compilateur, les descriptions :

```
01 NOMBRE-1         PIC S9(4).
01 NOMBRE-2         PIC SP(3)9(5).
01 NOMBRE-3         PIC S99P(5).
```

sont équivalentes aux suivantes :

```
01 NOMBRE-1     PIC S9(4)V.
01 NOMBRE-2     PIC SVP(3)9(5).
01 NOMBRE-3     PIC S99P(5)V.
```

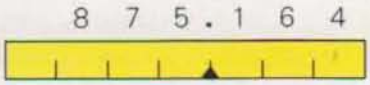
Les modalités de positionnement déjà décrites sont donc applicables aux champs de l'exemple. De plus, dans une constante numérique, le point décimal est assimilé à une position virtuelle.
Par exemple, si le champ ARRIVEE est défini par :

```
01 ARRIVEE      PIC S9(4)V9(3).
```

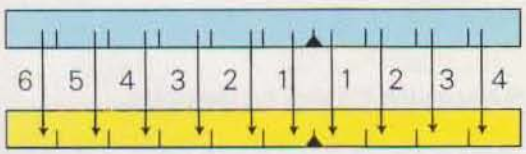
L'instruction :

```
MOVE 875.164 TO ARRIVEE.
```

comporte le positionnement du schéma suivant :



2/ Remplissage du champ d'arrivée. Après avoir positionné le point décimal des deux champs, la partie entière (arrivée) est remplie de la droite vers la gauche selon la séquence :



3/ Troncature ou complémentation du champ d'arrivée. Comme pour le MOVE alphanumérique, quand les champs n'ont pas la même longueur, on procède à la vérification du remplissage jusqu'aux limites du champ. De la même manière, on retranchera les chiffres excédentaires.

Soient les champs :

J. Pickerell/Marka

Tableau de contrôle de l'ordinateur Univac 1100/8.

```
01 NOM-DEPART      PIC S9(3)V9
                   VALUE 436.2.

01 NOM-ARRIVEE     PIC S9(6)V9(3).
                   VALUE 123456.789.
```

Voici la configuration des deux champs avant et après l'exécution de :

```
MOVE NOM-DEPART TO NOM-ARRIVEE
```

NOM-DEPART	**NOM-ARRIVEE**
Avant :	
4 3 6 2	1 2 3 4 5 6 7 8 9
Après :	
4 3 6 2	1 2 3 4 5 6 7 8 9

Puisque, dans le cas que nous venons d'examiner, nous avons une longueur de champ départ inférieure à celle du champ d'arrivée, on doit remplir les espaces libres avec des zéros non significatifs, à gauche et à droite des chiffres transférés.
Dans le cas contraire, il sera procédé à une troncature de la valeur transférée, aussi bien pour les entiers que pour les décimaux.
Pour clarifier le concept, considérons les champs :

```
01 NOM-DEPART          PIC S9(4)V9(2)
                       VALUE 1436.78.

01 ARRIVEE.
   05 NOM-ARRIVEE-1    PIC S9(3)V9(2).
   05 NOM-ARRIVEE-2    PIC S9(3)V9(4).
   05 NOM-ARRIVEE-3    PIC S9(4).
   05 NOM-ARRIVEE-4    PIC SV9.
   05 NOM-ARRIVEE-5    PIC S9(6).
```

On voit, à travers ces exemples, comment une opération de transfert vers un champ d'arrivée qui n'a pas été correctement dimensionné conduit à des résultats erronés.
En fait, si la troncature opérée par le MOVE est appliquée à la partie entière, on perd les chiffres les plus significatifs. En particulier, dans le cas du dernier exemple, où la valeur 1000000.796 devient 0 après transfert.

4/ Conversion de format. Au cours du transfert des données numériques, le compilateur traite les champs avec une codification interne différente.
Connaissant l'USAGE implicite ou explicite des champs de départ et d'arrivée, il est capable de transformer correctement les données dans le format demandé.
Cette conversion est faite, sur la valeur éventuellement retranchée de la donnée, à l'intérieur du compilateur. Notons que cela revient à le pénaliser lourdement quand il s'agit de convertir une grande masse de données.
L'exécution de routines spéciales et la mobilisation d'une mémoire additionnelle engendrent, en effet, un accroissement du temps de calcul et une extension de l'occupation en mémoire centrale.
Il est donc conseillé de décrire les champs numériques d'une manière homogène, en ne faisant appel à la conversion de format qu'en cas de force majeure.

5/ Traitement du signe algébrique. Dernière opération du compilateur sur le champ d'arrivée : le traitement du signe. Si le champ contient, dans sa propre PICTURE, le symbole S, le signe de la valeur transférée est bien codifié, conformément à la codification interne employée.
Inversement, l'instruction MOVE transfère uniquement la valeur absolue de la donnée en l'absence du symbole S.
Il est donc recommandé de recourir systématiquement au symbole S dans la description des champs numériques afin d'éviter une transcription tronquée toujours possible, en dehors de quelques rares exceptions qui seront signalées au fur et à mesure de leur apparition.
Quand elle ne sont pas respectées, ces ex-

RETRANCHEMENTS DANS LE TRANSFERT DES DONNEES

Instructions de transfert	Contenus des champs d'arrivée après le MOVE
MOVE NOM-DEPART TO NOM-ARRIVEE 1	4 3 6 7 8
MOVE NOM-DEPART TO NOM-ARRIVEE 2	4 3 6 7 8 0 0
MOVE NOM-DEPART TO NOM-ARRIVEE 3	1 4 3 6
MOVE NOM-DEPART TO NOM-ARRIVEE 4	7
MOVE 1000000.796 TO NOM-ARRIVEE 5	0 0 0 0 0 0

INSTRUCTION MOVE

CHAMP-DEPART \ CHAMP-ARRIVEE				Composé	Elémentaire								
					Alpha-numérique	Alphabétique	Numérique					Numérique à l'impression	Alphanum. à l'impression
							DISPLAY	COMP	COMP-3	COMP-1	COMP-2		
Composé				A	A	A						P	P
Elémentaire	Alphanumérique			A	A	A						P	P
	Alphabétique			A	A	A							P
	Constante	Alpha-numérique		A	A							P	P
		Alphabétique		A	A	A							P
		Figurative	SPACES	A	A	A							P
			HIGH-VALUES LOW-VALUES	A	A	A							P
			ZEROES	A	A		N	N	N	N	N	P	P
		Numérique		A	A		N	N	N	N	N	P	P
	Alphanumérique d'impression			A	A								P
	Numérique d'impression			A	A								P
	Numérique	DISPLAY		A	A*		N	N	N	N	N	P	P
		COMP		A	A*		N	N	N	N	N	P	P
		COMP-3		A	A*		N	N	N	N	N	P	P
		COMP-1					N	N	N	N	N	P	
		COMP-2					N	N	N	N	N	P	

* Uniquement pour les nombres entiers

MOVE permis

MOVE non permis

ceptions sont traitées par le compilateur comme des erreurs. Cette méthode permet d'éviter de perdre du temps à la recherche d'erreurs difficiles à analyser.
L'exemple suivant illustre le cas d'une description erronée d'une donnée numérique qui aboutit au résultat : - 1 + 1 = 2.

```
01 COMPTEUR          PIC 9.
—
—
—
—
—
PROCEDURE DIVISION.
COMPTER.
        MOVE-1 TO COMPTEUR.
        ADD  1 TO COMPTEUR.
```

Le COMPTEUR ne comportant pas de symbole S, le transfert de - 1 engendre la perte du signe algébrique. Après le MOVE, le contenu est, de ce fait, 1 et non - 1, et la valeur obtenue en additionnant 1 est 2 et non 0.
Le tableau de la page précédente résume toutes les combinaisons possibles des MOVE alphanumériques et numériques. On a utilisé le symbole A pour indiquer un MOVE alphanumérique, N pour un MOVE numérique et P (Print), pour un MOVE d'impression.

L'instruction Inspect

Pendant son déroulement, le programme peut avoir besoin de gérer le contenu d'un champ : analyser le nombre de caractères de blanc qu'il contient, vérifier si une certaine combinaison de caractères est présente... Les instructions décrites jusqu'à présent ne peuvent répondre de manière bienfaisante à ces questions. C'est pourquoi on a recours à une instruction spécifique, INSPECT.
Dans son format général, cette instruction est capable de décompter et de remplacer les répétitions d'un groupe de caractères par un ou plusieurs autres caractères d'un champ déterminé.
Les deux fonctions (décompter et substituer) peuvent cependant être exécutées séparément, grâce à deux sous-formats.

La fonction de comptage. Pour réaliser cette fonction, l'instruction a besoin de savoir :

1/ Quel champ numérique servira au compteur (**compteur**)

2/ Le nom du champ contenant la chaîne de caractères à analyser **(chaîne)**

3/ et les objets qu'elle doit compter (**combinaison**)

Le tableau ci-contre (page 1041) fournit le format de l'instruction où « combinaison » définit la série des caractères à rechercher dans la chaîne analysée.
En réalité, pour effectivement opérer, l'instruction doit aussi savoir selon quelle modalité cette recherche doit être effectuée.

La clause ALL. Considérons, par exemple, le cas où l'on veut connaître le nombre de caractères A présents dans le champ :

```
01 COULEUR PICX (20) VALUE 'AMARANTE'.
```

Le calcul peut être effectué en utilisant, comme compteur, le champ :

```
01 COMP PIC S9(5)    COMP VALUE 0.
```

Notons que, pendant le comptage, l'instruction INSPECT incrémente la valeur du compteur pour chaque combinaison valable rencontrée. Il est donc nécessaire d'initialiser le compteur, sauf si l'on ne désire pas un calcul total des différentes chaînes analysées.
Dans notre cas, INSPECT aura le format suivant :

```
INSPECT COULEUR TALLYING COMP
                    FOR ALL ' A '.
```

La « combinaison » requise est alors ALL ' A ' (« tous les A présents dans la chaîne »). Après exécution, le résultat sera 3, mémorisé dans le champ COMP.
On aurait obtenu le même résultat avec les instructions :

```
MOVE ' A ' TO FLAG.
INSPECT COULEUR TALLYING COMP
                    FOR ALL FLAG.
```

FORMAT DE L'INSTRUCTION INSPECT POUR COMPTAGE

INSPECT chaîne TALLYING compteur FOR combinaison

Dans la dernière des expression, FLAG indique le champ de la WORKING-STORAGE SECTION:

```
01 FLAG   PIC X.
```

INSPECT aide également à rechercher en même temps, dans un champ donné, plusieurs combinaisons de caractères.

Considérons encore le champ couleur précédemment défini (CHAMP) pour lequel on désire calculer simultanément combien de groupes AMA, R, ANTE sont présents.
On procède ainsi :

```
01 COULEUR     PICX(20)  VALUE
                         'AMARANTE'.
01 COMP-AMA  PIC 9(2)   COMP VALUE 0.
01 COMP-R    PIC 9(2)   COMP VALUE 0.
01 COMP-ANTE PIC 9(2)   COMP VALUE 0.
PROCEDURE DIVISION.
COMPTER.
    INSPECT COULEUR
        TALLYING
        COMP-AMA FOR ALL ' AMA ',
        COMP-R FOR ALL ' R ',
        COMP-ANTE FOR ALL ' ANTE '.
```

A présent, voyons les valeurs des compteurs après l'opération :

```
COMP-AMA  = 1
COMP-R    = 1
COMP-ANTE = 1
```

La clause LEADING. Avec cette clause, l'instruction INSPECT vérifie si une certaine combinaison de caractères se retrouve en début de champ.
Pour savoir si le contenu du champ COULEUR commence par les caractères AM, on écrit :

```
INSPECT COULEUR
    TALLYING COMPTEUR
    FOR LEADING ' AM '.
```

Dans le cas que nous sommes en train d'examiner, AMARANTE commence par le bloc de caractères ' AM ', l'instruction incrémente donc le champ COMPTEUR.

Pour le même champ, l'instruction :

```
INSPECT COULEUR
    TALLYING COMPTEUR
    FOR LEADING ' MAR '.
```

n'aurait pas modifié le contenu de COMPTEUR, simplement parce que le bloc MAR ne constitue pas la partie initiale du mot AMARANTE.

La clause CHARACTERS. Elle fournit le nombre de caractères que contient un champ déterminé. La valeur donnée par l'instruction comprend aussi les blancs.

L'instruction :

```
INSPECT COULEUR
    TALLYING COMPTEUR
    FOR CHARACTERS.
```

rétablit la valeur 20, dans le compteur correspondant au nombre de caractères déclarés dans la PICTURE du champ COULEUR, et non 8 (nombre de caractères différents de blanc constituant le mot AMARANTE).

La clause BEFORE. Dans toutes les clauses de l'instruction INSPECT que nous avons examinées jusqu'à présent, le champ à analyser a toujours été balayé à partir du premier caractère et de gauche à droite.
Cette contrainte peut être éliminée grâce à deux clauses qui permettent d'effectuer toutes les recherches décrites précédemment en partant de n'importe quelle position.
Ce point de départ, qui n'est pas fixé de manière rigide par le programmeur, est lié au contenu du champ.
Il est donc permis de prendre, pour point de

EMPLOI DE LA CLAUSE BEFORE

```
 01  ARTICLE.
     05  NUMERO-SERIES                  PIC X(10).
     05  DESCRIPTION                    PIC X(20).
     05  .........
     05  .........
     .
     .
     .
 01  K-SERIES-CAR                       PIC 99(2)  COMP VALUE 0.
     88  TROIS-CARACTERES          VALUE 3.
     88  DEUX-CARACTERES           VALUE 2.
     88  UN-CARACTERE              VALUE 1.
*
*
*
 PROCEDURE DIVISION.
 DEBUT.
     .
     .
     .
     .
 REGARDER-SERIES.
     INSPECT NUMERO-SERIES
             TALLYING K-SERIES-CAR FOR CHARACTERS BEFORE '/'.
     IF  TROIS-CARACTERES
         PERFORM CALCUL-SERIE-3.
     IF  DEUX-CARACTERES
         PERFORM CALCUL-SERIE-2.
     IF  UN-CARACTERE
         PERFORM CALCUL-SERIE-1.
     .
     .
     .
```

départ, la position de la chaîne où a été vérifiée l'existence d'une combinaison de caractères. Si la clause CHARACTERS précédemment décrite n'est pas très courante, elle peut avoir des applications très intéressantes dès qu'elle est utilisée en même temps que BEFORE.
Un exemple très parlant concerne l'application de BEFORE à la gestion d'un magasin, où le code de série de chaque article est composé d'un nombre quelconque de caractères toujours séparé du nombre d'articles, par le caractère /.
Celle-ci est ainsi programmée :

ABC/1234
ZX/789
U/045

Supposons que nous ayions à calculer autrement chaque article selon que la série est constituée de 1, 2, 3..., N caractères.

La seule manière de connaître exactement la série à laquelle appartient l'article est de procéder comme il est indiqué dans le listing ci-dessus.
En reprenant l'exemple illustrant la première forme de la clause CHARACTERS, on aurait pu dénombrer les caractères du mot AMARANTE en utilisant indifféremment une des formes suivantes :

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CHARACTERS BEFORE.
```

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CARACTERS BEFORE
        SPACES
```

Le résultat aurait été 8 dans les deux cas.

On notera que, dans le second format, la constante figurative SPACES a été utilisée à la

place du caractère ' '.
Ceci reste valable pour n'importe quel format INSPECT. Dans ce contexte, SPACES équivaut à un seul espace blanc, tout comme ZEROS est interprété comme un seul 0.

La clause AFTER. Alors qu'avec la clause BEFORE l'instruction INSPECT s'exécute à partir du premier caractère à gauche et s'arrête au moment où l'existence du caractère déclaré dans BEFORE a été vérifiée, on a le comportement inverse avec AFTER.
En effet, le champ est balayé à partir de la gauche jusqu'à détection du caractère déclaré. Le contrôle des autres conditions indiquées dans l'instruction commence après cette opération de vérification.
Si, en considérant le champ :

```
01 COULEUR PICX(20) VALUE ' AMARANTE '.
```

on désire augmenter le nombre de caractères après le premier N rencontré dans le champ, on emploie l'instruction :

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CHARACTERS AFTER 'N'.
```

Par contre, si on désire connaître le nombre de caractères compris entre le groupe AMA et le groupe TE, l'instruction :

```
INSPECT COULEUR
        TALLYING COMPTEUR
        FOR CHARACTERS
                BEFORE 'TO'
                AFTER 'AMA'.
```

n'est pas admise.
Il faut recourir à plusieurs INSPECT et finalement calculer le nombre recherché par différence entre les valeurs obtenues par chaque instruction.
Le format général de l'instruction INSPECT utilisée pour compter les caractères est schématisé dans le tableau ci-dessous.

La fonction de substitution. Cette fonction nécessite d'examiner les trois points suivants :

1/ Champ à analyser par INSPECT
2/ Elément à remplacer
3/ Caractères de substitution

La fonction se présente ainsi :

```
INSPECT champ
        REPLACING combinaison-existante
        BY
        combinaison-nouvelle.
```

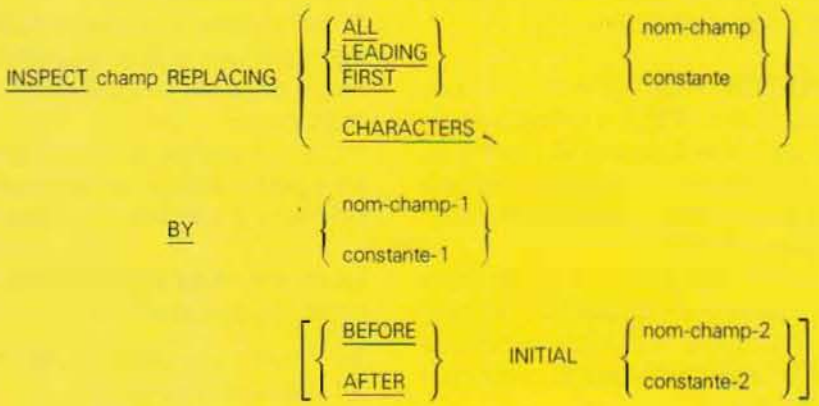
Cela revient à dire :

Balayer (INSPECT) le contenu du **champ** en remplaçant (REPLACING) la combinaison de caractères **combinaison-existante** par (BY) la nouvelle combinaison **combinaison-nouvelle**".
Comme pour la fonction de comptage, le terme "combinaison-existante" comprendra également, et d'une manière concise, la clause de description des caractères à substituer. Celle-ci

FORMAT GENERAL DE L'INSTRUCTION INSPECT AVEC FONCTION DE COMPTAGE

```
INSPECT champ TALLYING compteur FOR { { ALL       } { nom-champ          } }
                                    { { LEADING   } { constante          } }
                                    { { CHARACTERS} { constante-figurative} }

[ { BEFORE } INITIAL { nom-donnée } ]
[ { AFTER  }         { constante   } ]
```

FORMAT GENERAL DE L'INSTRUCTION INSPECT AVEC FONCTION DE SUBSTITUTION



est développée dans le format qui se trouve en haut de cette page.

Illustrons nos propos avec quelques exemples d'INSPECT.

Soit le champ NOMBRE-EN-LETTRES suivant :

```
01 NOMBRE-EN-LETTRES        PICX(30)
VALUE ' MILLESEPTCENTVINGTTROIS '.
```

Les configurations du champ, avant et après l'exécution des instructions INSPECT, sont données dans la page ci-contre.

Format général de l'instruction INSPECT.

Avec l'instruction INSPECT, on peut lancer, en même temps, des opérations de comptage et des opérations de substitution.
A noter qu'on peut même effectuer, avec cette instruction, plusieurs opérations de différent type sur le même champ.
Si on voulait effectuer sur le champ initial NOMBRE-EN-LETTRE :

1/ le décompte de tous les caractères différents de blanc, en utilisant K-CARACTERES comme compteur,
2/ le décompte de tous les caractères ' T ' (K-T comme compteur),
3/ la substitution de la chaîne ' WWWW ' au premier groupe ' CENT ',
4/ le remplacement de toutes les lettres ' T ' suivant le groupe ' SEPT ' par ' N ',

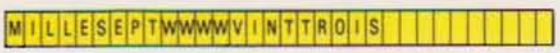
INSPECT s'écrira :

```
INSPECT NOMBRE-EN-LETTRES
        TALLYING K-CARACTERES
                 FOR CHARACTERS
                   BEFORE ' '
                 K-T
                 FOR ALL 'T'
      REPLACING FIRST ' CENT '
                 BY 'WWWW'
                 ALL 'T' BY 'N'
                 AFTER 'SEPT'.
```

Après l'exécution de l'instruction, on aura :

```
K-CARACTERES = 23
K-T          =  5
```

et la configuration du champ deviendra :



L'instruction INSPECT, (format ci-contre, page 1045), est également applicable aux champs numériques, à condition d'utiliser, implicitement ou explicitement, USAGE DISPLAY.

APPLICATION D'UNE INSTRUCTION INSPECT

CONFIGURATION INITIALE DU CHAMP NOMBRE-EN-LETTRES

```
M I L L E S E P T C E N T V I N G T T R O I S
```

INSPECT NOMBRE-EN-LETTRES REPLACING CHARACTERS BY ' * '.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

INSPECT NOMBRE-EN-LETTRES REPLACING LEADING 'MILLE' BY '— — — — —'.

```
— — — — — S E P T C E N T V I N G T T R O I S
```

INSPECT NOMBRE-EN-LETTRES REPLACING ALL ' ' BY ' * '.

```
M I L L E S E P T C E N T V I N G T T R O I S * * * * * * * *
```

INSPECT NOMBRE-EN-LETTRES REPLACING FIRST 'E' BY 'A'.

```
M I L L A S E P T C E N T V I N G T T R O I S
```

INSPECT NOMBRE-EN-LETTRE REPLACING ALL 'T' BY 'S'
ALL 'S' BY '8'
BEFORE 'VINGT'

```
M I L L E 8 E P S C E N S V I N G T T R O I S
```

FORMAT GENERAL DE L'INSTRUCTION INSPECT

```
INSPECT champ TALLYING compteur FOR { { ALL     }  { nom-donnée-1 } }
                                    { { LEADING }  { constante-1  } }
                                    {   CHARACTERS                  }

    [ { BEFORE }   INITIAL   { nom-donnée-2 } ]
    [ { AFTER  }             { constante-2  } ]

                 { { ALL     }      { nom-donnée-3 } }
    REPLACING    { { LEADING }      { constante-3  } }
                 { { FIRST   }                       }
                 {   CHARACTERS                      }

            BY   { nom-donnée-4 }
                 { constante-4  }

    [ { BEFORE }   INITIAL   { nom-donnée-5 } ]
    [ { AFTER  }             { constante-5  } ]
```

L'instruction STRING

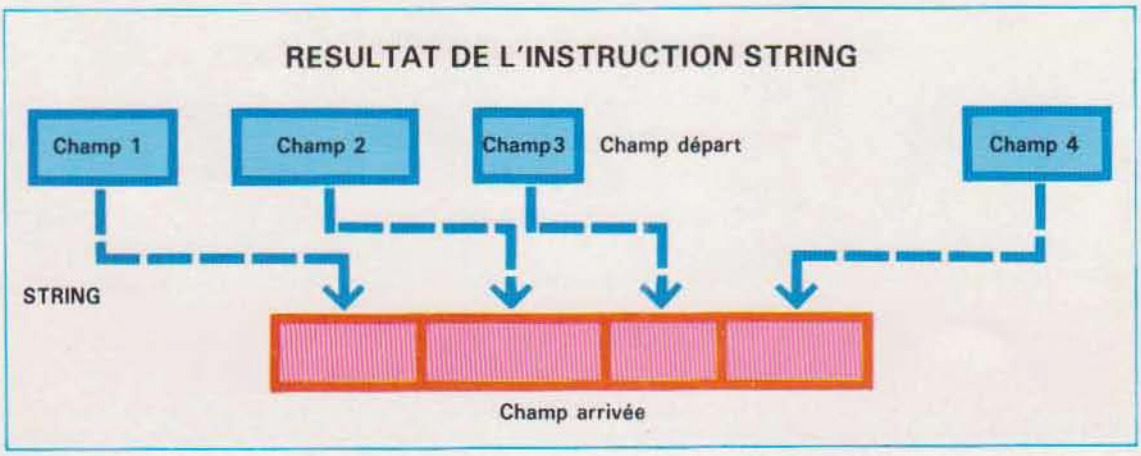
Pour compléter l'examen des instructions spécialisées dans la manipulation des données, analysons maintenant deux instructions très importantes : STRING et UNSTRING. Dans un certain sens, ces instructions regroupent les possibilités de MOVE et d'INSPECT.
Nous savons désormais que MOVE est capable de transférer des données d'un champ de départ vers un champ d'arrivée à condition d'être préalablement décrits dans la division données. Deux cas se présentent : soit l'instruction MOVE n'entre pas dans le contenu du champ (MOVE alphanumérique), soit elle le fait (MOVE numérique), mais alors il s'agit de respecter le type de champ d'arrivée. Pour MOVE, la valeur de la donnée transférée est complètement transparente et, en général, il n'est pas permis de ne déplacer qu'une partie de la donnée, à moins que le champ ne soit volontairement défini de cette manière.
Le champ CHAINE décrit comme suit :

```
01 CHAINE
   05 CHAINE-1          PIC X(3).
   05 CHAINE-2          PIC X(5).
      10 CHAINE-31       PIC X.
      10 CHAINE-32       PIC XX.
      10 CHAINE-33       PIC X(4).
```

contient la valeur « EXEMPLE DE MOVE », avec, comme contenu des différents sous-champs :

```
CHAINE-1  = 'EXE'
CHAINE-2  = 'MPLE'
CHAINE-31 = 'D'
CHAINE-32 = 'E'
CHAINE-33 = 'MOVE'
```

Dans ce champ, il est possible de déplacer, par exemple, la préposition 'DE' puisque les caractères la composant appartiennent à deux sous-champs différents.
Contrairement à l'instruction MOVE, l'instruction INSPECT fait abstraction d'éventuelles sous-descriptions du champ analysé. Elle « entre » dans son contenu et le modifie le cas échéant.
Les instructions STRING et UNSTRING constituent la seule manière d'analyser un contenu et d'effectuer le déplacement de certaines de ses parties. STRING permet de transférer le contenu complet ou partiel d'un ou de plusieurs champs vers un champ unique d'arrivée en les disposant l'un à la suite de l'autre selon l'ordre déclaré par le programmeur (voir schéma ci-dessous).
Supposons que l'on doive contrôler si une date saisie à la console sous la forme JJ/MM/AA (jour/mois/année) est antérieure ou non à celle de l'ordinateur.
Grâce à l'instruction ACCEPT, on peut prélever la date du jour sur le calendrier de la machine. Cette date est fournie dans le champ déclaré sous la forme AAMMJJ (année/mois/jour), plus appropriée aux opérations de contrôle. Ainsi le 13 décembre 1983 est facilement vérifiable car 831213 (représentation du 13/12/83) est plus grand que 831111 (représentation du 11/11/83). Il est évident que la date fournie sous la forme JJ/MM/AA



CONVERSION D'UNE DATE

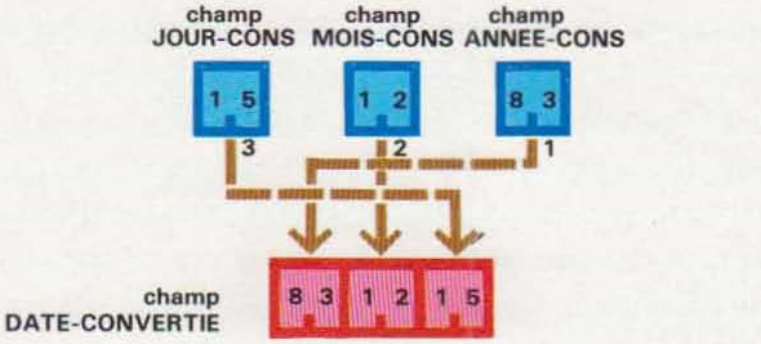
```
 01  DATE-CONSOLE.
     05  JOUR-CONS                PIC 9(2).
     05  FILLER                   PIC X.
     05  MOIS-CONS                PIC 9(2).
     05  FILLER                   PIC X.
     05  ANNEE-CONS               PIC 9(2).
     05  FILLER                   PIC X.
*
*
 01  DATE-CALCUL                 PIC 9(6).
 01  DATE-CALCUL    REDEFINES DATE-CALCUL.
     05  ANNEE-CALCUL             PIC 9(2).
     05  MOIS-CALCUL              PIC 9(2).
     05  JOUR-CALCUL              PIC 9(2).
*
*
 01  DATE-CONVERTIE.
     05  ANNEE-CONV               PIC 9(2).
     05  MOIS-CONV                PIC 9(2).
     05  JOUR-CONV                PIC 9(2).
*
*
 PROCEDURE DIVISION.
 ACCEPTER-DATES.
     DISPLAY '** DIGITALISER DATE (JJ,MM,AA)'
             UPON CONSOLE.
     ACCEPT  DATE-CONSOLE FROM CONSOLE.
     STRING  ANNEE-CONS
             MOIS-CONS
             JOUR-CONS
                        DELIMITED BY SIZE
                        INTO      DATE-CONVERTIE.
     ACCEPT  DATE-CALCUL FROM DATES.
     IF  DATE-CONVERTIE IS GREATER THAN DATE-CALCUL
         PERFORM .........
         ..................
         ..................
*
*
```

doit être opportunément convertie en AAMMJJ.
Le problème peut être résolu par le programme ci-contre (voir listing ci-dessus).
L'instruction :

```
STRING  ANNEE-CONS
        MOIS-CONS
        JOUR-CONS
        DELIMITED BY SIZE
INTO DATE-CONVERTIE
```

génère les opérations schématisées dans la page suivante où l'on suppose que la date du 15/12/83 a été saisie à la console.
En définitive, l'instruction STRING transfère, dans le champ arrivée (INTO DATE-CONVERTIE), et dans l'ordre indiqué, le contenu entier des champs ANNEE-CONS, MOIS-CONS, JOUR-CONS.
La clause DELIMITED BY SIZE indique que le transfert concerne le champ dans ses dimensions maximales, telles qu'elles ont été déclarées dans division données. On peut, en effet, noter dans cet exemple que les champs transférés étaient déclarés chacun pour 2 caractères : PIC 9(2).
Pour obtenir le même résultat, on aurait également pu utiliser trois MOVE à la place de STRING.
Connaissant toutes les clauses prévues par l'instruction STRING, son emploi paraît beaucoup plus simple. Les exemples suivants donnent une idée de sa puissance. Elle agit sur des champs alphanumériques et peut créer, dans un champ arrivée, une combinaison de chaînes

CONVERSION D'UN FORMAT DATE AVEC L'INSTRUCTION STRING



de caractères en provenance de plusieurs champs départ.
Les caractères à prélever sont déterminés en fonction d'un ou de plusieurs caractères déclarés comme bornes.
Soit, par exemple, le champ :

```
01 DEPART          PIC X(4).
```

dont le contenu, à un certain moment, vaut

```
ABCD
```

On veut composer dans le champ :

```
01 ARRIVEE         PIC X(40).
```

une chaîne de caractères contenant :

1/ la constante alphanumérique : EXEMPLE DE STRING,
2/ un blanc,
3/ le caractère : (deux points),
4/ un blanc,
5/ toutes les lettres qui précèdent la lettre C dans le contenu du champ DEPART.

L'instruction STRING (voir format général en page 1049) sera alors :

```
STRING 'EXEMPLE DE STRING' DELIMITED
                                BY SIZE
       ' '                      DELIMITED
                                BY SIZE
       ' : '                    DELIMITED
                                BY SIZE
       SPACE                    DELIMITED
                                BY SIZE
       DEPART                   DELIMITED
                                BY 'C'
INTO
       ARRIVEE.
```

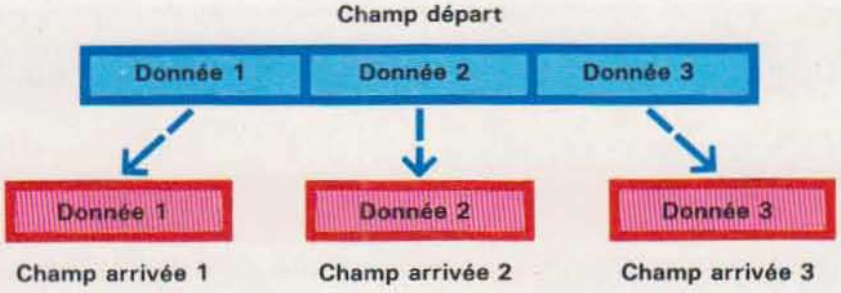
Dans ce cas, le champ arrivée commencera à partir du premier caractère à gauche. Il est possible d'établir un autre point de départ en utilisant un pointeur.
Ainsi, la chaîne précédente serait composée à partir de la septième position du champ ARRIVEE. Elle utiliserait alors les instructions suivantes :

```
MOVE 7 TO POSITION.
STRING 'EXEMPLE DE STRING'
       ' '
       ' : '
       SPACE      DELIMITED BY SIZE
       DEPART     DELIMITED BY 'C'
INTO
       ARRIVEE WITH POINTER POSITION
ON OVERFLOW
       DISPLAY
```

FORMAT GENERAL DE L'INSTRUCTION STRING

```
STRING   { nom-donnée-1 }  [ { nom-donnée-2 } ]
         { constante-1  }  [ { constante-2  } ]  ....

         DELIMITED BY  { nom-donnée-3 }
                       { constante-3  }
                       { SIZE         }

[        { nom-donnée-4 }  [ { nom-donnée-5 } ]
         { constante-4  }  [ { constante-5  } ]  ....

         DELIMITED BY  { nom-donnée-6 }
                       { constante-6  }
                       { SIZE         }  ]

         INTO nom-donnée-7
              [WITH POINTER nom-donnée-8]
              [ON OVERFLOW phrase-impérative]
```

DEROULEMENT DE L'INSTRUCTION UNSTRING



'CHAMP ARRIVEE INSUFFISANT'
UPON CONSOLE.

Dans notre exemple, a été insérée la clause OVERFLOW qui permet de contrôler si le champ d'arrivée est capable de contenir tous les caractères déclarés.

L'instruction UNSTRING

UNSTRING est l'inverse de STRING. L'instruction prélève des données d'un champ alphanumérique unique et les transfère dans un ou plusieurs champs d'arrivée, selon le schéma ci-dessus. La détermination des caractères à prélever est effectuée selon la même logique utilisée par l'instruction STRING, en spécifiant le caractère-limite (DELIMITED BY...).
De plus, UNSTRING comporte des clauses sans équivalents dans STRING (voir format général page 1050).
La clause ALL permet au compilateur d'interpréter un groupe de caractères identiques comme borne unique. Dans la chaîne :

BELLE

par exemple :

DELIMITED BY ALL 'L'

FORMAT GENERAL DE L'INSTRUCTION UNSTRING

```
UNSTRING nom-donnée-1
    [ DELIMITED BY [ALL]  { nom-donnée 2 | constante 1 }
          [ OR [ALL]      { nom-donnée 3 | constante 2 } ] ]
INTO                                  nom-donnée 4
     [DELIMITER IN                    nom-donnée 5]
     [COUNT IN                        nom-donnée 6]
    [ nom-donnée 7                                    ]
    [ [DELIMITER IN                   nom-donnée 8]   ] .....
    [ [COUNT IN                       nom-donnée 9]   ]
[WITH POINTER                         nom-donnée 10]
[TALLYING IN                          nom-donnée 11]
[ON OVERFLOW                          phrase-impérative].
```

EXEMPLE D'APPLICATION DE L'INSTRUCTION UNSTRING

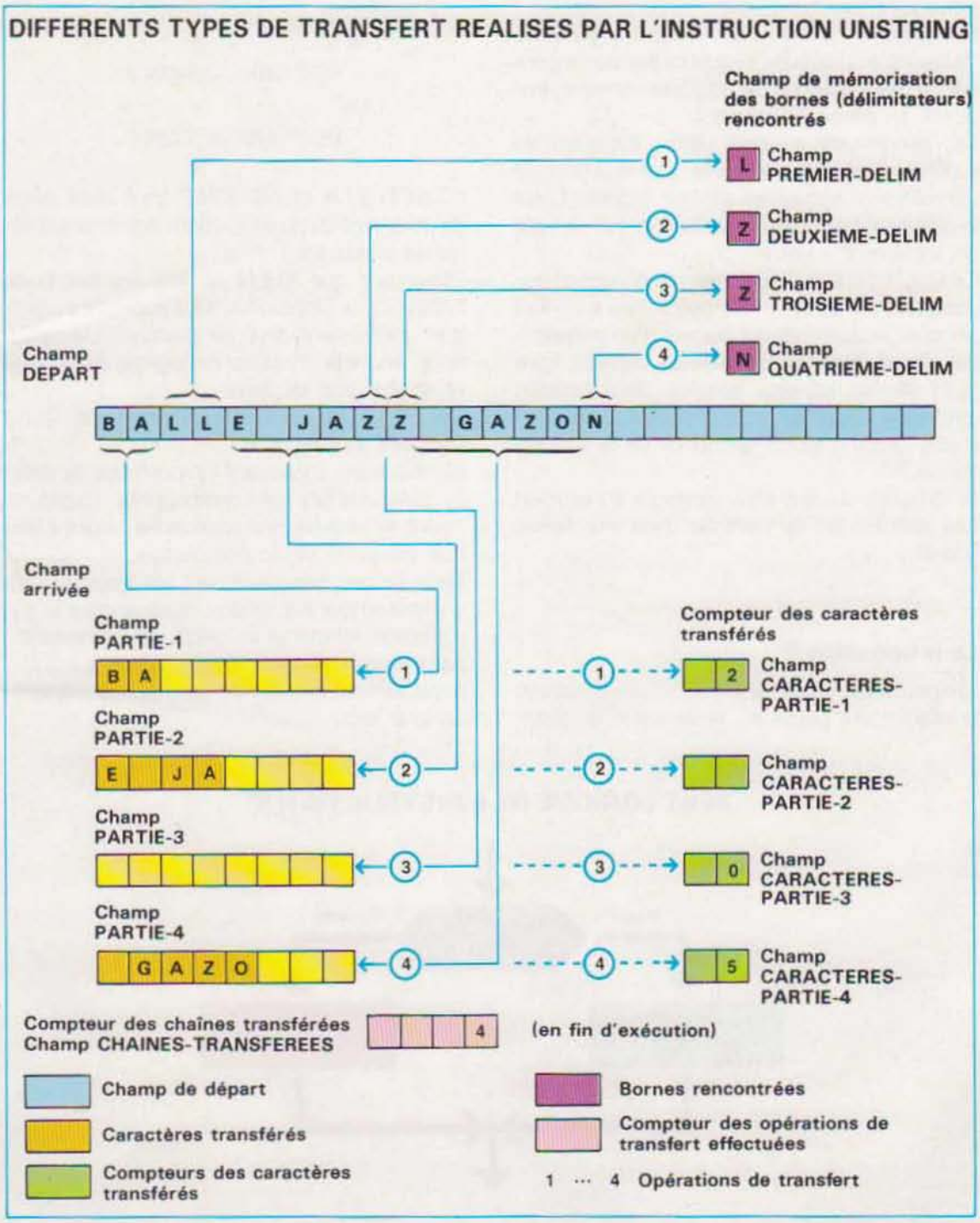
```
MOVE 'BALLE JAZZ GAZON' TO DEPART
UNSTRING DEPART
         DELIMITED BY Z,                          Définition des
                      ALL 'L'                     bornes (DELIM)
INTO PARTIE-1
     DELIMITER  IN  PREMIER-DELIM
     COUNT      IN  CARACTERES-PARTIE 1           1er transfert
*
     PARTIE-2
     DELIMITER  IN  SECOND-DELIM
     COUNT      IN  CARACTERES-PARTIE 2           2e transfert
*
     PARTIE-3
     DELIMITER  IN  TROISIEME-DELIM
     COUNT      IN  CARACTERES-PARTIE 3           3e transfert
*
     PARTIE-4
     DELIMITER  IN  QUATRIEME-DELIM
     COUNT      IN  CARACTERES-PARTIE 4           4e transfert
*
     TALLYING   CHAINE-TRANSFEREES                Compteur des opérations
     ON OVERFLOW                                  de transfert effectuées
          DISPLAY 'OVERFLOW UNSTRING'
               UPON CONSOLE.
```

fera ressortir le groupe 'LL' comme borne unique. Sans cette spécification de la clause ALL, seul le premier "L" aurait été traité comme délimiteur, (le second "L" aurait été identifié comme appartenant à la chaîne suivante).

Nous avons représenté graphiquement (tableau ci-dessous) les définitions des principales clauses de l'instruction UNSTRING.

Verbes de contrôle

Ce qui a été développé jusqu'à présent a per-



mis au lecteur de connaître toutes les opérations sur les données, celles d'entrée/sortie (INPUT/OUTPUT) pour le traitement des fichiers séquentiels en lecture ou en écriture, ainsi que les opérations qui sont spécifiques aux manipulations de données en mémoire. Elles constituent les outils essentiels au programmeur en Cobol.

Un programme est une structure logique de décisions et d'actions séquentielles correspondant à des situations particulières mises en évidence en phase d'analyse.

Le programmeur doit être capable de « débrancher », à n'importe quel moment, le déroulement séquentiel de test logique. C'est la réponse à ce test qui va déterminer le type de traitement à lancer.

Ce Cobol dispose d'un ensemble d'instructions destinées au contrôle des opérations en cours pendant le déroulement normal d'un programme. Ces instructions ont déjà été utilisées, sous leurs formes les plus simples, dans certains exemples présentés, que le lecteur a pu interpréter grâce à sa connaissance de la syntaxe du BASIC.

Le chapitre suivant sera consacré à l'examen des instructions de contrôle dans leur forme générale.

La proposition IF

Elle permet de tester le produit d'une condition préalablement posée et, en fonction du résultat, de choisir entre deux types d'actions. IF équivaut à la phrase suivante = "si (IF) une certaine condition est vérifiée, alors (THEN) exécuter l'ACTION-A, sinon (ELSE) exécuter l'ACTION-B". En utilisant le verbe PERFORM, on a (voir organigramme ci-dessous) :

```
IF condition
    THEN
        PERFORM ACTION-A
    ELSE
        PERFORM ACTION-B.
```

où ACTION-A et ACTION-B sont deux noms de paragraphes ou de sections contenant différentes instructions.

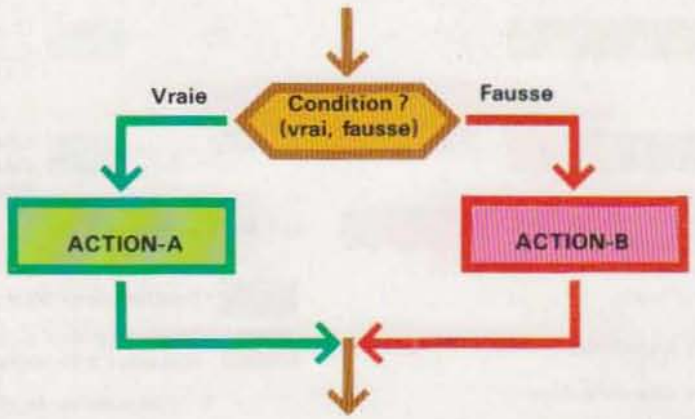
Observons que THEN ne fait pas partie de l'ANS Cobol. Donc nous nous bornerons à l'utiliser uniquement dans les premiers exemples, pour rendre la construction logique de la phrase entière plus évidente.

Nous suivrons, par la suite, les règles du Cobol standard de plus près.

Mentionnons également l'importance du point (.) dans une phrase conditionnelle. Oublié ou inséré de manière non appropriée, le point sera mal interprété par le compilateur.

Dans ce cas, non seulement les résultats sont complètement inattendus, mais encore le déroulement lui-même du calcul est imprévisible ; par exemple fin pour erreur, boucles infinies, dépassement des limites de l'espace mémoire assigné, etc.

TEST LOGIQUE DE L'INSTRUCTION IF



FORMAT GENERAL DE L'INSTRUCTION IF

```
IF condition [THEN]  { NEXT SENTENCE       }  [ ELSE  { NEXT SENTENCE       } ]
                     { phrase-impérative-1 }         { phrase-impérative-2 }
```

Avant de procéder à son analyse, il est bon de rappeler combien, en langage naturel, les phrases conditionnelles constituent des points de repère indispensables à la compréhension d'un organigramme.
Lorsque vous relisez un programme écrit par vous-même ou par une autre personne, l'étape essentielle consiste à repérer les points décisionnels, les points de branchement et les conditions posées.
En effet, si le contexte logique dans lequel une certaine instruction s'exécute n'est pas bien analysé, il devient, par avance, inutile d'entrer dans le détail du programme.

Lire un programme équivaut donc à lire les IF qu'il contient.

De ce fait, les points de décision du programme doivent être rédigés de manière à accrocher le regard dès la première lecture.
Le Cobol permet d'écrire la plupart des instructions tout en respectant ses règles syntaxiques. Et la seule disposition des opérandes devrait donner une idée de leur fonction logique.
Tel qu'il est analysé à la page précédente, et tel qu'il est explicité dans l'encadré ci-dessus, le format de l'instruction IF est difficile à interpréter. Il est donc vivement recommandé de ne l'utiliser qu'avec des clauses détaillées :
De cette façon :

```
[THEN]  { NEXT SENTENCE       }
        { phrase-impérative-1 }

[ ELSE  { NEXT SENTENCE       } ]
        { phrase-impérative-2 }
```

- on fait ressortir la condition dans le contexte du programme,
- on isole clairement les instructions qui représentent l'alternative offerte par le test.

Dans l'exemple suivant, les instructions IF ont une syntaxe correcte. Pourtant, à cause de leur disposition, l'alternative ne saute pas aux yeux.

```
IF TYPE-CARTE = 'A' MOVE CARTE
TO CARTE-DISPONIBLE
COMPUTE SOLDE = CREDIT-DEBIT
ADD SOLDE TO SOLDE-TOTAL
ADD 1 TO CARTES-LUES
GO TO LIRE-CARTES ELSE
ADD 1 TO CARTES-LUES
ADD 1 TO CARTES-DECOUVERTES
GO TO LIRE-CARTES.
```

Voici le même exemple programmé comme nous vous le conseillons :

```
IF TYPE-CARTE       = 'A'

   MOVE CARTE TO BAC-DISPONIBLE
   COMPUTE SOLDE = CREDIT-DEBIT
   ADD SOLDE      TO SOLDE-TOTAL
   ADD 1          TO CARTES-LUES
   GO TO          LIRE-CARTE

ELSE
   ADD 1          TO CARTES-LUES
                  CARTES-DECOUVERTES
   GO TO          LIRE-CARTES.
```

Sous cette forme, on peut facilement remonter à l'organigramme et, par conséquent, avoir une compréhension immédiate des fonctions dans leur déroulement.

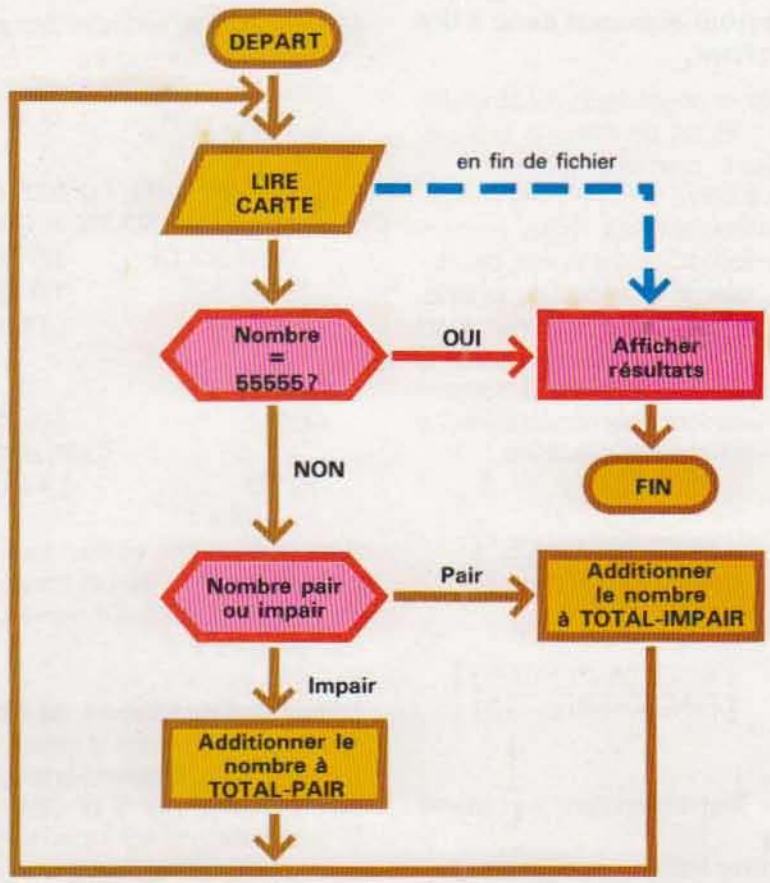
Logique d'exécution de l'instruction IF.
Avant d'entrer dans le détail des différentes analyses accompagnant la proposition IF, il est bon d'en examiner le déroulement.
Si la condition spécifiée immédiatement après la proposition IF se révèle vraie, la phrase qui la suit est exécutée jusqu'à ce qu'elle rencontre

le mot ELSE ou le premier point. Après exécution de la phrase, le contrôle passe à la deuxième phrase (la phrase après le 1er point). Si, au contraire, la condition est fausse, c'est la phrase qui suit immédiatement la proposition ELSE (quand elle existe) qui est exécutée et le contrôle passe ensuite à la première instruction suivant le point de fermeture de la phrase conditionnelle.

Voici quelques exemples :
Nous voulons calculer les sommes des nombres entiers, pairs et impairs, perforés (un nombre par carte) sur un fichier. Le calcul terminé les résultats obtenus sont affichés à l'écran dès que l'on rencontre, dans le fichier, une carte avec le nombre 55555. Il est évident que, pour effectuer la somme des nombres impairs de l'autre, il est nécessaire de reconnaître la nature du nombre qu'on vient de lire sur la carte.
Par ailleurs, avant de traiter le nombre lu, il faut vérifier qu'il n'est pas égal à 55555. A première vue, le programme suit l'organigramme ci-dessous. On divise un nombre par 2 pour connaître sa parité : si le reste de la division est 0, le nombre est pair, sinon impair. Le programme décrit peut donc être détaillé tel qu'il appartient dans le tableau ci-contre. A sa lecture, on observe que si la condition NOMBRE = 55555 s'avère vraie, on exécute, l'instruction GO TO FIN-CALCUL, alors que, toutes les fois où elle est fausse, le programme passe à l'instruction suivant le point :

```
DIVIDE    NOMBRE BY 2
          GIVING QUOTIENT
          REMAINDER RESTE
```

EXEMPLE DE PHRASES CONDITIONNELLES



APPLICATION DE L'INSTRUCTION IF

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ESSAI-IF.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. ..............
 OBJECT-COMPUTER. ..............
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT CARTES ASSIGN TO CARD-READER CARTES.
*
*
*
 DATE DIVISION.
 FILE SECTION.
 FD  CARTES
     LABEL RECORD OMITTED.
 01  CARTE                    PIC X(80).
*
*
 WORKING STORAGE SECTION.
 01  CARTE-WS.
     05  NUMERO               PIC 9(5) COMP.
     05  FILLER               PIC X(75).
*
 01  QUOTIENT                 PIC 9(5)V99 COMP.
 01  RESTE                    PIC 9(2)    COMP.
 01  ADDITIONNER-PAIRS        PIC 9(10)   COMP VALUE 0.
 01  ADDITIONNER-IMPAIRS      PIC 9(10)   COMP VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 DEBUT SECTION.
 OUVRIR-FICHIER.
     OPEN INPUT CARTES.
 LIRE-CARTE.
     READ CARTES INTO CARTE-WS
                 AT END
                 GO TO FIN-CALCUL.
     IF  NOMBRE = 55555
         GO TO FIN-CALCUL.
     DIVISE NOMBRE BY 2
         GIVING QUOTIENT
         REMAINDER RESTE.
     IF  RESTE = 0
         ADD NOMBRE TO ADDITIONNER-PAIRS
     ELSE
         ADD NOMBRE TO ADDITIONNER-IMPAIRS.
     GO TO LIRE-CARTE.
*
*
*
 FIN-CALCUL.
     DISPLAY '** ADDITIONNER NOMBRES PAIRS = ' ADDITIONNER-PAIRS
             UPON CONSOLE.
     DISPLAY ' '
             UPON CONSOLE.
     DISPLAY '** ADDITIONNER NOMBRES IMPAIRS = ' ADDITIONNER-IMPAIRS
             UPON CONSOLE.
 FERMER-FICHIER.
     CLOSE CARTES.
 FIN.
     DISPLAY '*** CALCUL TERMINE ***'
             UPON CONSOLE.
     STOP RUN.
```

Après la division du nombre par 2, le reste est égal à 0 ou ne l'est pas. Dans le 1er cas, le nombre (pair) est ajouté à ADDITION-PAIR, sinon (ELSE) il est ajouté à ADDITION-IMPAIR. La phrase conditionnelle se termine avec le point de fin d'instruction :

```
ADD NOMBRE TO ADDITION-IMPAIR
```

Quelle que soit la décision prise par le programme dans le test conditionnel, la main est toujours cédée à l'instruction suivant immédiatement la phrase conditionnelle elle-même. Une fois l'analyse sur la valeur du reste (RESTE) terminée et l'instruction correspondante exécutée, le programme retourne au paragraphe LIRE-CARTE.
Ce cycle est exécuté jusqu'à épuisement du fichier (AT END) ou jusqu'à lecture d'une carte contenant 55555.
A propos de l'importance du point dans l'exacte définition d'une phrase conditionnelle, on observe que si le second IF avait été mal écrit, nous aurions :

```
IF RESTE = 0
    ADD NOMBRE TO ADDITION-PAIR
ELSE
    ADD NOMBRE TO ADDITION-IMPAIR
GO TO LIRE-CARTE.
```

(autrement dit si le point avait été omis dans l'instruction ADD NOMBRE TO ADDITION-IMPAIR), le compilateur aurait interprété la séquence des opérations comme page 1057.
Le programme n'aurait relu les cartes que dans le cas où le nombre traité précédemment avait été impair. Une fois le premier nombre pair rencontré, et après l'incrémentation de ADDITION-PAIR, le programme serait passé au paragraphe FIN-CALCUL avec comme conséquence l'arrêt des calculs !

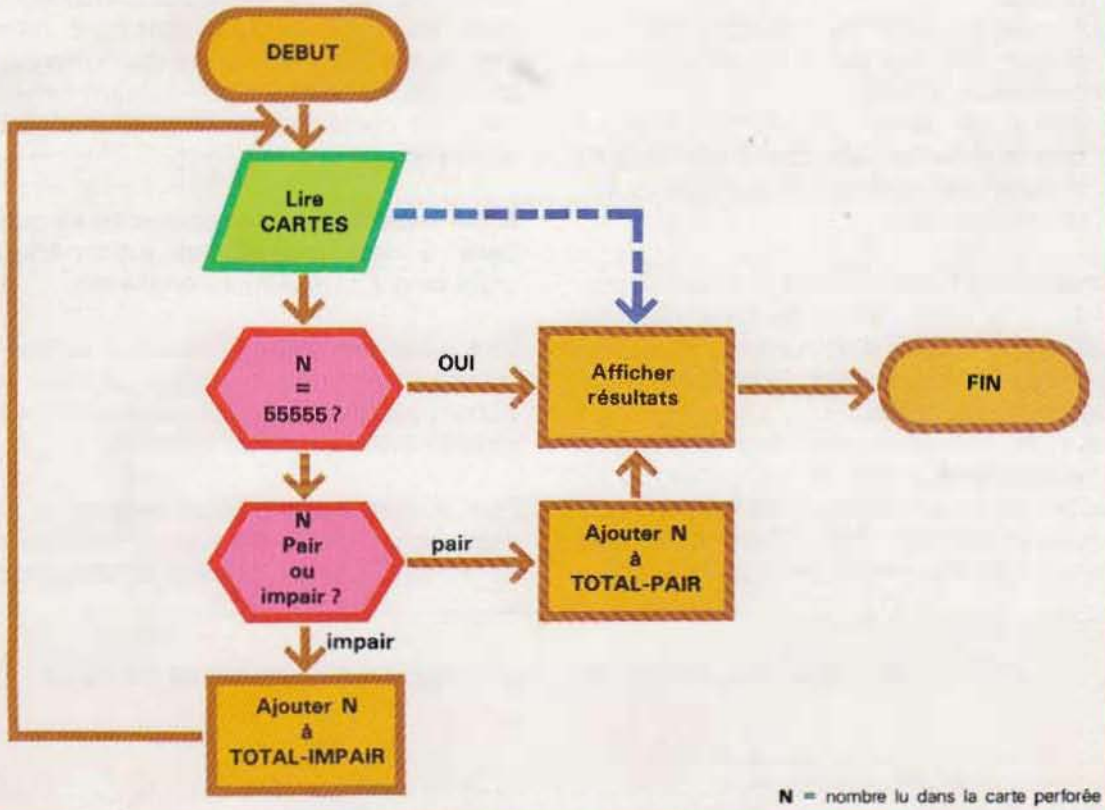
L'option NEXT SENTENCE. Elle permet de transférer le contrôle des opérations à la première instruction suivant immédiatement le point de fermeture de la phrase conditionnelle. Pour éclairer ce concept, considérons à nouveau le premier IF de l'exemple précédent :

```
IF NOMBRE = 55555
  GO TO FIN-CALCUL.
DIVIDE . . . . . .
. . . . . .
```

Test numérique d'un afficheur à cristaux liquides LCD (Liquid Cristal Display).

M. Wolf/Graza Neri-Black Star

INTERPRETATION ERRONEE D'UNE SEQUENCE CONDITIONNELLE



On aurait obtenu le même résultat en écrivant :

```
IF NOMBRE NOT = 55555
   NEXT SENTENCE
ELSE
   GO TO FIN-TRAITEMENT.
DIVIDE .....
```

Chaque fois que le contenu du champ NOMBRE est différent de 55555, la main est donnée à l'instruction immédiatement après le point, à savoir :

```
DIVIDE NOMBRE BY 2
     GIVING QUOTIENT
     REMAINDER RESTE.
```

Dans le cas contraire, c'est GO TO FIN-TRAITEMENT qui est exécutée.

Bien que les deux formes soient équivalentes quant au résultat, la seconde est d'une lecture moins immédiate que la première.
De là l'avantage de structurer les IF (si) d'une manière simple et linéaire.
Cela rend plus facile la rédaction et la lecture du programme et surtout la maintenance.

Conditions liées à l'instruction IF. Dans le format général de l'instruction IF, la nature des conditions posées n'a pas encore été précisée pas plus que la phrase conditionnelle.

Il existe, en effet, plusieurs tests conditionnels applicables au IF qui peuvent être résumées en 4 types :

- Analyse de relation : elle met une condition à l'exécution d'une section déterminée du programme portant sur la comparaison (plus grand, égal, plus petit, etc.). Elle revêt un caractère particulier quand il s'agit de données alphabétiques ou alphanumériques.

- Analyse de signe : elle subordonne un traitement à la nature du signe (+ ou -) de la donnée.
- Analyse de condition : l'exécution de l'instruction est soumise à la vérification du contenu du champ.
- Analyse de classe : elle vérifie la classe à laquelle appartient une donnée (numérique / alphabétique) avant de lancer le type de traitement approprié.

Analyse de relation. Dans ce mode, on procède à la comparaison de deux quantités (opérandes) à l'aide d'un opérateur relationnel spécifique (voir ci-dessous, 1er encadré, le format de la proposition IF).
Les opérateurs relationnels (résumés en bas de page) qualifient le type de comparaison à effectuer sur les opérandes. Tous les opérateurs s'écrivent indifféremment à l'aide du symbole correspondant ou en clair (en anglais). Les opérandes (objets de la comparaison) sont des champs décrits dans la division donnée, des constantes ou des expressions mathématiques. Toutes les analyses sont donc autorisées, quelle que soit la nature des opérandes : un champ peut être comparé à un autre champ avec une constante ou avec une expression arithmétique.

Mais attention : une constante se compare à une expression arithmétique mais non à une autre constante.

Lorsqu'une des deux opérandes à comparer est une expression arithmétique, sa valeur est d'abord calculée, et ensuite seulement mise en parallèle avec la seconde opérande.

Pour illustrer notre propos, prenons le cas d'une lecture. Nous disposons d'un fichier de cartes dont la forme de l'enregistrement est :

FORMAT DE L'INSTRUCTION IF DANS UNE ANALYSE DE RELATION

```
IF   { nom-donnée-1 }                  { nom-donnée-2 }
     { constante-1 }   opérateur       { constante-2 }
     { expression-1 }                  { expression-2 }

[THEN]
     { NEXT SENTENCE }
     { phrase-1      }

[ ELSE
     { NEXT SENTENCE }
     { phrase-2      }
```

OPERATEURS RATIONNELS

Symbole Cobol	Equivalent mathématique	Forme complète	Signification
=	=	IS EQUAL [TO]	... égal ...
NOT =	≠	IS NOT EQUAL [TO]	... différent ...
>	>	IS GREATER [THAN]	... supérieur ...
<	<	IS LESS [THAN]	... inférieur ...
NOT >	⩽	IS NOT GREATER [THAN]	... inférieur ou égal ...
NOT <	⩾	IS NOT LESS [THAN]	... supérieur ou égal ...

Finsiel

Système IBM 370 dans une application de gestion d'archives stockées sur bandes magnétiques.

```
01 CARTE-WS.
   05 TYPE-CARTE          PIC X(3).
   05 ARTICLE.
      10 SERIE            PIC X(3).
      10 NUMERO           PIC 9(5).
      10 DESCRIPTION      PIC X(30).
   05 STOCK               PIC 9(5).
   05 SEUIL-MINIMUM       PIC 9(3).
   05 VENTE               PIC 9(5).
   05 VENTES-MOIS         PIC 9(2).
```

Supposons que chaque carte lue doive être soumise aux contrôles suivants :

1/ La carte est égale à MAG. Sinon la rejeter en incrémentant un compteur de cartes lues (à afficher en fin de traitement).

2/ Les descriptions de l'article ne sont pas vides. Sinon rejeter la carte sans oublier d'afficher, après totalisation, le nombre de cartes écartées en fin de traitement.

3/ Le stock (quantité réelle) n'est pas inférieur au seuil d'approvisionnement minimum (pour éviter d'être en rupture de stock »). Sinon, signaler l'événement par un message approprié (STOCK D'ALARME).

4/ Totaliser les articles mouvementés dans le mois de référence indiqué par l'opérateur de saisie.

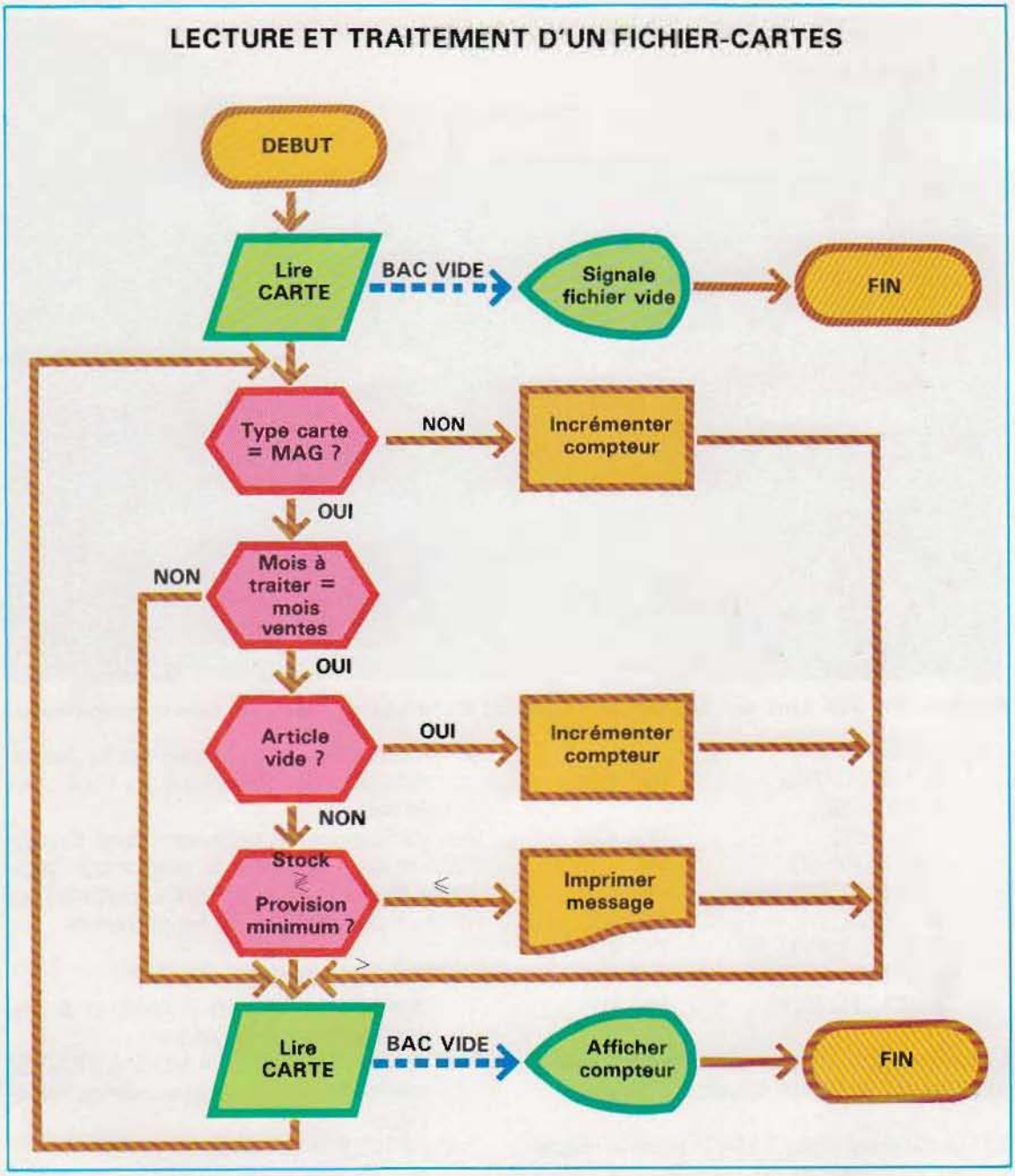
Ces vérifications se retrouvent dans la page 1060 et dans le listing du programme (page 1061). Plusieurs types d'IF ont été utilisés pour l'analyse de relation entre les opérandes :

IF TYPE-CARTE NOT = 'MAG'
compare un champ numérique à une constante alphanumérique.

IF VENTES-MOIS NOT = MOIS-A-TRAITER
vérifie l'égalité entre deux champs numériques.

IF ARTICLE = SPACES
met en relation et vérifie l'égalité entre un champ numérique et la constante figurative SPACES. Cela équivaut à vérifier si le contenu entier du champ est composé d'espaces.

IF STOCK-VENTES < SEUIL-MINIMUM
compare le résultat de l'expression arithmétique STOCK-VENTES avec le contenu du champ numérique SEUIL-MINIMUM (stock d'alarme, dans l'organigramme).



L'analyse de relation s'effectue aussi bien entre champs de même classe (alphanumériques, numériques) qu'entre champs différents.
Une bonne connaissance des règles de comparaison est essentielle pour écrire un programme, et ainsi éviter certaines erreurs qui n'apparaissent, étant donné le nombre de comparaisons, qu'en phase de compilation.
A ce niveau, ou bien les résultats sont faux, ou bien le calcul s'achève d'une manière inattendue.

Soit la phrase conditionnelle :

```
IF SW-ACCORD = '0'
   PERFORM TRAITEMENT
ELSE
   PERFORM FIN-TRAITEMENT.
```

LECTURE ET TRAITEMENT D'UN FICHIER CARTES

```
01  CARTE-WS.
    05  TYPE-CARTE                      PIC X(3).
    05  ARTICLE.
        10  SERIES                      PIC X(3).
        10  NUMERO                      PIC 9(5).
        10  DESCRIPTION                 PIC X(30).
    05  STOCK                           PIC 9(5).
    05  SEUIL-MINIMUN                   PIC 9(3).
    05  VENTES                          PIC 9(5).
    05  VENTES-MOIS                     PIC 9(2).
*
01  CALCUL-MOIS                         PIC 9(2).
*
01  CARTES-LUES                         PIC 9(3) COMP VALUE 0.
01  CARTES-INCORRECTES                  PIC 9(3) COMP VALUE 0.
01  CARTES-ECARTEES                     PIC 9(3) COMP VALUE 0.

PROCEDURE DIVISION.
DEBUT.
    MOVE 0 TO CARTES-ECARTEES
              CARTES-INCORRECTES
              CARTES-LUES.
    DISPLAY '** ENTRER MOIS A TRAITER **'
            UPON CONSOLE.
    ACCEPT CALCUL-MOIS FROM CONSOLE.
OUVRIR-FICHIER.
    OPEN INPUT CARTES.
PREMIERE-LECTURE.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY '** FICHIER CARTES VIDE **'
                        UPON CONSOLE
                GO TO FIN-CALCUL.
CONTROLE.
    ADD 1       TO CARTES-LUES.
    IF  TYPE-CARTE NOT = 'MAG'
        ADD 1       TO CARTES-ECARTEES
        GO TO LIRE-CARTES.
    IF  VENTES-MOIS NOT = CALCUL-MOIS
        GO TO LIRE-CARTES.
    IF  ARTICLE = SPACES
        ADD 1       TO CARTES-INCORRECTES
        GO TO LIRE-CARTES.
    IF  STOCK - VENTES LESS THAN SEUIL-MINIMUM
        DISPLAY '** ARTICLE : '
                DESCRIPTION
                '  * SERIE  : '
                SERIE
                '  * NUMERO : '
                NUMERO
                '  * SOUS SEUIL **'
                UPON PRINTER.
LIRE-CARTES.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY '***      CARTES LUES        ='
                        CARTES-LUES
                        UPON PRINTER
                DISPLAY '***      CARTES ECARTEES ='
                        CARTES-ECARTEES
                        UPON PRINTER
                DISPLAY '***      CARTES INCORRECTES ='
                        CARTES-INCORRECTES
                        UPON PRINTER
                GO TO FIN-CALCUL.
    GO TO CONTROLE.
*
FIN-CALCUL.
    CLOSE CARTES.
    STOP RUN.
```

avec SW-ACCORD décrit ainsi dans la WORKING-STORAGE SECTION :

```
01 SW-ACCORD*          PIC X(3)
```

Si, à un certain moment, SW-ACCORD contient la valeur non numérique :

0	0	0

l'analyse de relation IF SW-ACCORD = '0' sera fausse, et le programme effectuera l'appel à la procédure FIN-TRAITEMENT.
Ce résultat ne s'explique que si l'on a bien compris le mécanisme qui règle la comparaison. La proposition IF permet de mettre en relation des champs non numériques ou des champs numériques et non numériques, quelle que soit leur longueur.
Considérons, pour le moment, des champs de même longueur.
Dans ce cas, la comparaison est faite caractère par caractère en partant de la gauche jusqu'à l'épuisement des caractères à analyser ou jusqu'au rejet de la condition exigée (non satisfaite).
Soit, par exemple, CHAMP-1 et CHAMP-2 décrits :

```
01 CHAMP-1          PIC X(3).
02 CHAMP-2          PIC X(3).
```

Posons les valeurs :

CHAMP 1 =

A	B	C

CHAMP 2 =

A	B	D

Nous voulons vérifier la condition d'égalité suivante :

```
IF CHAMP-1 = CHAMP-2
   DISPLAY'LES CHAMPS SONT EGAUX'
           UPON CONSOLE
ELSE
   DISPLAY'LES CHAMPS NE SONT PAS
   EGAUX'
           UPON CONSOLE
   STOP RUN.
```

Le mécanisme de comparaison est schématisé dans le graphique de la page ci-contre.
Examinons maintenant le cas où l'on veut vérifier si le contenu du CHAMP-1 est plus petit que le contenu du CHAMP-2.
Nous entreprendrons, par conséquent, les actions décrites dans la phrase suivante :

```
IF CHAMP-1   CHAMP-2
   DISPLAY 'CHAMP-1 PLUS PETIT
           QUE CHAMP-2'
           UPON CONSOLE
ELSE
   DISPLAY 'CHAMP-1 PLUS GRAND OU
           EGAL AU CHAMP-2'
           UPON CONSOLE
STOP RUN.
```

Dans ce cas, le programme s'assure avant tout que la longueur des deux champs est égale ; il procède ensuite à la comparaison caractère par caractère en partant de la gauche (comme dans l'exemple précédent).
Le critère, qui permet de comparer les valeurs respectives de deux caractères, dépend de la manière dont ils sont hiérarchisés à l'intérieur du code utilisé par la machine (ASCII ou EBCDIC).
Ainsi, le résultat de la comparaison entre les deux valeurs A B Z et A B 9 est :

ABZ AB9 en EBCDIC
et
ABZ AB9 en ASCII.

Dans le premier langage, les représentations des lettres de l'alphabet ont toutes des valeurs inférieures à celles des nombres, contrairement au code ASCII (voir tableau ci-contre).
Quand les opérandes d'une proposition IF n'ont pas la même longueur, la comparaison s'effectue comme si l'opérande inférieure était remplie de blancs sur sa droite.
A ce propos, considérons à nouveau l'exemple déjà étudié :

```
IF SW-ACCORD = '0'
   PERFORM TRAITEMENT
ELSE
   PERFORM FIN-TRAITEMENT.
```

avec, pour contenu de SW-ACCORD, au moment de l'analyse :

COMPARAISON ENTRE DEUX CHAMPS ALPHANUMERIQUES

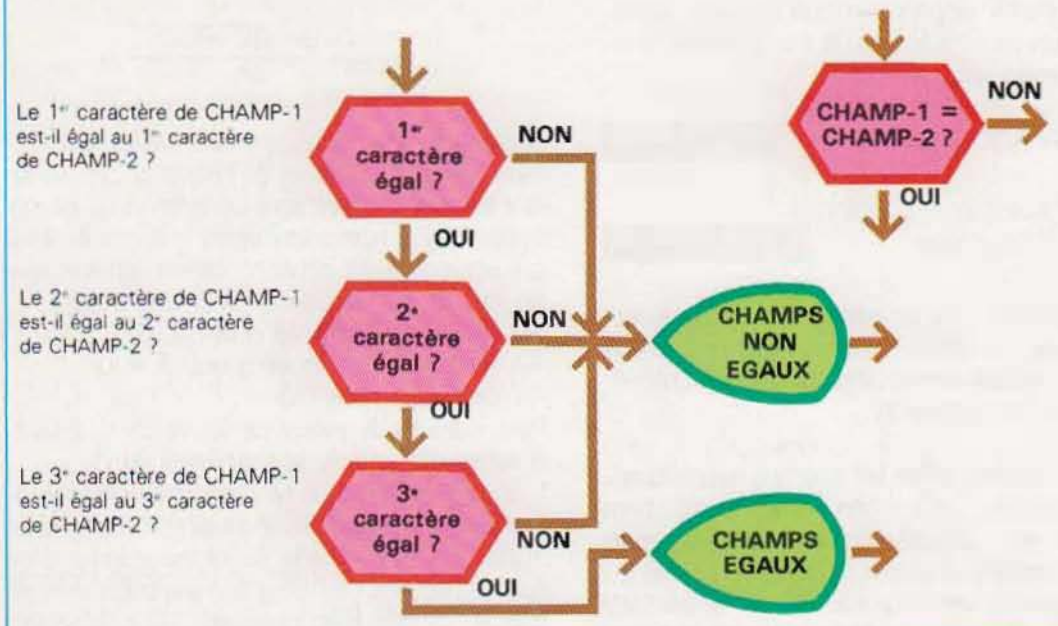


TABLEAU COMPARATIF DES CODES EBCDIC ET ASCII

Caractères EBCDIC		Caractères ASCII	
(valeurs inférieures)		**(valeurs inférieures)**	
b	blanc	b	blanc
.	point ou point décimal	"	guillemets
<	inférieur	$	dollar
(	parenthèse ouvrante	'	apostrophe
+	plus	(	parenthèse ouvrante
$	dollar	)	parenthèse fermante
*	astérisque	*	astérisque
)	parenthèse fermante	+	plus
;	point-virgule	,	virgule
–	moins ou trait d'union	–	moins ou trait d'union
/	barre de fraction	.	point ou point décimal
,	virgule	/	barre de fraction
>	supérieur	0-9	chiffres de 0 à 9
'	apostrophe	;	point-virgule
=	égal	<	inférieur
"	guillemets	=	égal
A-Z	lettres de A à Z	>	supérieur
0-9	chiffres de 0 à 9	A-Z	lettres de A à Z
(valeurs supérieures)		**(valeurs supérieures)**	

0	0	0

La constante alphanumérique 0 étant constituée d'un seul caractère, la comparaison s'effectue entre les champs :

SW-ACCORD	0	0	0

et constante '0' étendue à 3 caractères

0	0	0

Comme on le voit, et indépendamment du code utilisé, les deux valeurs ne seront jamais égales. Le programme appellera donc la procédure FIN-TRAITEMENT.

La comparaison entre les champs numériques, contrairement aux non-numériques, tient compte non des champs eux-mêmes, mais seulement de leur valeur algébrique. Toutes les règles mathématiques sont donc respectées dans la comparaison entre champs numériques. Il en résulte, par exemple (notation américaine), que :

+0000005.0000		vaut 5
- 425	est plus grand que	- 728
+12	est plus grand que	- 65

Analyse de signe. La proposition IF est capable d'analyser le signe du contenu d'un champ numérique ou du résultat d'une expression arithmétique en utilisant le format représenté dans le tableau ci-dessous.

Soit l'équation du second degré :

$$Ax^2+Bx+C=0$$

dont il faut extraire les racines. Les solutions de l'équation sont :

$$x_1=\frac{-B-\sqrt{B^2-4AC}}{2A}$$

$$x_2=\frac{-B+\sqrt{B^2-4AC}}{2A}$$

Supposons que nous voulions seulement calculer les racines réelles de l'équation, en signalant les cas où elles sont complexes et en désignant par Δ **(discriminant)** le terme B^2-4AC qui apparaît sous la racine carrée, on sait que les racines de l'équation du 2e degré sont :
— réelles et distinctes quand $\Delta>0$
— réelles et identiques quand $\Delta = 0$
— complexes si $\Delta<0$
Pour calculer la valeur de x_1 et de x_2 il faut, d'autre part, que A soit différent de 0.
L'organigramme d'un tel programme est représenté dans la page ci-contre et son listing page 1066. Rappelons que la racine carrée d'un nombre équivaut à l'élévation à la puissance du même nombre avec exposant 1/2 (0,5), soit :

$$\sqrt{B^2-4AC}=\sqrt{DELTA}=(DELTA)^{1/2}=$$
$$=(DELTA)^{0.5}=DELTA**0.5$$

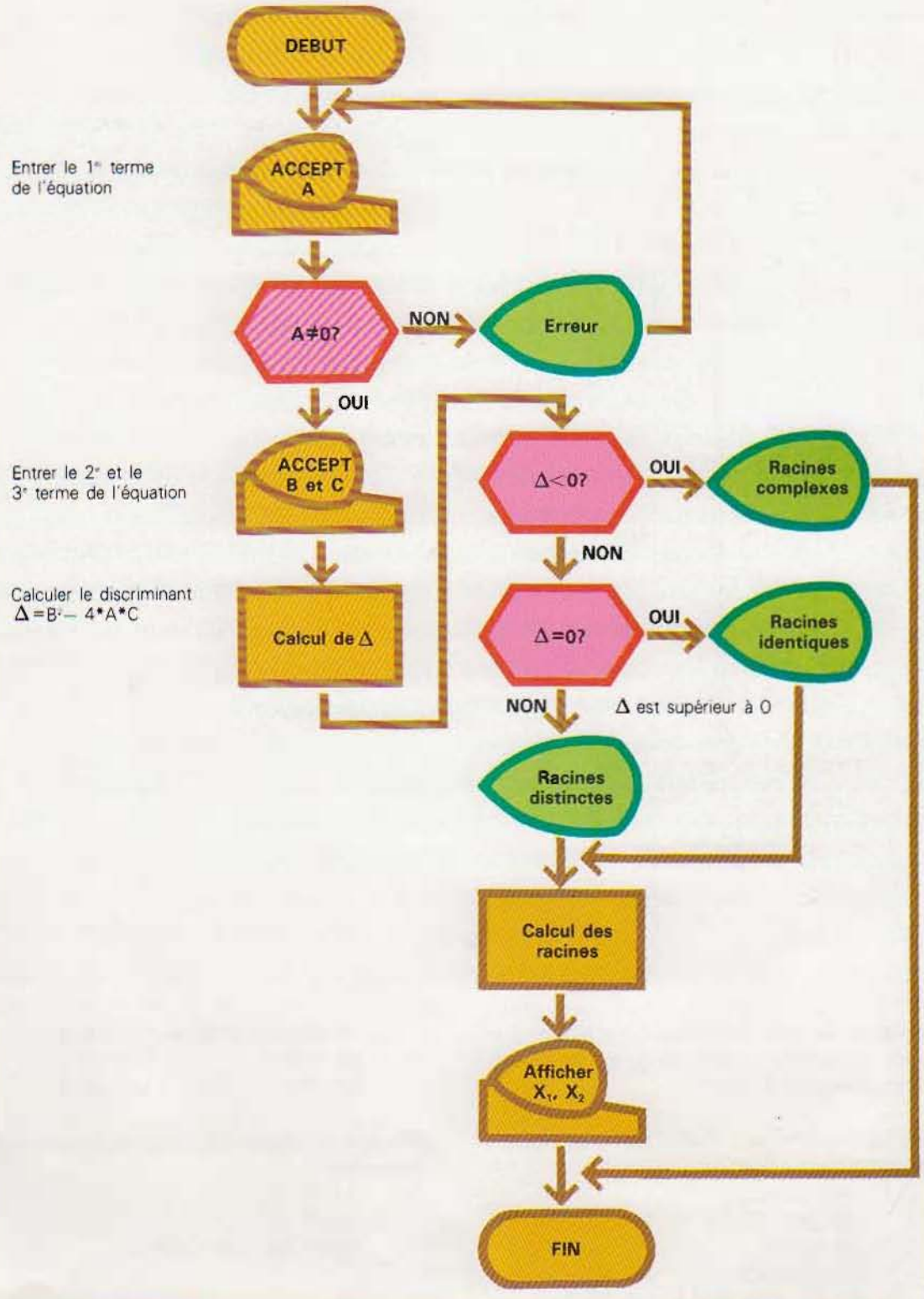
Notons que la valeur de DELTA est calculée en dehors de la proposition pour des raisons de rapidité (un calcul au lieu de trois) ; l'analyse de signe vient après.
Pour conclure, notons qu'une analyse de signe peut se ramener à une analyse de relation en comparant la valeur du champ ou de l'expression à 0.

Analyse de condition. La fonction du niveau 88 a été étudiée lors de la description des noms des champs de la division données. Rappelons-nous, dans ce contexte, que si un champ a été décrit avec une ou plusieurs définitions de niveau 88, le contenu peut être analysé en se référant directement à elles.

FORMAT DE L'INSTRUCTION IF POUR L'ANALYSE DU SIGNE

IF { nom-donnée / expression-arithmétique } IS [NOT] { POSITIVE / NEGATIVE / ZERO }

CALCUL DES RACINES D'UNE EQUATION DU SECOND DEGRE



CALCUL DES RACINES D'UNE EQUATION DU SECOND DEGRE

```
  01  A                          PIC S9(6)V9(3)  COMP.
  01  B                          PIC S9(6)V9(3)  COMP.
  01  C                          PIC S9(6)V9(3)  COMP.
  01  DELTA                      PIC S9(6)V9(3)  COMP.
  01  X1                         PIC S9(6)V9(3)  COMP.
  01  X2                         PIC S9(6)V9(3)  COMP.
*
*
  PROCEDURE DIVISION.
  DEBUT.
      DISPLAY ' CALCUL EQUATION 2.EME DEGRE'
              UPON CONSOLE.
  ACCEPTER-A.
      DISPLAY '* ENTRER PREMIER COEFFICIENT A = '
               UPON CONSOLE.
      ACCEPT A FROM CONSOLE.
      IF  A  IS EQUAL ZERO
          DISPLAY '*** ERREUR ***'
                  ' COEFFICIENT A DOIT ETRE DIFFERENT DE 0'
                  UPON CONSOLE.
                  GO TO ACCEPTER-A.
      DISPLAY '* ENTRER SECOND COEFFICIENT B = '
              UPON CONSOLE.
      ACCEPT B FROM CONSOLE.
      DISPLAY '* ENTRER TROISIEME COEFFICIENT C ='
           UPON CONSOLE.
  ACCEPT C FROM CONSOLE.
  COMPUTE DELTA = B ** 2 - 4 * A * C.
  IF  DELTA IS NEGATIVE
      DISPLAY '* DELTA NEGATIF *
              '* RACINES COMPLEXES *'
              UPON CONSOLE
      GO TO FIN.
  IF  DELTA IS ZERO
      DISPLAY '* DELTA NUL *'
              '* RACINES IDENTIQUES *'
              UPON CONSOLE.
  IF  DELTA IS POSITIVE
      DISPLAY '* DELTA POSITIF *'
              '* RACINES REELLES ET DISTINCTES *'
               UPON CONSOLE.
    COMPUTE X1 = (0 - B - DELTA ** .5) / 2 * A.
    COMPUTE X2 = (0 - B + DELTA ** .5) / 2 * A.
    DISPLAY '** X1 = ' X1
            '     '
            '** X2 = ' X2
            UPON CONSOLE.
FIN.
    DISPLAY 'FIN CALCUL'
            UPON CONSOLE.
    STOP RUN.
```

Les deux exemples qui suivent mettent en évidence l'avantage qui découle du recours à cette procédure du Cobol.

Soit l'enregistrement EMPLOYE ainsi décrit :

```
01 EMPLOYE.
   08 MATRICULE        PIC 9(5).
   05 NOM              PIC X(30).
   05 ETAT-CIVIL       PIC 9.
   88 CELIBATAIRE-F    VALUE 1.
   88 CELIBATAIRE-H    VALUE 2.
   88 MARIE            VALUE 3.
   88 VEUF             VALUE 4.
```

A l'évidence, dans le contexte du programme, l'instruction :

```
IF VEUF
   PERFORM TRAITEMENT.
```

s'interprète plus rapidement que la suivante :

```
IF ETAT-CIVIL = 4
   PERFORM TRAITEMENT
```

A travers les exemples cités, on voit que l'analyse de condition équivaut à vérifier, dans une forme familière, la signification logique de l'événement lié à un contenu déterminé du champ.
L'avantage est encore plus frappant quand ces conditions sont très nombreuses et chaînées en séquence.

Considérons l'opération de lecture d'un fichier de cartes.
Les articles décrits sur chaque fiche sont codés au moyen de deux caractères, une lettre suivie d'un chiffre compris entre 4 et 9.
La validité de chaque carte lue passe par la vérification de toutes les combinaisons possibles.
La solution devient, en revanche, plus simple si on adopte la description suivante, dans laquelle la liste de tous les caractères alphabétiques (VALUE 'A' THRU 'Z') est associée à SERIE-VALABLE et les chiffres allant de 4 à 9 (VALUE 4 THRU 9) à NOMBRE-VALABLE :

```
01 CARTE-ARTICLE.
   05 CODE-ARTICLE.
      10 SERIE-ARTICLE          PIC X.
         88 SERIE-VALABLE       VALUE 'A'
                               THRU 'Z'.
      10 NOMBRE-ARTICLE         PIC 9.
         88 NOMBRE-VALABLE      VALUE 4
                               THRU 9.
```

L'instruction, pour le contrôle du CODE-ARTICLE, s'écrit donc :

```
IF NOT SERIE-VALABLE
   PERFORM ERREUR-SERIE
   GO TO LIRE-CARTES.
IF NOT NOMBRE-VALABLE
   PERFORM ERREUR-NOMBRE
   GO TO LIRE-CARTES.
```

Analyse de classe. Grâce à ce dernier type d'analyse associé à l'instruction IF, il est possible d'examiner si le contenu d'un certain champ est, à un moment déterminé, de classe numérique ou alphanumérique (voir format ci-dessous).

L'analyse nécessite le respect de certaines règles intuitives :
- Dans un champ alphanumérique (PIC X), effectuer un test pour évaluer si son contenu est numérique ou alphabétique et a un sens.

- Inversement, vérifier si un champ alphabétique (PIC A), contenant des nombres, est une erreur logique et syntaxique.

Ces tests sont résumés dans le tableau qui se trouve en bas de cette page.

FORME DE L'INSTRUCTION IF POUR L'ANALYSE DE CLASSE

```
IF nom-donnée  IS [NOT] { NUMERIC    }
                        { ALPHABETIC }
```

ANALYSE DE CLASSE

PICTURE	[NOT] NUMERIC	[NOT] ALPHABETIC
A (Alphabétique)	NON	OUI
9 (Numérique)	OUI	NON
X (Alphanumérique)	OUI	OUI

N'oublions pas que le champ examiné doit être implicitement ou explicitement USAGE DISPLAY.

Emploi des opérateurs logiques. Tous les tests que nous avons examinés jusqu'à présent portent sur un événement unique. Il s'agissait de conditions simples. En fait, l'action à déclencher est souvent liée au résultat de plusieurs analyses subordonnées de différents types et réalisées par un nombre quelconque de variables.
L'enchaînement logique des conditions est obtenu au moyen d'opérateurs logiques propres à l'algèbre booléenne : AND, OR, NOT, (ET, OU, NON).

Quand une décision est subordonnée au test simultané de deux ou de plusieurs événements, l'analyse peut être conduite en utilisant AND avec un seul IF.

La phrase conditionnelle est vraie quand les deux conditions s'avèrent également vraies.
Dans l'exemple :

```
01 CHAMP-1      PIC 9 VALUE 1.
02 CHAMP-2      PIC 9 VALUE 0.
```

Pour vérifier, en cours d'exécution, si le contenu des deux champs est resté le même qu'en entrée, on peut écrire :

```
IF      CHAMP-1 = 1
        AND
        CHAMP-2 = 0
        PERFORM CHAMPS-OK
ELSE
```

L'appel à la procédure CHAMPS-OK est activé seulement si les deux champs vérifient simultanément les conditions demandées. Bien noter qu'il suffit qu'une condition ne soit pas validée pour que le contrôle passe aux instructions subordonnées à la clause ELSE.
On emploie AND très fréquemment pour constituer un fichier des enregistrements d'entrée sur la base de l'appartenance d'une donnée à un intervalle préalablement établi.

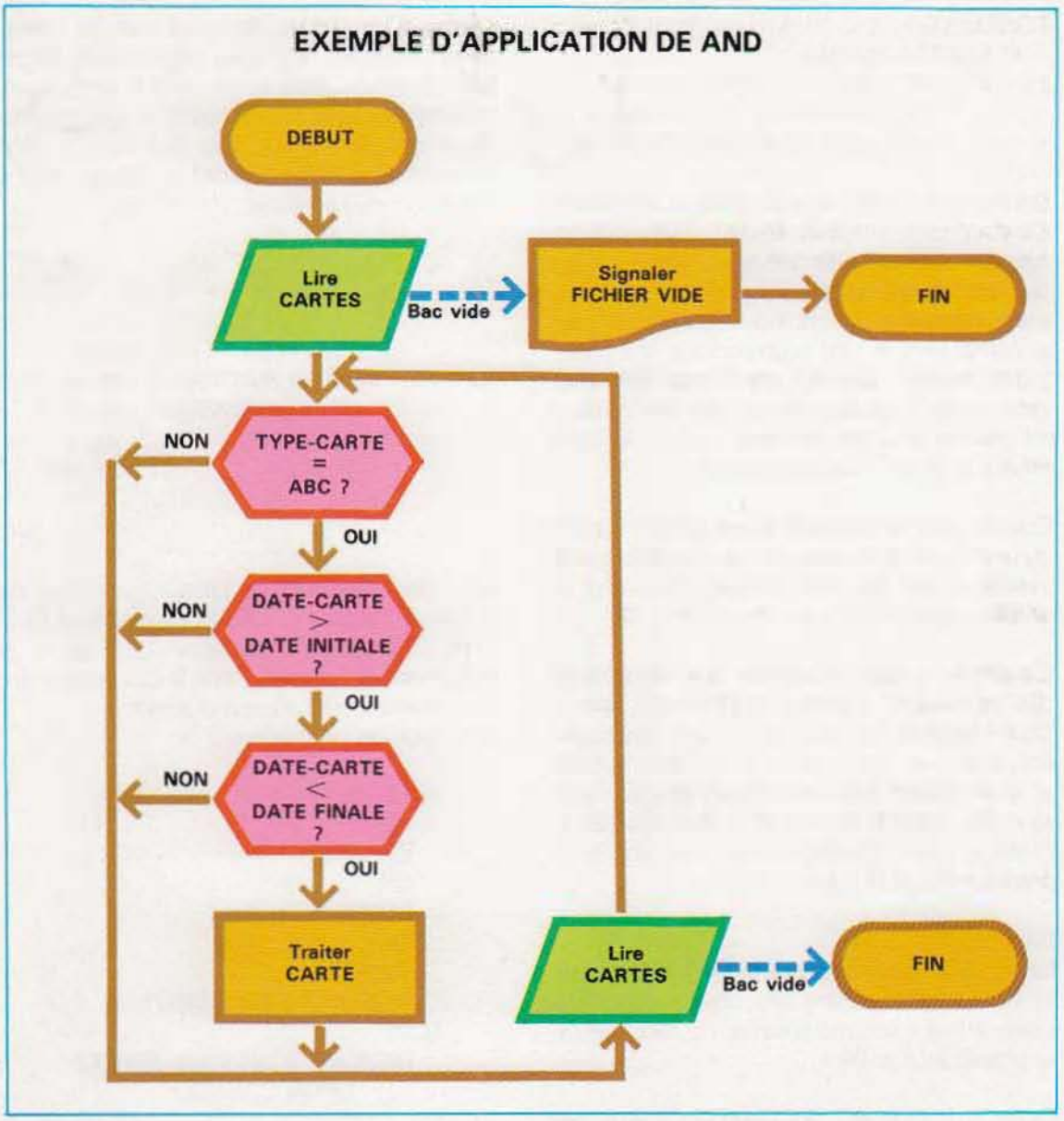
Soit le fichier CARTES suivant :

```
01 CARTE-WS.
   05 TYPE-CARTE              PIC X(3).
      88 CARTES-VALABLE VALUE 'ABC'.
   05 DATE-CARTE.
      10 ANNEE-CARTE          PIC 9(2).
      10 MOIS-CARTE           PIC 9(2).
      10 JOUR-CARTE           PIC 9(2).
   05
```

EXEMPLE D'APPLICATION DE AND

```
PROCEDURE DIVISION.
DEBUT.
    OPEN INPUT CARTES.
PREMIERE-LECTURE.
    READ  CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER
                 CLOSE CARTES
                 STOP RUN.
CONTROLE.
    IF   CARTE-VALABLE
         AND
         DATE-CARTE > DATE-INITIALE
         AND
         DATE-CARTE < DATE-FINALE
         PERFORM CALCUL-CARTE.
    READ  CARTES INTO CARTE-WS
                  AT END
                  CLOSE CARTES
                  STOP RUN.
    GO TO CONTROLE.
*
*
*
```

EXEMPLE D'APPLICATION DE AND



Nous voulons ne traiter que les cartes de type 'ABC' et relatives à la période comprise (extrêmes exclus) entre :

```
01 DATE-INITIALE.
   05 ANNEE-INIT          PIC 99.
   05 MOIS-INIT           PIC 99.
   05 JOUR-INIT           PIC 99.
*
*
01 DATE-FINALE.
   05 ANNEE-FIN           PIC 99.
   05 MOIS-FIN            PIC 99.
   05 JOUR-FIN            PIC 99.
```

La phase de choix et de traitement est détaillée par le programme ci-contre (page 1068) et par l'organigramme ci-dessus.

Il suffit que la carte ne soit pas de type ABC, ou que l'une des deux limites de l'intervalle d'extraction ne soit pas respectée pour que la main soit donnée à l'instruction suivant immédiatement le point qui ferme IF.

Quel que soit le résultat de IF, on passera quand même par cette instruction, car l'opération READ n'est pas assujettie à la clause ELSE. La validité et la clarté de la solution proposée apparaissent mieux dans le détail du paragraphe suivant :

```
CONTROLE.
   IF CARTE-VALABLE
      IF DATE CARTE > DATE-INITIALE
         IF DATE-CARTE < DATE-FINALE
            PERFORM TRAITEMENT-CARTE.
```

Dans ce cas, nous avons eu recours à une série d'instructions IF imbriquées.
Adopter une telle codification, surtout dans des cas moins simples, comporte généralement un risque de moindre lisibilité du programme, ainsi qu'une augmentation des possibilités d'erreur. En effet, les IF imbriqués peuvent mener le programmeur vers des niveaux de plus en plus hiérarchisés, jusqu'à lui faire perdre le fil de l'analyse logique.

Quand, pour entreprendre une action, il suffit qu'une seule parmi une série de conditions, soit vérifiée, il est dès lors possible d'exprimer la phrase conditionnelle en recourant à OR.

L'exemple suivant constitue une application des opérateurs logiques AND et OR utilisés pour recenser les employés d'une entreprise non mariés et nés entre le 31 décembre 1940 et le 1er janvier 1960 inclus. Les données sont stockées dans le fichier de cartes ANAGRAPHIQUE (voir l'organigramme ci-contre et le programme de la page 1072).

Bien noter :
La signification de l'opérateur NOT est intuitive et son utilisation, dans une phrase conditionnelle, entraîne une modification du sens réel de la phrase tout entière.

Cette altération est pratiquement impossible dans le cas de conditions simples, n'indiquant la vérification que d'un seul événement.
Ainsi, la phrase :

```
IF A NOT = B
   PERFORM PREMIERE-ACTION
ELSE
   PERFORM SECONDE-ACTION.
```

ne comporte aucune ambiguïté et sa lecture est immédiate.
A l'inverse, la compréhension d'une phrase, avec une ou plusieurs conditions ne dépendant pas d'autres opérateurs logiques AND et OR, n'est pas immédiate. D'autant que ces opérateurs respectent les règles générales de l'algèbre booléenne. Sans entrer dans le vif du sujet, considérons tout de même l'exemple suivant. Nous devons lire le fichier EMPLOYES dans lequel chaque carte contient le champ relatif à l'état-civil de l'employé.

```
01 EMPLOYE.
   05 .....
   05 .....
   05 ETAT-CIVIL          PIC 9.
      88 CELIBATAIRE-H  VALUE 1.
      88 CELIBATAIRE-F  VALUE 2.
      88 MARIE          VALUE 3.
      88 VEUF           VALUE 4.
   05 .....
      .....
```

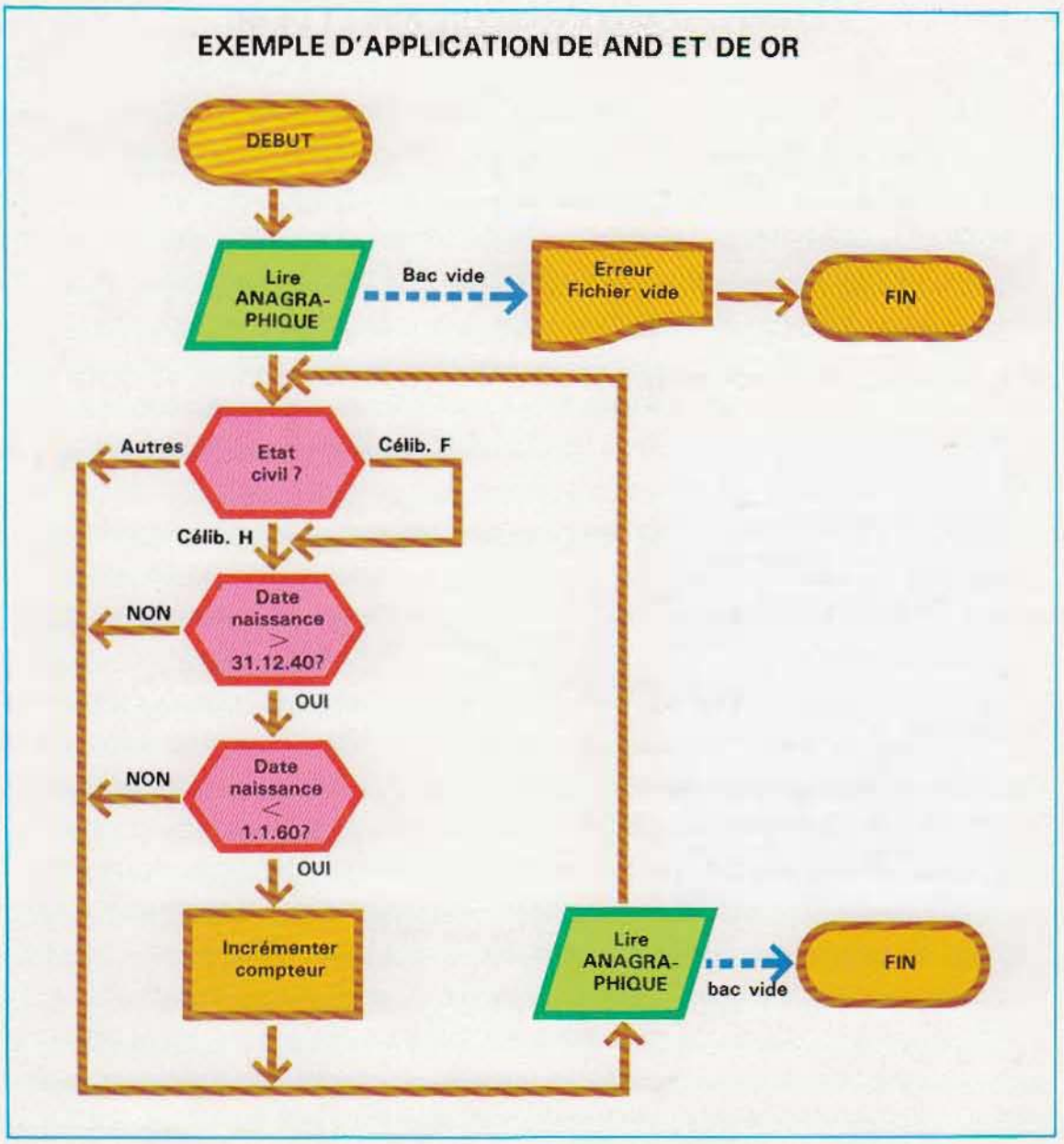
Nous voulons contrôler, pendant la lecture du fichier de cartes, la validité du champ ETAT-CIVIL. S'il contient le chiffre 1,2,3 ou 4, le traitement se poursuit. Dans le cas contraire, il faut émettre un message d'erreur.
En Cobol, on peut écrire :

```
IF CELIB H
   OR
   CELIB F
   OR
   MARIE
   OR
   VEUF
   PERFORM TRAITEMENT
ELSE
   DISPLAY 'ETAT CIVIL ERRONE'
      UPON CONSOLE.
```

Si, pour des raisons de structuration d'un programme, on doit analyser en premier le cas d'erreur, (c'est-à-dire si l'état-civil n'est ni célibataire homme, ni célibataire femme, ni marié, ni veuf), on peut intuitivement penser à codifier les instructions de la façon suivante :

```
IF NOT CELIB H
      OR
   NOT CELIB F
      OR
   NOT MARIE
      OR
   NOT VEUF
```

EXEMPLE D'APPLICATION DE AND ET DE OR




```
        DISPLAY 'ETAT CIVIL ERRONE'
                UPON CONSOLE
    ELSE
        PERFORM TRAITEMENT.
```

En réalité, en adoptant les instructions décrites, toutes les cartes seront signalées comme erronées.
Supposons, en effet, que l'employé décrit dans la carte à traiter, soit un homme célibataire. Dans ce cas, la première condition, pour écarter la carte, n'est pas vérifiée puisqu'il est vrai que l'employé est un jeune homme ; c'est pourquoi, en procédant à l'analyse de la phrase, l'employé ne peut être une célibataire femme et la condition d'erreur est donc vérifiée.
L'analyse de la phrase se poursuit jusqu'au test veuf, mais à ce niveau, trois possibilités d'erreur ont déjà été vérifiés. C'est pourquoi le message 'ETAT CIVIL ERRONE' est anormalement affiché.
Ce qui résume la formule :

NOT (A OR B) = (NOT A) AND (NOT B)

La négation globale d'une série de conditions

EXEMPLE D'APPLICATION DE AND ET DE OR

```
 01 EMPLOYE-WS.
    05  MATRICULE                    PIC 9(3).
    05  NOM                          PIC X(30).
    05  ETAT-CIVIL                   PIC 9.
        88  CELIBATAIRE-F            VALUE 1.
        88  CELIBATAIRE-H            VALUE 2.
        88  MARIE                    VALUE 3.
        88  VEUF                     VALUE 4.
    05  DATE-NAISSANCE.
        10  ANNEE                    PIC 9(2).
        10  MOIS                     PIC 9(2).
        10  JOUR                     PIC 9(2).
    05  ..................
    05  ..................
        10  ..............
*
*
 01 COMPTEUR                         PIC 9(4) COMP VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 DEBUT.
    OPEN INPUT ANAGRAPHIQUE.
 PREMIERE-LECTURE.
    READ ANAGRAPHIQUE INTO EMPLOYE-WS
                      AT END
                      DISPLAY '** FICHIER ANAGRAPHIQUE VIDE' **'
                              UPON PRINTER
                      CLOSE ANAGRAPHIQUE
                      STOP RUN.
 CONTROLE.
    IF  (CELIBATAIRE-F OR CELIBATAIRE-H)
        AND
        DATE-NAISSANCE > 401231
        AND
        DATE-NAISSANCE < 600101
        ADD 1 TO COMPTEUR.
    READ ANAGRAPHIQUE INTO EMPLOYE-WS
                      AT END
                      CLOSE ANAGRAPHIQUE
                      DISPLAY '** EMPLOYES NON MARIES '
                              'DEMANDES = '
                              COMPTEUR
                              UPON PRINTER
                      STOP RUN.
*
*
```

alternatives (OR) peut être exprimée **seulement** comme un produit logique (AND) de négations des conditions composantes.
L'écriture correcte de la seconde formule est donc :

```
IF NOT CELIBATAIRE-H
        AND
   NOT CELIBATAIRE-F
        AND
   NOT MARIE
        AND
   NOT VEUF
        DISPLAY 'ETAT CIVIL ERRONE'
                UPON CONSOLE
ELSE
   PERFORM TRAITEMENT.
```

En conclusion :
Quand plusieurs conditions sont analysées en même temps, il est bon de contrôler leur déroulement en mettant entre parenthèses les groupes simples.
De cette façon, on a le double avantage d'éviter des erreurs et de préserver l'ordre logique de l'analyse.

L'instruction GO TO

Les instructions d'un programme Cobol s'exécutent dans l'ordre de leur écriture par le programmeur, sauf indication contraire de celui-ci, par recours à l'instruction de saut GO TO.
En théorie, un programme Cobol pourrait être construit selon une cascade d'instructions appartenant toutes à un paragraphe unique :

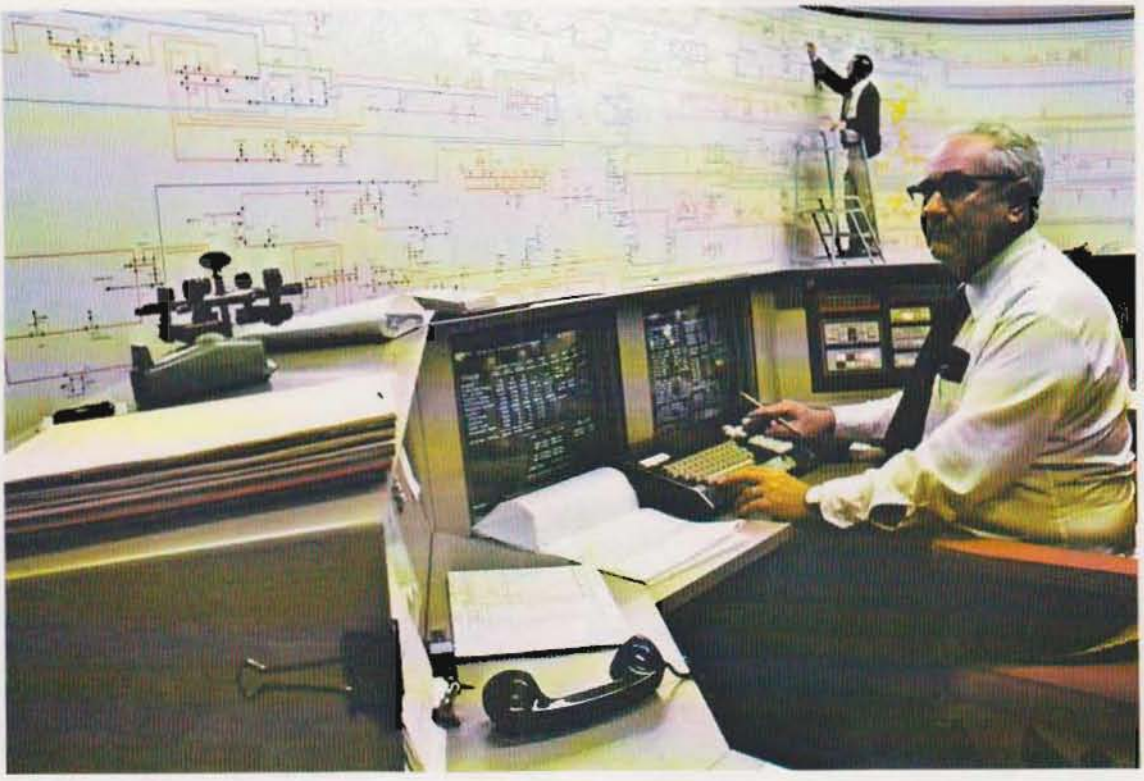
```
PROCEDURE DIVISION.
DEBUT.
    OPEN.....
    READ..... INTO.....
    ADD.....
    MOVE.....
    WRITE.....
    .....
    STOP RUN.
```

Un tel programme ne peut remplir que des fonctions très simples. La répétition d'une ou de plusieurs opérations identiques, comme la lecture d'un fichier et le traitement d'enregistrements, oblige à modifier le déroulement séquentiel du programme en créant des phases itératives sur un certain nombre d'instructions spécialement groupées.
Nous savons qu'un programme Cobol s'organise en paragraphes et en sections.
Cette organisation est particulièrement fonctionnelle quand, associée à l'emploi du verbe PERFORM, elle permet de créer des points de branchement auxquels le contrôle peut être cédé par l'instruction GO TO.
Pour le compilateur, les noms de paragraphes et de sections sont complètement transparents à moins qu'ils ne soient appelés par l'instruction GO TO. La forme la plus simple de l'instruction est :

```
GO TO nom-procédure
```

où nom-procédure comporte au maximum 30 caractères alphanumériques perforés à partir de la marge E (colonne 8) sur la carte.
Le saut du programme d'un point à un autre peut être conditionnel (lors de la vérification d'un événement prévisible) ou inconditionnel.

Tableau de contrôle des circuits de distribution dans une grande centrale électrique.



K. Reese/Marka

Situations qu'on retrouve dans les exemples suivants :

Premier cas

```
ADD 1 TO COMPUTER
IF COMPTEUR > 100
   GO TO FIN
ELSE
   .....
   .....
```

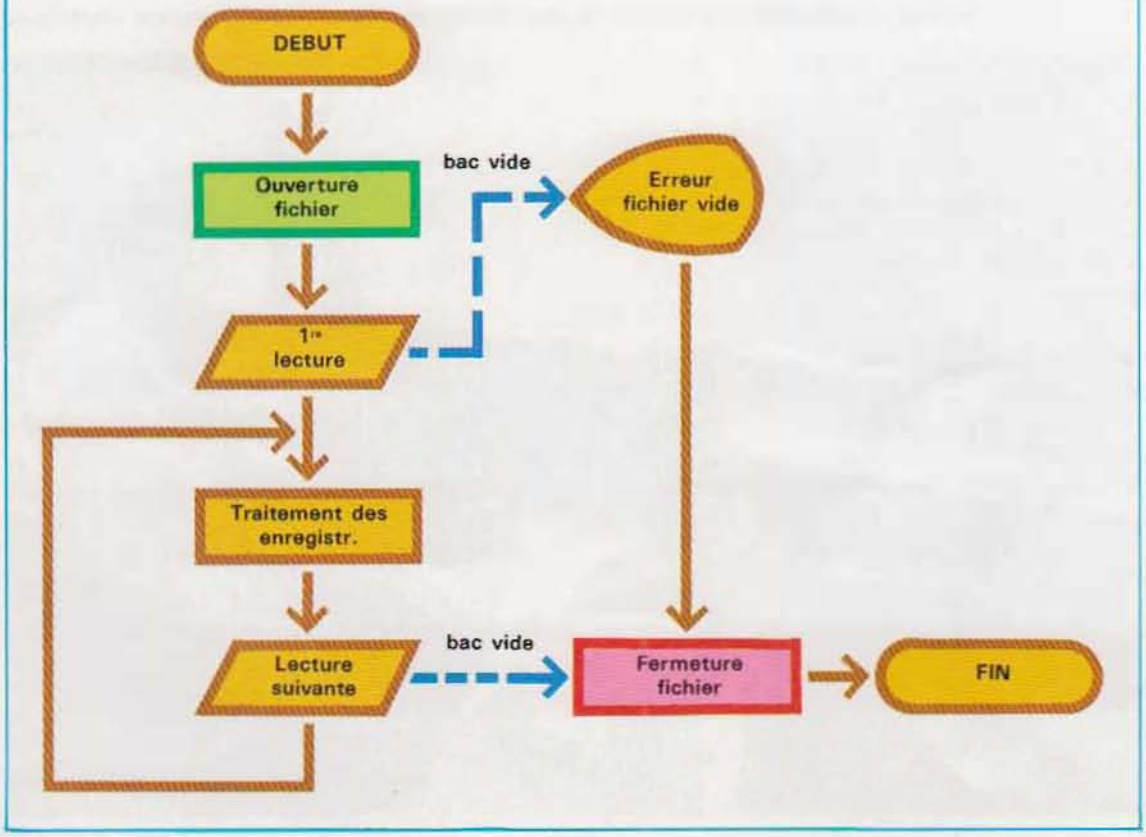
Second cas

```
MOVE A TO B.
ADD 100 TO K.
GO TO PROCEDURE-1.
```

Dans le premier cas, le saut à la procédure FIN n'est effectif que si le contenu de COMPTEUR dépasse 100, alors, que dans le second cas, le saut à la PROCEDURE-1 est réalisé après avoir additionné 100 à la variable K.
Le programmeur est libre d'utiliser les instructions de saut à n'importe quel point du programme, en sachant toutefois qu'elles altèrent sa lisibilité.
Inversement, un recours judicieux et limité de GO TO met en évidence la logique des opérations. C'est le cas des applications de lecture d'un fichier et de contrôle de la condition de fin de fichier.

Le déroulement des opérations de lecture d'un fichier séquentiel, par exemple, respecte le schéma de la figure ci-dessous.
La première lecture (également appelée « lecture hors cycle »), est exécutée une seule fois, immédiatement après l'ouverture du fichier.
Il est évident que, dans ce contexte, le test de la condition « fin de fichier » déclare que le fichier est vide. Evidemment, une telle condition est anormale et l'arrêt du traitement est donc justifiée.
Si le fichier n'est pas vide,le contrôle passe à la première instruction exécutable après le READ et l'enregistrement traité.

CYCLE DE LECTURE D'UN FICHIER SEQUENTIEL



CYCLE DE LECTURE D'UN FICHIER SEQUENTIEL

```
      ..................
      OPEN INPUT FICHIER-ENTREE.
  PREMIERE-LECTURE.
      READ FICHIER-ENTREE INTO ENR-LU
                  AT END
                  DISPLAY '** FICHIER FICHIER-ENTREE VIDE **'
                          UPON CONSOLE
                  GO TO FIN-TRAITEMENT.
  TRAITEMENT-ENREGISTREMENT
      MOVE ..........  TO ..........
      ADD  ..........  TO ..........
                          ..........
                          ..........
      IF   .......... = ............
           ..........
           COMPUTE ....... = ......... + .........
           MOVE    ....... TO.........
      ELSE
           ..........
           ..........
           ..........
      MOVE ENR-LU      TO .........
*
  LIRE-FICHIER-ENTREE.
      READ FICHIER-ENTREE INTO ENR-LU
                  AT END
                  DISPLAY '** FIN LECTURE FICHIER-ENTREE **'
                          UPON CONSOLE
                  GO TO FIN-TRAITEMENT.
*
  FIN-TRAITEMENT.
      CLOSE FICHIER-ENTREE.
      STOP RUN.
```

Cette phase terminée, c'est au tour de la seconde instruction de lecture d'être exécutée autant de fois qu'il y a d'enregistrements dans le fichier.
Le détail de la lecture d'un fichier séquentiel se trouve sous forme de programme en haut de cette page et ci-contre sous forme d'organigramme.

La clause DEPENDING ON. Le GO TO est généralement employé en aval du test de validité d'une événement : selon le résultat, la main est donnée à telle ou telle procédure.
Le cas le plus courant est décrit dans l'exemple suivant :

```
IF A = 1
   ..............
   ..............
   GO TO A-EGAL-1
ELSE
   ..............
GO TO A-DIFFERENT-1.
```

Ainsi, le contrôle est cédé, selon le résultat du test, à l'une des procédures A-EGAL-1 ou A-DIFFERENT-1.
Dans le cas de plusieurs conditions, il serait inélégant d'écrire :

```
IF A = 1
   GO TO PROCEDURE-1.
IF A = 2
   GO TO PROCEDURE-2.
IF A = 3
   GO TO PROCEDURE-3.
   ..............
   ..............
IF A = N
   GO TO PROCEDURE-N.
```

La clause DEPENDING ON, associée à l'instruction GO TO, permet d'aboutir au même résultat, comme il apparait ci-après :

```
GO TO PROCEDURE-1
      PROCEDURE-2
      PROCEDURE-3
      ..........................
      PROCEDURE-N DEPENDING ON A.
```

La séquence d'opérations lancée par le compilateur se déroule de la même manière que dans la formule précédente.
Il faut préciser que :

- le champ spécifié dans la clause DEPENDING ON (A dans notre exemple) doit être numérique ;
- si le contenu du champ spécifié est un nombre différent de 1, 2, ..., N (nombre des procédures énumérées), le contrôle passe à l'ordre suivant immédiatement l'instruction de saut ;
- les noms des procédures doivent être énumérés de telle manière que le nombre contenu dans le champ spécifié (dans la clause DEPENDING ON) détermine leur position dans la liste.

Par exemple, la phrase :

```
GO TO A
      B
      C
      D
      E DEPENDING ON NOMBRE.
```

passera le contrôle aux procédures A si NOMBRE=1, B si NOMBRE=2, et ainsi de suite.

Le verbe PERFORM

Déjà introduite dans certains des exemples présentés, cette instruction permet, avec une seule commande, de traiter toutes celles d'un paragraphe ou d'une section. Cela permet de subdiviser le programme en blocs fonctionnels pouvant être rappelés depuis n'importe quel point du programme. Son emploi, en accord avec la description du Cobol, permet de structurer le programme de manière concise et auto-documentée.
Naturellement, cette dernière caractéristique est liée à la possibilité qu'a le programmeur d'utiliser, pour chaque procédure appelée par PERFORM, des noms désignant directement la fonction.
Si, par exemple, une section du programme calcule la TVA, il est conseillé de la nommer CALCUL-TVA, ce qui est plus explicite qu'une abréviation du type C-T (Calcul TVA).

L'instruction PERFORM est dotée de nombreuses clauses qui seront analysées par la suite. Elle se présente ainsi :

PERFORM nom-procédure

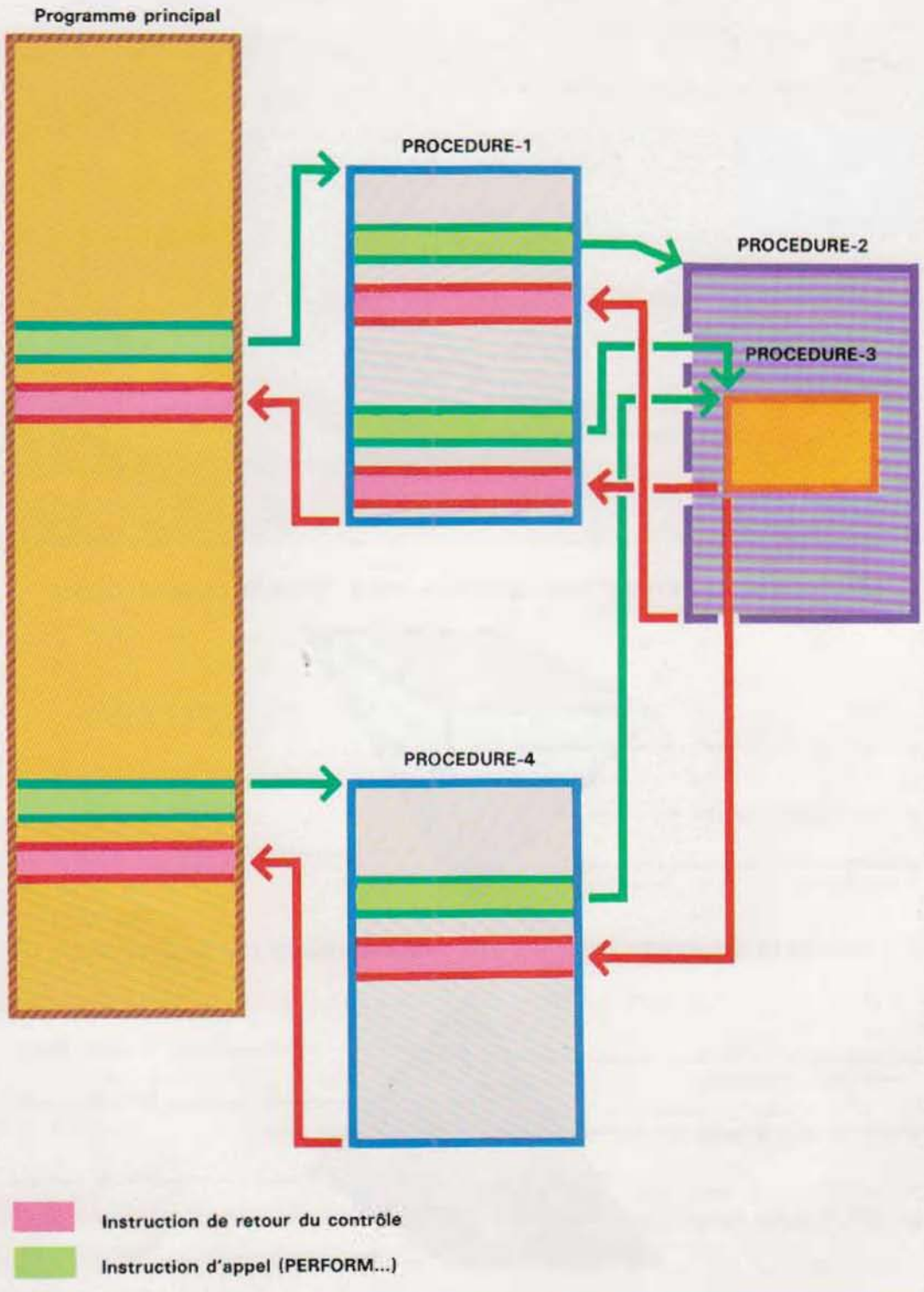
où « nom-procédure » désigne un paragraphe ou une section.
Le diagramme ci-contre montre comment le programme exécute toutes les instructions contenues dans une procédure quand celle-ci est appelée par un PERFORM.
On y voit également comment le contrôle des opérations passe du programme principal aux procédures appelées avec retour à la première instruction à exécuter immédiatement après le verbe d'appel.
Notons qu'une procédure peut contenir l'appel à une autre procédure à condition que celle-ci soit entièrement externe ou interne à la procédure appelante.
De plus, une procédure peut être appelée autant de fois que nécessaire et depuis un point quelconque du programme.
Dans le diagramme fonctionnel, la PROCEDURE-3 faisant partie de PROCEDURE-2 apparaît néanmoins dans d'autres procédures. Supposons, pour simplifier, que la fonction de PROCEDURE-3 soit remplie par l'ensemble des instructions énumérées ci-dessous et comprises entre le nom de paragraphe PARAGRAPHE-1 et le nom de paragraphe PARAGRAPHE-4 :

```
PROCEDURE-1.
        MOVE A TO B.
        ..............
        ..............
PARAGRAPHE-2.
        ..............
        ..............
PARAGRAPHE-3.
        ..............
        ..............
        ADD 1 TO K.
PARAGRAPHE-4.
        ..............
        ..............
```

PROCEDURE-3

Il est possible d'appeler, avec une seule instruction, toutes celles indiquées par l'accolade en écrivant simplement :

APPEL DE PROCEDURE PAR L'INTERMEDIAIRE DE PERFORM

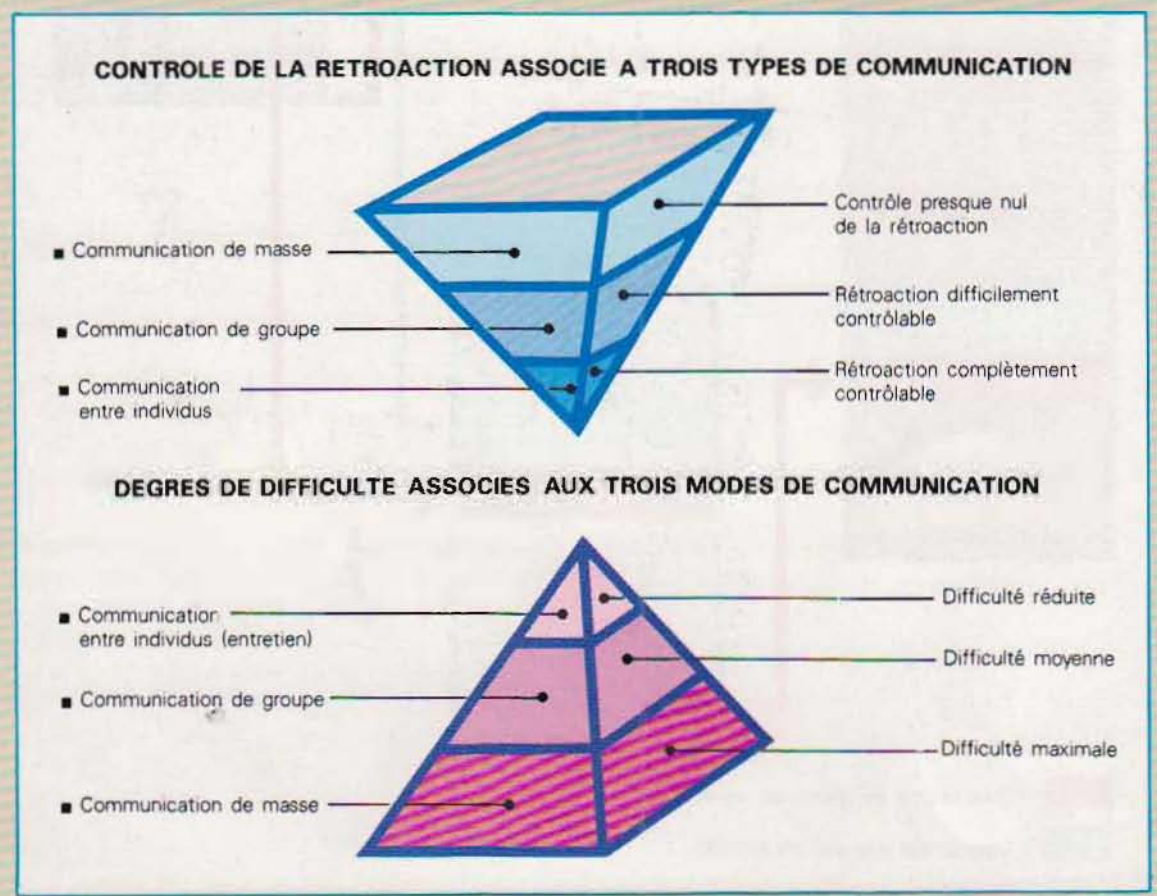


Le dialogue homme-machine

Pensez à l'une de vos journées et analysez le nombre de fois où vous avez eu l'occasion du matin au soir de :

- *parler avec un simple interlocuteur* (communication entre individus) ;
- *parler à un groupe de personnes* (communication de groupe) ;
- *parler face à un média, comme la radio ou la télévision* (communication de masse).

Evidemment, à moins d'être un super président, la plupart de vos rencontres seront de type interpersonnel.
Cet aspect des choses fait ressortir un ordre de fréquence d'utilisation de la communication nettement en faveur des entretiens entre simples personnes.
On peut classer les difficultés de communication selon une échelle de valeurs progressives (voir schéma ci-dessous).
Les gens ont donc, pour la plupart, ***l'habitude du dialogue avec des individus.*** *Par contre, ils se sentent extrêmement démunis face à un groupe et, pire encore, quand il s'agit de passer par l'intermédiaire d'un dispositif audio ou vidéo (enregistrement, passage à la radio ou à la télévision).*
La panique éprouvée par les sens entraîne une prestation de mauvaise qualité, bien moins bonne que si cela s'était passé en petit comité et, surtout, hors de la présence d'appareils intimidants.
Mais pourquoi est-il difficile de communiquer en groupe ?
La raison en semble la réduction progressive du contrôle sur la communication de retour (rétroaction ou feedback).
Le phénomène s'explique par ce que l'on peut appeler la théorie de la chauve-souris. Cet animal aveugle possède une sorte de radar qui lui sert à repérer les obstacles situés sur son passage.
Il s'agit là d'un système d'analyse de signaux de retour.



L'homme se comporte de la même manière au cours d'un entretien.
Observez attentivement votre interlocuteur : vous percevrez toute une série de signaux (mouvements des yeux, de la bouche, des mains) qui révèlent l'attention, l'ennui, la hâte, la stupéfaction.
Par contre, on se rend tout de suite compte que quelqu'un écoute : ses yeux sont bien fixes et sa tête immobile, contrairement à celui qui s'ennuie et qui essaie de fixer son attention en tambourinant des doigts sur la table, en regardant souvent sa montre, et qui s'oublie même à bailler...
Ce type de **contrôle des signaux de retour** peut aussi être effectué dans la communication de groupe, même si le degré de difficulté augmente à cause du nombre des interlocuteurs.
En revanche, le phénomène de **feed-back** disparaît complètement face à un micro ou à une caméra. C'est pourquoi il est souvent si angoissant de se retrouver devant un appareil sans réaction.
Le même phénomène se produit face au clavier et à l'écran d'un ordinateur.
A travers ces signaux, nous recherchons d'abord des « renforts » et un réconfort. Finalement, nous demandons inconsciemment de l'aide.
Une des fonctions fondamentales de la communication est, en effet, de soulager de l'anxiété. En effet, en plus de plaire, l'homme a besoin de communiquer.
Richard Stevens répertorie huit fonctions de base autour desquelles l'activité de relation humaine se développe. Nous communiquons :

- pour accomplir ou pour atteindre quelque chose (fonction instrumentale),
- pour dicter un comportement (fonction de contrôle),
- pour découvrir ou expliquer quelque chose (fonction d'information),
- pour transmettre notre intérêt pour quelque chose (fonction de stimulation),
- parce que la situation l'exige (fonction liée au rôle),
- pour exprimer nos sentiments ou pour nous imposer d'une certaine manière (fonction d'expression),
- pour montrer notre plaisir à être en compagnie (fonction de contact social),
- pour approfondir un problème ou pour nous soulager d'une préoccupation (fonction de soulagement de l'anxiété).

Trois, au moins, des fonctions décrites (expression, contact social et soulagement de l'anxiété) mettent clairement en évidence la nécessité des échanges avec les autres. Voyons maintenant comment Stevens illustre ces trois fonctions.

Fonction d'expression. Une femme se confie à une amie, un homme relève avec colère l'erreur d'un collègue, un amoureux chuchote des paroles d'amour... La communication est un procédé qui permet d'exprimer notre façon d'être : elle est spontanée et authentique ou soigneusement construite pour atteindre un but désiré, par exemple, pour faire impression sur un employeur potentiel pendant un entretien. La fonction auto-expressive de la communication peut opérer de façon dissimulée. Ce qui pourrait n'être qu'un échange fortuit d'informations, pendant une réception, devient dans la réalité une tentative de briller par l'esprit et l'intelligence. Une discussion politique peut être, face à un interlocuteur, l'occasion de manifester sa supériorité sur l'autre.
Le style de la communication, le langage employé, les paroles utilisées constituent une manière d'affirmer **l'appartenance à un groupe.** De nombreuses sous-cultures (groupes professionnels par exemple) ont leur jargon spécifique qu'il faut connaître pour être accepté par le groupe.

Fonction de contrat social. La communication peut être une fin en soi. Comme pour toutes les caractéristiques humaines, il y a, même dans ce cas, des différences d'un individu à un autre.
La plupart des gens éprouvent (plus ou moins) le désir d'être en compagnie d'autres personnes pour le simple plaisir d'être ensemble. Nous apprécions et recherchons davantage le contact avec des personnes qui ont des comportements et des expériences semblables aux nôtres avec lesquelles une interaction s'établit.
Les expériences psychologiques, d'une part, ainsi que les récits de naufragés ou d'ermites,

de l'autre, montrent combien l'isolement social produit souvent des résultats désastreux.

Fonction thérapeutique. Il est amplement démontré que, lorsqu'un individu se trouve dans une situation qui provoque **l'anxiété,** il tente d'établir le contact avec les autres. Le psychologue américain Stanley Schachter a trouvé, par exemple, que certaines étudiantes qui croyaient devoir se soumettre à des tests psychologiques à base de « secousses » électriques douloureuses, ont préféré (contrairement à un autre groupe qui ignorait la nature du remède) attendre leur tour en même temps que d'autres. Ceci ne se produisait, naturellement, que si les camarades en attente se trouvaient dans la même situation. Schachter (1959) la résuma ainsi :
« Le malheur ne se contente pas de n'importe quelle compagnie, il a besoin de la compagnie d'autres malheureux ».
Il est clair que la communication nous est utile pour échanger une série complexe de messages dont beaucoup sont riches en implications secondaires. Ce n'est pas par hasard que Lévi-Strauss définit la société comme un système d'individus et de groupes humains qui communiquent entre eux. La **communication** n'est pas seulement un système de langage, elle est aussi un **procès de personnes.**
Nous nous sommes rendu compte combien la communication est utile aux contacts humains pour survivre sans trop d'angoisse. Nous avons enfin vu comment ces rapports, même pour les habituels échanges interpersonnels qui privilégient les contacts individuels (plutôt que les contacts à travers un média), altèrent en fait le dialogue entre individus.
Mais quand le **dialogue** n'a pas lieu entre deux personnes mais **entre un homme et une machine**, que se passe-t-il ?
Un nouveau type de dialogue naît. Un dialogue spécial fait d'un côté d'humanité et de l'autre de rapidité d'exécution quand la machine est un ordinateur. Tout va bien quand l'homme, face à l'ordinateur, demande des solutions rapides sur la base des programmes que lui-même à écrit. Cependant, les choses se gâtent quand l'homme ne demande pas uniquement de la rapidité mais également un autre type d'aide. Bref un dialogue, c'est-à-dire un processus d'échange, de relation, de sentiments, un moyen de **fuir la solitude et la peur...**
Il n'est pas légitime d'attendre de l'aide, au sens psychologique du terme, de la part de l'ordinateur, mais cela se produit souvent. De là, les questions angoissées de nombreux parents : passer un long temps face à la machine, n'est-ce pas nuisible au développement intellectuel de l'enfant ? Ne va-t-il pas se rendre dépendant de la machine ? Ne s'agit-il pas d'une aide apparente qui masque un frein au développement de ses capacités mentales ?
Tout d'abord, chaque génération devrait avoir le bon sens de ne pas trop intervenir dans le vécu quotidien de la génération suivante et de ne pas trop chercher à la gérer avec son propre étalon mental. Pour une raison simple : cette génération **N'A PAS** les mêmes besoins que nous.

Examinons d'abord un premier élément : LE TYPE DE RAPPORT DANS LES DIALOGUES.
La quantité de dialogue entre personnes sera en voie de diminution au profit des dialogues entre l'homme et l'ordinateur. Celui qui prétend obtenir le même type de communication avec un ordinateur que dans le dialogue humain vit en dehors du temps. Aujourd'hui, l'homme doit apprendre à dialoguer avec la machine sans se sentir frustré.

L'un des exemples les plus simples est celui du téléphone. Ajourd'hui, on fait tout avec le téléphone ; on a presque l'impression de ne plus pouvoir vivre sans lui. Et pourtant, il y a quelques décennies seulement, le téléphone était considéré comme une espèce de monstre dont il fallait se méfier. Aujourd'hui, beaucoup d'organisations d'assistance téléphonique sauvent des vies humaines ou, à tout le moins, libèrent les dépressifs de l'angoisse.
Mais on dira qu'il se trouve, à l'autre bout, une **présence humaine.** C'est vrai. Et le programme avec lequel nous dialoguons ou jouons aux échecs n'est pas, en termes psychologiques, de même nature. Mais, dans ce rapport, lequel doit changer d'entre la machine et l'homme ? Si, à la machine, nous demandons chaleur humaine, sentiments, aide psychologique, est-ce de sa faute si nous ne les obtenons pas ? Non, n'est-ce pas ? On ne devrait, logiquement, jamais attendre d'une chose ce qu'elle ne contient pas par nature.

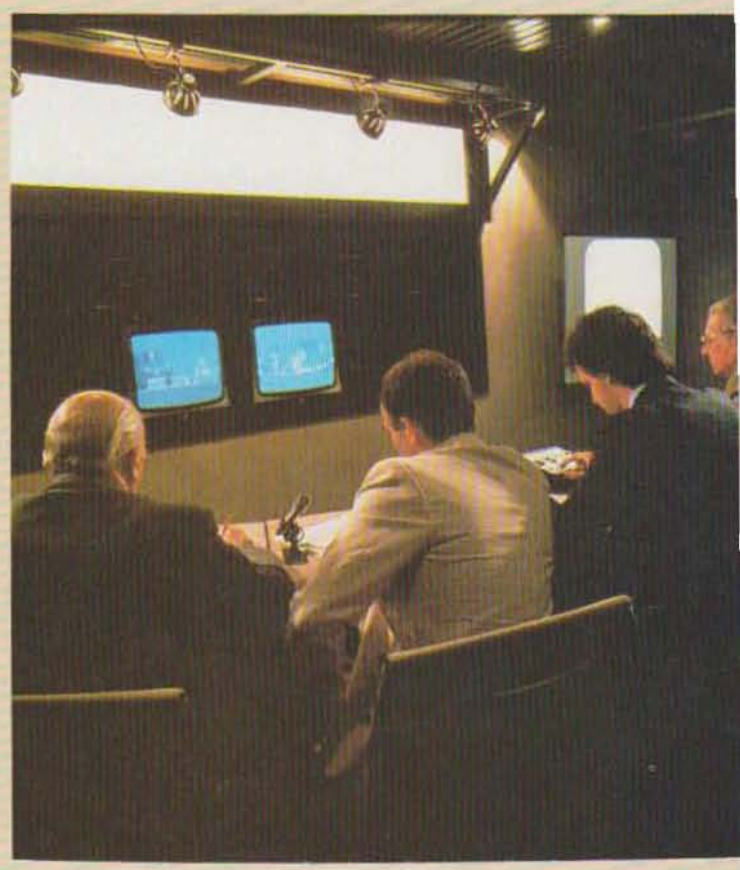
On a vu, dans le précédent numéro, comment le dialogue avec l'ordinateur était effectivement chargé et comment s'opéraient des phénomènes de transfert.

L'ordinateur finit donc par devenir un test d'équilibre psychologique. Si nous ne nous sentons pas bien avec les machines, peut-être est-ce parce que nous ne le sommes pas avec nous-mêmes. Peut-être aussi avons-nous besoin de relations d'aide, mais cela ne concerne pas l'ordinateur.

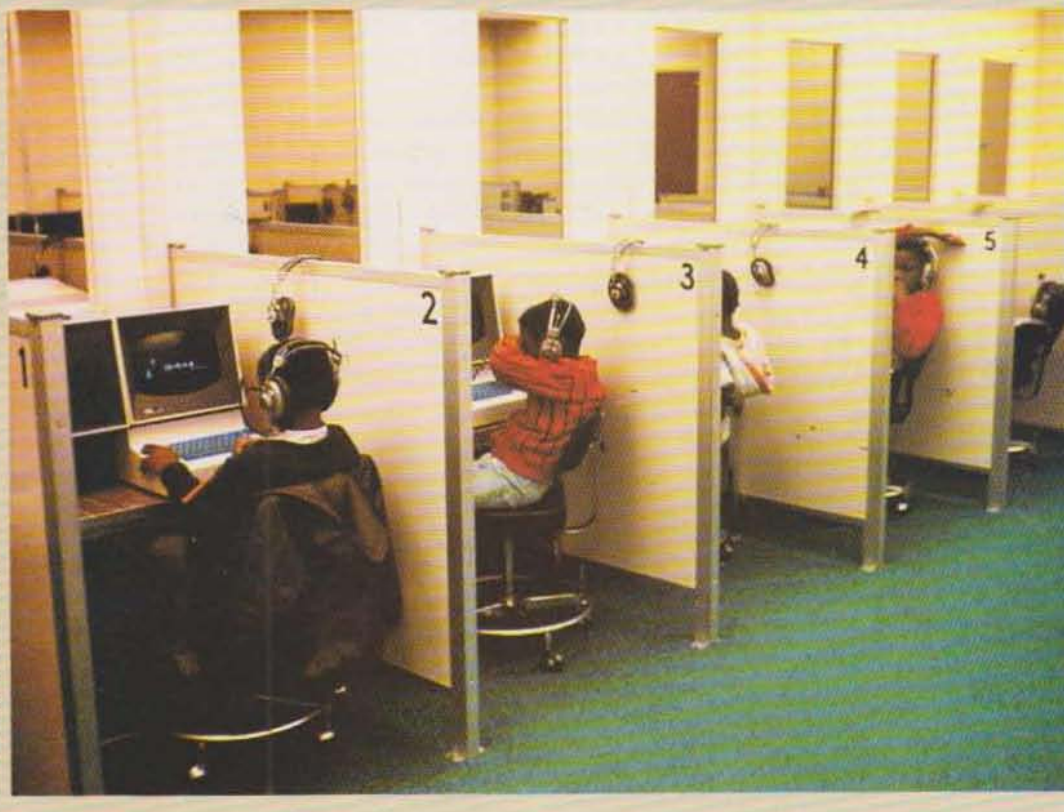
La machine est un ***instrument*** *qui, par définition est* ***neutre.*** *Le couteau sert à couper le pain et à tuer les adversaires : toutes choses ne dépendant pas de l'instrument mais de l'usage que l'on en fait. Nous pourrions donc affirmer qu'une personne équilibrée, indépendante d'une aide extérieure, peut sereinement « dialoguer » avec l'ordinateur.*

Les ***psychanalystes*** *freudiens savent combien la « neutralité » de l'analyse apparaît comme une forme d'hostilité au cours des premières séances. « Vous ne participez pas... Vous restez là à m'écouter parler... Vous moquez-vous de moi ?... »*

Nous demandons donc à nos interlocuteurs une intervention active. La neutralité (mais la



SIP



J. Launois/Grazia Neri-Black Star

La difficulté de communiquer réduit souvent la capacité des étudiants. L'enseignement assisté par ordinateur (EAO) aide à surmonter la barrière du rapport étudiant-enseignement (photo ci-contre). Dans ces conditions, la machine étant un moyen neutre, le contrôle de la rétroaction est total. Les solutions novatrices comme la téléconférence (en haut) introduisent une interface életronique qui abolit les distances.

psychanalyse n'est pas un dialogue) nous irrite jusqu'à ce que nous comprenions que la réponse à nos problèmes n'est pas dans l'autre mais en nous.
Ce n'est ni chez l'analyste ni dans l'ordinateur que se trouve la solution, même s'il est facile de ***transférer notre angoisse sur ces « passifs ».***
Le type de langage adopté à une grande importance pour l'individu qui utilise l'ordinateur avec une approche psychologique correcte et équilibrée.
Il semble aujourd'hui normal que les machines s'adaptent à l'homme en termes ergonomiques mais aussi en termes de « capacité expressive ». Par cette expression, on ne doit pas seulement entendre amélioration linguistique (pourtant indispensable dans le dialogue homme-machine), mais aussi une forme de communication directe avec l'homme, et surtout, un échange fait sur un mode neutre et évitant toute occasion d'irritation.
Les couleurs et les sons constituent, avec les mots, un univers qui doit encore être complètement exploré dans la nouvelle approche homme-machine.
En général, la tendance à assimiler la communication à des faits visuels et sonores, confère une plus grande importance à l'aspect linguistique et à la langue écrite.
L'avènement de ***l'écriture informatique*** *provoque déjà une révolution comparable à celle de Gutenberg avec l'utilisation des caractères mobiles d'impression, ce qui a développé la communication entre des personnes éloignées à un point inimaginable auparavant.*
La communication écrite devra être plus concrète, plus concise. Il sera nécessaire d'apprendre à transmettre l'émotion sans nécessairement recourir à des flots de termes lyriques.
Comment ? La voie est déjà tracée par les techniciens des communications écrites, qu'il s'agisse des ***journalistes*** *ou des rédacteurs de textes* ***publicitaires.***

Voici quelques conseils pratiques pour rendre la communication écrite plus concrète, plus concise et de ce fait, mieux adaptée à l'ère électronique :

- Recourir à des mots courts. *Selon Rudolph Flesh, un texte de 100 mots écrits en anglais ne devrait pas dépasser 200 syllabes.*
- Construire des phrases brèves : *sujet-verbe-attribut avec, toujours selon la règle de lisibilité de Flesh, une moyenne d'environ 15 mots par phrase.*
- Donner des informations utiles *en se demandant constamment ce que le lecteur retient de cette affirmation.*
- Ne pas essayer de dire trop de choses *en même temps. Quand on en dit trop, rien ne passe.*
- Expliquer avec des exemples bien choisis : *les exemples éclairent beaucoup mieux que les définitions détaillées.*
- Placer l'exemple avant la définition. *Si l'on doit expliquer l'effet* ***synergique,*** *on peut commencer avec un exemple : « Deux plus deux égal cinq. La synergie... » Dans la communication écrite, on procède généralement de manière inverse : définition et ensuite exemple ; il s'ensuit qu'il faut tout relire du début puisqu'habituellement la définition est comprise après avoir lu l'exemple.*
- Anticiper les conclusions *à la manière d'un compte-rendu journalistique : d'abord la conclusion du fait, ensuite sa chronologie.*
- Répéter pour augmenter la mémorisation. *Afin d'éviter de créer une réaction négative (« je l'ai déjà lu... »), il est nécessaire de répéter selon une autre formulation. On laisse ainsi le temps au lecteur de comprendre, tout en continuant à éveiller son attention et, éventuellement, en attirant sa réflexion sur un point particulier.*

Et si, maintenant, nous ***récapitulions*** *tout ce que nous avons appris ?*

Comme outil, l'ordinateur est neutre. Ne lui demandons pas une aide psychologique, comme nous faisons avec des personnes. Le dialogue homme-machine a d'autres finalités. Acceptons donc la nouvelle réalité : moins de communication interpersonnelle pour fuir son angoisse et plus de dialogue orienté vers l'échange d'information.

Enrico Cogno

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-3
```

Dans cette forme, PERFORM passe la main à la première instruction exécutable de PARAGRAPHE-1 (MOVE A TO B) immédiatement après l'exécution de la dernière instruction de PARAGRAPHE-3 (ADD 1 TO K.).
Le recours à une telle forme de PERFORM présente deux inconvénients majeurs qui poussent à en déconseiller l'utilisation.
La première erreur serait d'écrire :

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-4
```

en croyant n'envoyer, en exécution, que les instructions comprises entre les mots PARAGRAPHE-1 et PARAGRAPHE-4. En rélité, seront également concernées toutes les instructions de PARAGRAPHE-4. On peut cependant éviter ce problème avec un peu d'expérience.
Le second inconvénient a trait à la manipulation du programme, ce qui est bien plus grave.
Supposons, en effet, qu'au cours de mises à jour successives, il faille modifier le programme de la façon suivante :

```
PARAGRAPHE-1.
     MOVE A TO B
     .......
     .......
PARAGRAPHE-2.
     .......
     .......
PARAGRAPHE-3.
     .......
     .......
RETOUR.
     .......
     .......
     IF A NOT=C
        .......
        .......
          GO TO RETOUR.
     .......
     .......
     ADD 1 TO K.
PARAGRAPHE-4.
     .......
     .......
```

On a créé, entre PARAGRAPHE-3 et PARAGRAPHE-4, le nouveau paragraphe

Le portable HP110 est un vrai micro-ordinateur et ne pèse que 4,10 kg.

Hewlett Packard

RETOUR, ce qui est justifié par la nécessité d'effectuer un contrôle sur la variable A.
Dans ce cas, l'instruction :

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-3
```

restitue le contrôle au programme principal dès que l'on rencontre le paragraphe RETOUR, rétablissant le déroulement du traitement. Il faut donc corriger toutes les instructions d'appel en les modifiant ainsi :

```
PERFORM PARAGRAPHE-1 THRU RETOUR.
```

Les corrections demandées peuvent être importantes, selon le nombre de modifications implémentées et la complexité du programme.
En conclusion, il est recommandé de structurer le programme en sections prenant complètement en charge la fonction à éxécuter.
Les appels doivent toujours être écrits selon la forme suivante :

```
PERFORM nom-section.
```

Dans le cas où l'on désire adopter l'appel de plusieurs procédures avec la proposition THRU, on peut recourir à l'instruction EXIT.

L'instruction EXIT. Elle n'a, en réalité, aucun effet, si ce n'est de pallier, en partie, le second inconvénient de PERFORM. Il suffit que le programme soit structuré ainsi :

```
PARAGRAPHE-1.
      MOVE A TO B.
      . . . . . .
      . . . . . .
PARAGRAPHE-2.
      . . . . . .
      . . . . . .
PARAGRAPHE-3.
      . . . . . .
      . . . . . .
      ADD 1 TO K.
PARAGRAPHE-3-EX.
      EXIT.
PARAGRAPHE-4.
```

L'instruction :

```
PERFORM PARAGRAPHE-1
THRU    PARAGRAPHE-3-EX.
```

envoie alors EXIT en exécution, comme dernière instruction.
Le contrôle est ensuite restitué à la partie appelante.
De ce fait, n'importe quel paragraphe inséré à l'intérieur de la procédure appelée est exécuté sans interruption.

Exécutions itératives de la même procédure. Grâce à la clause TIMES, l'instruction PERFORM permet de répéter l'exécution de la même procédure ou du même groupe de procédures (THRU).
Le tout se trouve résumé dans le format de PERFORM (voir ci-contre, haut de la page 1085), complété par les clauses que l'on vient de décrire.
Soit le calcul de la somme des nombres entiers de 1 à 1000. Le programme s'écrit :

```
PROCEDURE DIVISION.
DEBUT SECTION.
MISE-A-ZERO.
     MOVE ZEROES TO NOMBRE
                    TOTAL.
INCREMENTER-NOMBRE.
     ADD 1 TO NOMBRE.
     IF NOMBRE GREATER 1000
       DISPLAY'*** TOTAL='
               TOTAL
               '***'
               UPON CONSOLE
       STOP RUN.
     ADD NOMBRE TO TOTAL.
     GO TO INCREMENTER-NOMBRE.
*
```

On parvient au même résultat avec l'instruction suivante :

```
PROCEDURE DIVISION.
DEBUT SECTION.
MISE-A-ZERO.
     MOVE ZEROES TO NOMBRE
                    TOTAL.
     PERFORM SOMME 1000 TIMES.
     DISPLAY'*** TOTAL='
             TOTAL
             '***'
             UPON CONSOLE.
```

PREMIER FORMAT DE L'INSTRUCTION PERFORM.

```
PERFORM nom-procédure-1
        [THRU nom-procédure-2]

[ { nombre-entier-1 }      TIMES ]
  { nom-donnée—1    }
```

```
        STOP RUN.
*
   SOMME-SECTION.
   SOMME-PARAGR.
        ADD 1 TO NOMBRE.
      ADD NOMBRE TO TOTAL.
   SOMME-EX. EXIT.
```

La clause UNTIL. La possibilité de connaître a priori le nombre de fois qu'un groupe d'instructions doit être exécuté est assez rare. Surtout que le cycle d'itération utilise un champ numérique imposé par le programme et non une constante fixe comme dans le dernier exemple. Il est plus fréquent de faire intervenir l'arrêt de l'itération au moment où un certain événement se vérifie.
Nous voulons, par exemple, additionner les premiers entiers jusqu'à obtenir un résultat supérieur à 3425.
Notons que la valeur, pour stopper l'itération, peut aussi être contenue dans un champ numérique défini dans WORKING-STORAGE.
Le résultat désiré s'obtient avec les instructions suivantes, qui intègrent l'utilisation de la clause UNTIL (jusqu'à ce que) :

```
   PROCEDURE DIVISION.
   DEBUT SECTION.
   MISE A ZERO.
      MOVE ZEROES TO NOMBRE.
                     TOTAL
                     NOMBRE-SOMMES.
      PERFORM SOMME
              UNTIL
              TOTAL GREATER 3425.
      DISPLAY '*** NOMBRES-SOMMES='
              NOMBRES-SOMMES
              UPON CONSOLE.
      DISPLAY'***TOTAL FINAL='
              TOTAL
              UPON CONSOLE.
         STOP RUN.
*
   SOMME SECTION.
   SOMME-PARAGR.
      ADD 1 TO NOMBRE
               NOMBRES-SOMMES.
      ADD NOMBRE TO TOTAL
   SOMME-EX. EXIT.
```

La clause VARYING... FROM... BY. Cette clause étend la puissance de la précédente en conditionnant les itérations lors de la vérification d'une condition (UNTIL).
Son intérêt vient de ce qu'elle est dotée d'une plus grande flexibilité.
Une telle souplesse est due à la possibilité de varier (VARYING) la valeur d'une donnée quelconque pendant tout le déroulement de la procédure.
Cette valeur est incrémentée à chaque itération d'un pas déterminé (BY) en partant d'une valeur initiale (FROM) (voir format, haut de la page suivante).
Une application typique de ce format concerne la gestion des tableaux, surtout en phase de chargement et de recherche séquentielle.
On étudiera plus longuement cet aspect par la suite.

L'instruction STOP. La forme de cette instruction, déjà rencontrée dans les exemples précédemment étudiés, est la suivante :

```
STOP RUN.
```

A noter qu'il existe une seconde forme qui permet de suspendre momentanément le traitement.

SECOND FORMAT DE L'INSTRUCTION PERFORM

```
PERFORM     nom-procédure-1
            [THRU nom-procédure-2]
            VARYING     nom-donnée-1
            FROM    { constante numérique-1 }
                    { nom-donnée-2          }
            BY      { constante numérique-2 }
                    { nom-donnée-3          }
            UNTIL       condition
```

On envoie, à ce moment-là, un message à la console du système. L'interruption est alors gérée par l'opérateur et le traitement redémarre en fonction de la réponse fournie, à partir de la première instruction suivant l'instruction STOP.

La ligne :

STOP'ERREUR DE CALCUL

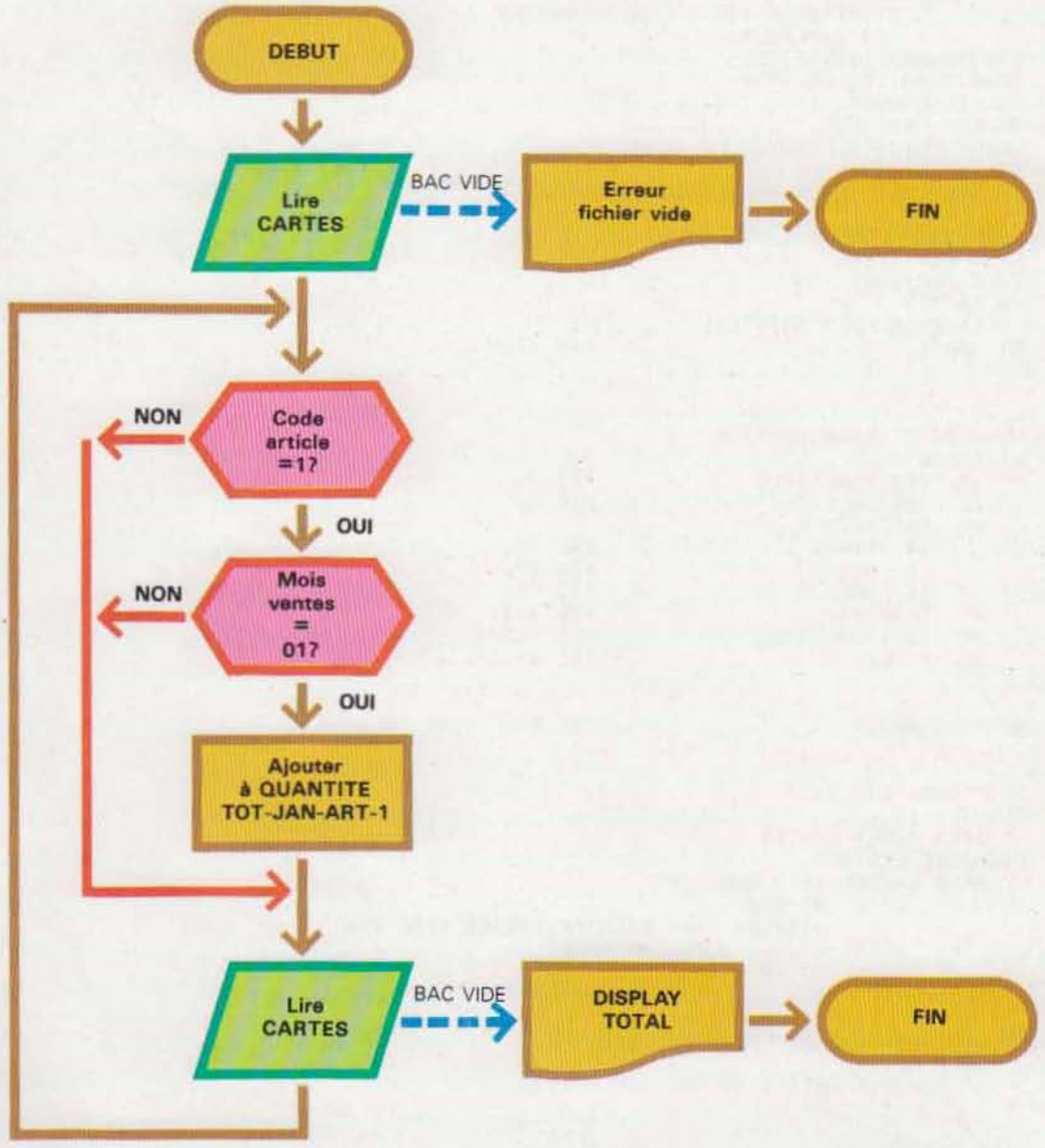
envoie le message ERREUR DE CALCUL à la console.

Gestion des tableaux en Cobol

En Cobol, un tableau est constitué d'un ensemble de champs contigus destinés à contenir des informations homogènes. Les tableaux constituent un des meilleurs moyens dont le Cobol dispose pour gérer des données en mémoire. En effet, si le recours à cette technique pour les petits volumes de données (à gérer directement en mémoire) peut être un choix du programmateur, il est pratiquement impossible de s'en passer lorsqu'on doit manipuler de nombreuses données.
Soit l'exemple d'un fichier de cartes à lire où chaque carte décrit la quantité des articles vendus dans un magasin.
A la fin de la lecture du fichier entier, le total des quantités vendues au mois de janvier du seul article de code 1 doit être imprimé. Un programme très simple suffirait à répondre au problème.
On peut observer comment, après avoir vérifié que la carte lue se réfère à l'article de code-1 et aux ventes de janvier, la somme de la quantité vendue est calculée dans l'unique totalisateur défini : TOT-JAN-ART-1. Ce programme est détaillé page 1088.
Pour s'appuyer sur une représentation visuelle qui nous sera utile par la suite, imaginons la mémoire réservée aux données comme une feuille où le totalisateur TOT-JAN-ART-2 occupe la partie indiquée dans l'encadré du haut de la page 1089.

Considérons maintenant le cas où l'on veut calculer la somme des quantités vendues, toujours de l'article de code 1, mais pour tous les autres mois de l'année.
Dans ce cas, l'organigramme du programme se complique considérablement, bien que le nombre d'informations à gérer soit encore limité (voir page 1089). Le programme Cobol se trouve aux pages 1090 et 1091.
Il n'est pas très pratique d'utiliser des totalisateurs séparés, comme nous venons de le faire, quand le nombre des totalisations devient supérieur à 3 ou 4 ; il devient donc quasiment impossible de la faire quand ce nombre avoisine la centaine ou le millier ce qui est courant en matière de gestion.

LECTURE ET TRAITEMENT D'UN FICHIER (1)



FLUX DES DONNEES

LECTURE ET TRAITEMENT D'UN FICHIER (1)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TABLEAUX.
 REMARKS. CE PROGRAMME EST UN EXEMPLE
          D'INTRODUCTION A L'UTILISATION
          DES TABLEAUX.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  ..........
 OBJECT-COMPUTER.  ..........
 INPUT-OUPUT SECTION.
 FILE-CONTROL.
     SELECT CARTES ASSIGN TO CARD-READER.
*
*
 DATA DIVISION.
 FILE SECTION.
 FD CARTES
    LABEL RECORD OMITTED.
 01 CARTE                          PIC X(80).
*
*
*
 WORKING-STORAGE SECTION.
 01 CARTE-WS.
    05  SERIE-ARTICLE              PIC 9.
    05  CODE-ARTICLE               PIC 9.
    05  DATE-VENTE.
        10  ANNEE                  PIC 99.
        10  MOIS                   PIC 99.
        10  JOUR                   PIC 99.
    05  QUANTITE                   PIC 9(3).
    05  COUT-UNITAIRE              PIC 9(3).
    05  FILLER                     PIC X(66).
*
*
 01 TOT-JAN-ART-1                  PIC 9(6) COMP VALUE 0.
*
*
 PROCEDURE DIVISION.
 DEBUT.
    OPEN INPUT CARTES.
 PREMIERE LECTURE.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY '*** FICHIER CARTES VIDE ***'
                        UPON PRINTER
                GO TO FIN-CALCUL.
 CONTROLE.
     IF  CODE-ARTICLE NOT = 1
         GO TO LIRE-CARTES.
     IF  MOIS = 1
         ADD QUANTITE TO TOT-JAN-ART-1.
*
*
 LIRE-CARTES.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY ' TOTAL VENTES ART.1 JANVIER = '
                        TOT-JAN-ART-1
                        UPON PRINTER
                GO TO FIN-CALCUL.
     GO TO CONTROLE.
*
*
 FIN-CALCUL.
     CLOSE CARTES.
     STOP RUN.
```

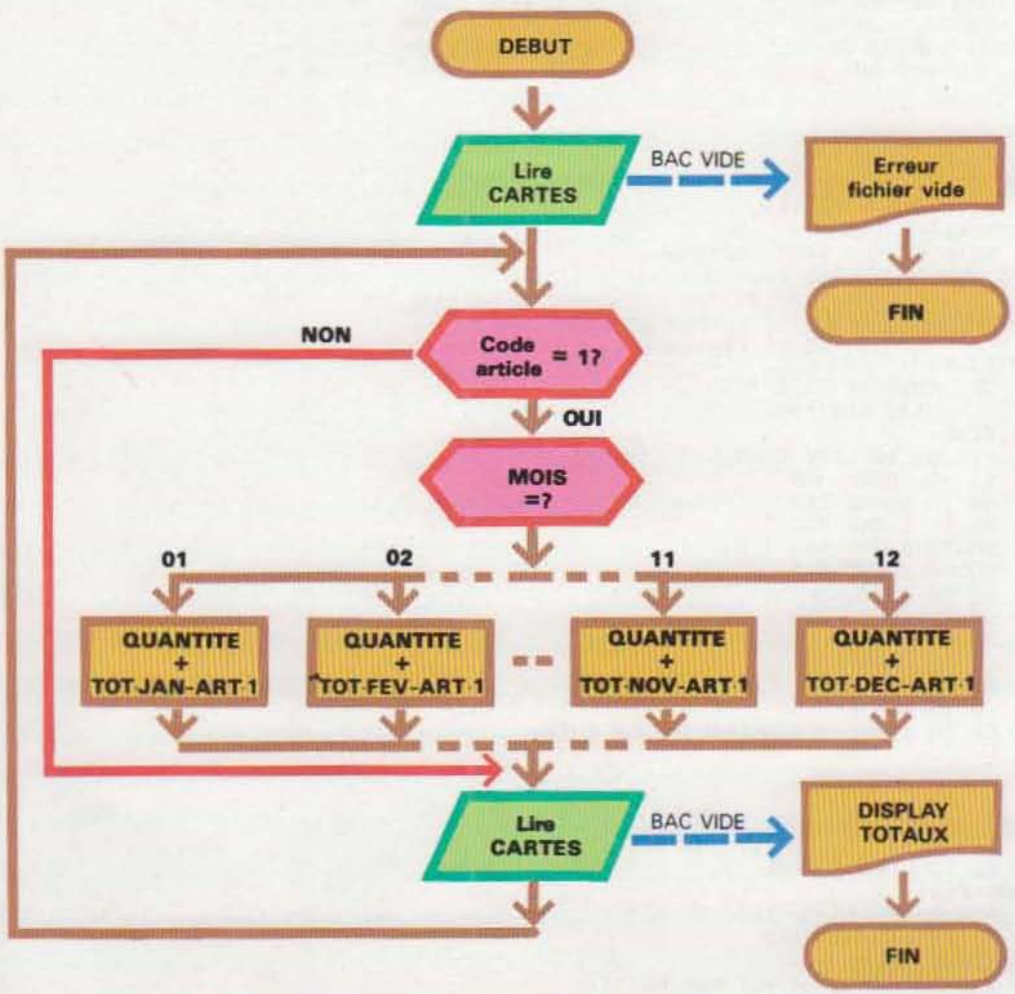
REPRESENTATION VISUELLE DE LA MEMOIRE AVEC UN TOTALISATEUR UNIQUE

Revenons à la mémoire des données disposée comme une feuille ; la partie occupée par les compteurs réservés aux quantités vendues d'articles du code-1 chaque mois est schématisée comme nous l'avons indiqué en haut de la page 1092.

Tableaux à une dimension

Les problèmes dont nous venons de parler ont une solution très simple. Les totalisateurs ont été déclarés au compilateur comme les 12 éléments d'un tableau. Chacun d'eux peut être adressé avec un nom générique d'élément et son positionnement est exprimé par un nombre dans le tableau.
Autrement dit, si l'on crée un tableau des

LECTURE ET TRAITEMENT D'UN FICHIER (2)



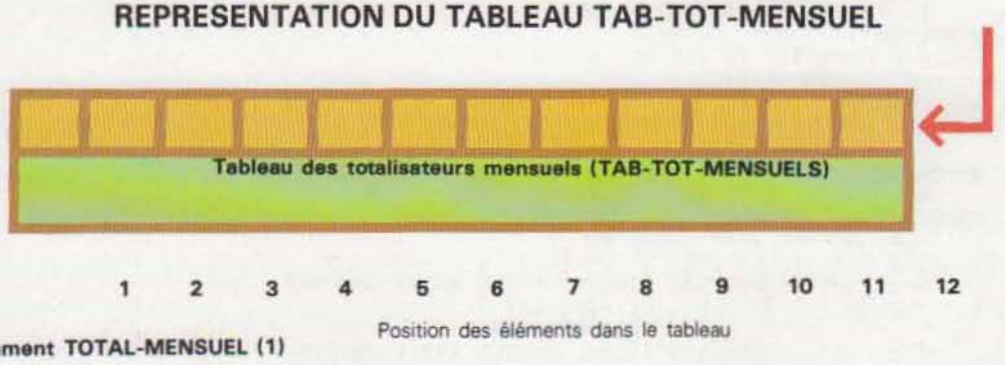
LECTURE ET TRAITEMENT D'UN FICHIER (2)

```
 WORKING-STORAGE SECTION.
 01  CARTE-WS.
     05  SERIE-ARTICLE              PIC 9.
     05  CODE-ARTICLE               PIC 9.
     05  DATE-VENTE.
         10  ANNEE                  PIC 9(2).
         10  MOIS                   PIC 9(2).
         10  JOUR                   PIC 9(2).
     05  QUANTITE                   PIC 9(3).
     05  COUT-UNITAIRE              PIC 9(3).
     05  FILLER                     PIC X(66).
*
*
*    ---------- T O T A L I S A T E U R S ----------
*
*
*
 01  TOT-JAN-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-FEV-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-MAR-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-AVR-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-MAI-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-JUI-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-JUL-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-AOU-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-SEP-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-OCT-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-NOV-ART-1                  PIC 9(6)  COMP  VALUE 0.
 01  TOT-DEC-ART-1                  PIC 9(6)  COMP  VALUE 0.
*
*
*
 PROCEDURE DIVISION.
 DEBUT.
     OPEN INPUT CARTES.
 PREMIERE-LECTURE.
     READ CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER
                 GO TO FIN-CALCUL.
 CONTROLE.
     IF  CODE-ARTICLE = 1
         NEXT SENTENCE
     ELSE
         GO TO LIRE-CARTES.
     GO TO SOMME-JAN
     GO TO SOMME-FEV
     GO TO SOMME-MAR
     GO TO SOMME-AVR
     GO TO SOMME-MAI
     GO TO SOMME-JUI
     GO TO SOMME-JUL
     GO TO SOMME-AOU
     GO TO SOMME-SEP
     GO TO SOMME-OCT
     GO TO SOMME-NOV
     GO TO SOMME-DEC DEPENDING ON MOIS.
*
*
*
 SOMME-JAN.
     ADD QUANTITE TO TOT-JAN-ART-1.
     GO TO LIRE-CARTES.
 SOMME-FEV.
     ADD QUANTITE TO TOT-FEV-ART-1.
     GO TO LIRE-CARTES.
 SOMME-MAR.
     ADD QUANTITE TO TOT-MAR-ART-1.
     GO TO LIRE-CARTES.
```

```
SOMME-AVR.

    ADD QUANTITE TO TOT-AVR-ART-1.
    GO TO LIRE-CARTES.
SOMME-MAI.
    ADD QUANTITE TO TOT-MAI-ART-1.
    GO TO LIRE-CARTES.
SOMME-JUI.
    ADD QUANTITE TO TOT-JUI-ART-1.
    GO TO LIRE-CARTES.
SOMME-JUL.
    ADD QUANTITE TO TOT-JUL-ART-1.
    GO TO LIRE-CARTES.
SOMME-AOU.
    ADD QUANTITE TO TOT-AOU-ART-1.
    GO TO LIRE-CARTES.
SOMME-SEP.
    ADD QUANTITE TO TOT-SEP-ART-1.
    GO TO LIRE-CARTES.
SOMME-OCT.
    ADD QUANTITE TO TOT-OCT-ART-1.
    GO TO LIRE-CARTES.
SOMME-NOV.
    ADD QUANTITE TO TOT-NOV-ART-1.
    GO TO LIRE-CARTES.
SOMME-DEC.
    ADD QUANTITE TO TOT-DEC-ART-1.
LIRE-CARTES.
    READ CARTES INTO CARTE-WS
                AT END
                DISPLAY 'TOTAL VENTES ART.1 JANVIER   = '
                        TOT-JAN-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 FEVRIER   = '
                        TOT-FEV-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 MARS      = '
                        TOT-MAR-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 AVRIL     = '
                        TOT-AVR-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 MAI       = '
                        TOT-MAI-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 JUIN      = '
                        TOT-JUI-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 JUILLET   = '
                        TOT-JUL-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 AOUT      = '
                        TOT-AOU-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 SEPTEMBRE = '
                        TOT-SEP-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 OCTOBRE   = '
                        TOT-OCT-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 NOVEMBRE  = '
                        TOT-NOV-ART-1
                        UPON PRINTER
                DISPLAY 'TOTAL VENTES ART.1 DECEMBRE  = '
                        TOT-DEC-ART-1
                        UPON PRINTER
                GO TO FIN-CALCUL.
    GO TO CONTROLE.
FIN-CALCUL.
    CLOSE CARTES.
    STOP RUN.
```

REPRESENTATION FIGUREE DE LA MEMOIRE AVEC 12 TOTALISATEURS SEPARES

REPRESENTATION DU TABLEAU TAB-TOT-MENSUEL



DESCRIPTION D'UN TABLEAU EN DATA-DIVISION

```
01 nom-tableau.
   05 nom-élément    PIC... USAGE... OCCURS N TIMES
```

totaux mensuels contenant 12 éléments, de nom générique TOTAL-MENSUEL, le compteur du mois d'avril sera adressé comme TOTAL-MENSUEL (4) c'est-à-dire comme le champ TOTAL-MENSUEL qui occupe la 4e position dans le tableau.

Le tableau est une donnée composée (ou structurée) dans laquelle les données composantes, (qui ne sont pas nécessairement élémentaires), ont toutes le même nom et peuvent être adressées seulement en fonction de leur position (voir 2e schéma de cette page).

Le compilateur doit évidemment savoir qu'une certaine donnée constitue un tableau. Cette information lui est fournie par le programmeur dans la DATA-DIVISION, lors de la description du champ, selon le format ci-dessus.

A l'aide de cette description, le compilateur réserve un espace « nom tableau » constitué par la série d'éléments « nom-élément ». L'occupation mémoire est identique à celle que l'on aurait obtenue si N champs avaient été définis séparément. Différence essentielle avec le tableau TAB-TOT-MENSUELS : au lieu des 12 compteurs TOT-JAN-ART-1... TOT-FEV-ART-1, le programme de l'exemple précédent est certainement moins fastidieux à détailler et plus aisé à lire grâce à la réduction draconienne du nombre d'instructions présentes (voir listing en pages 1093 et 1094).

L'exemple décrit dans ce programme contient certaines notions importantes qu'il est bon de souligner.

Pour éviter d'écrire, en fin de traitement,

LECTURE ET TRAITEMENT D'UN FICHIER (3)

```
 WORKING-STORAGE SECTION.
 01  CARTE-WS.
     05  SERIE-ARTICLE              PIC 9.
     05  CODE-ARTICLE               PIC 9.
     05  DATE-VENTE.
         10  ANNEE                  PIC 9(2).
         10  MOIS                   PIC 9(2).
         10  JOUR                   PIC 9(2).
     05  QUANTITE                   PIC 9(3).
     05  COUT-UNITAIRE              PIC 9(3).
     05  FILLER                     PIC X(66).
*
*
*    ------------  T A B L E A U X  -----------
*
**** TABLEAU TOTALISATEURS DES QUANTITES MENSUELLES ART.1 ****
*
*
 01  TAB-TOT-MENSUELS.
     05  TOTAL-MENSUEL              PIC 9(6)  COMP OCCURS 12.
**** TABLEAU DES NOMS DES MOIS ****
*
*
 01  MOIS.
     05  FILLER                     PIC X(9)  VALUE 'JANVIER   '.
     05  FILLER                     PIC X(9)  VALUE 'FEVRIER   '.
     05  FILLER                     PIC X(9)  VALUE 'MARS      '.
     05  FILLER                     PIC X(9)  VALUE 'AVRIL     '.
     05  FILLER                     PIC X(9)  VALUE 'MAI       '.
     05  FILLER                     PIC X(9)  VALUE 'JUIN      '.
     05  FILLER                     PIC X(9)  VALUE 'JUILLET   '.
     05  FILLER                     PIC X(9)  VALUE 'AOUT      '.
     05  FILLER                     PIC X(9)  VALUE 'SEPTEMBRE '.
     05  FILLER                     PIC X(9)  VALUE 'OCTOBRE   '.
     05  FILLER                     PIC X(9)  VALUE 'NOVEMBRE  '.
     05  FILLER                     PIC X(9)  VALUE 'DECEMBRE  '.
 01  TABLEAU-MOIS  REDEFINES   MOIS.
     05  NOM-MOIS                   PIC X(9)  OCCURS 12.
*
 01  INDICE-MOIS                    PIC 9(2)  COMP.
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 DEBUT.
     MOVE LOW-VALUES TO TAB-TOT-MENSUELS.
     OPEN INPUT CARTES.
 PREMIERE-LECTURE.
     READ CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER


                 GO TO FIN-CALCUL.
 CONTROLE.
     IF  CODE-ARTICLE = 1
         ADD QUANTITE TO TOTAL-MENSUEL (MOIS).
     READ CARTES INTO CARTE-WS
                 AT END
                 PERFORM AFFICHAGE-TOTAUX
                 GO TO FIN-CALCUL.
     GO TO CONTROLE.
 FIN-CALCUL.
     CLOSE CARTES.
     STOP RUN.
```

```
*
*============  S E C T I O N S  ============
*
*
*
 AFFICHAGE-TOTAUX SECTION.
 AFFI-TOT.
     MOVE ZEROES TO INDICE-MOIS.
 BOUCLE-AFFICHAGE.
     ADD 1 TO INDICE-MOIS.
     IF  INDICE-MOIS > 12
         GO TO BOUCLE-AFFICHAGE-EX.
     DISPLAY ' TOTAL VENTES ART.1 '
             NOM-MOIS (INDICE-MOIS)
             ' = '
             TOTAL-MENSUEL (INDICE-MOIS)
             UPON PRINTER.
     GO TO BOUCLE-AFFICHAGE.
 BOUCLE-AFFICHAGE-EX. EXIT.
*
```

12 instructions DISPLAY contenant les messages :

TOTAL-VENTES ART. 1 mois = total-mois

on fait appel à un autre tableau dans lequel chaque élément contient, en clair, le nom d'un des douze mois de l'année. De cette façon, le message à imprimer devient paramétré et il peut être répété 12 fois en reportant, à chaque fois, le nom du mois et le nombre des ventes correspondantes.
La description d'un tableau ne peut contenir la clause VALUE. Quand on a besoin de mémoriser des valeurs constantes et prédéfinies dans les éléments d'un tableau, on recourt à la technique utilisée dans l'exemple, à savoir :

1 / On commence par définir le champ composé d'autant de sous-champs qu'il y a d'éléments dans le tableau (dans ce cas, 12). Il n'est pas nécessaire d'utiliser des noms définis par le programmeur pour nommer les sous-champs ; le nom réservé FILLER est suffisant. Les sous-champs utilisés, tous de même longueur, commencent à la valeur qui nous intéresse. Dans notre exemple, la longueur d'un champ est égale à celle du nom le plus long (SEPTEMBRE).

2 / Avec la clause REDEFINES du champ précédent, on décrit un champ-tableau et on précise le nombre et le nom des éléments dans le tableau (clause OCCURS).

Avec la technique des tableaux, on peut ainsi adresser un élément générique. Celui-ci, grâce à la clause REDEFINES, partage la partie mémoire du champ redéfini avec la valeur imposée. Par rapport au champ MOIS et au TABLEAU-MOIS qui le définit, la partition de la mémoire fournit le schéma ci-contre. La section AFFICHAGE-TOTAUX, qui intervient à la fin de la lecture du fichier CARTES par l'intermédiaire du verbe PERFORM, utilise le champ INDICE-MOIS pour se positionner simultanément sur les éléments des tableaux TABLEAU-MOIS et TAB-TOT-MENSUELS.
La valeur d'INDICE-MOIS est incrémentée de 1 jusqu'à 12. Pour chaque valeur prise par l'indice, le nom du mois correspondant (par exemple 1 = JANVIER) et le contenu de la quantité totalisée dans ce mois sont prélevés de façon à composer le message désiré (voir ci-contre le diagramme de construction des 12 messages).
Une attention particulière doit être accordée aux cycles de lecture ou d'écriture des données dans le tableau.
Le compilateur n'est pas capable de se rendre

compte si un indice assume une valeur supérieure ou inférieure à celles accordées aux dimensions du tableau. Si, par exemple, en lecture de TABLEAU-MOIS, l'indice est imposé de façon erronée à la valeur 13 et si la situation en mémoire est celle décrite en haut de la page 1096, le message composé s'écrit :

TOTAL VENTES ART. 1 ? <A47Y/&=S78)(5

Ici, l'erreur saute aux yeux, mais ce n'est pas toujours le cas : parfois, la gestion erronée d'un indice peut entraîner une anomalie totale des résultats ou la fin anormale du programme.

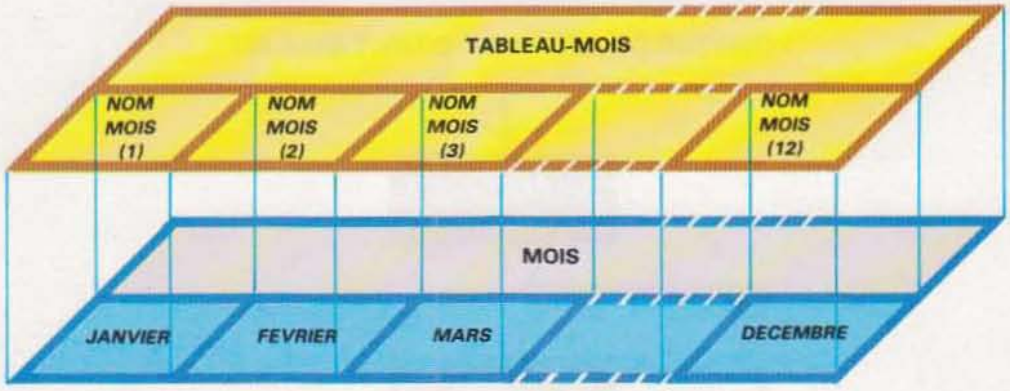
Nous reviendrons plus longuement sur ce sujet lorsque nous parlerons du chargement d'un tableau. Pour le moment, le lecteur doit observer la section AFFICHAGE-TOTAUX de l'exemple. Son organigramme (page 1096) est très simple. Quand il est adopté pour la lecture de toutes les données d'un tableau, il permet d'éviter les erreurs précédemment décrites.

Tableaux à deux dimensions

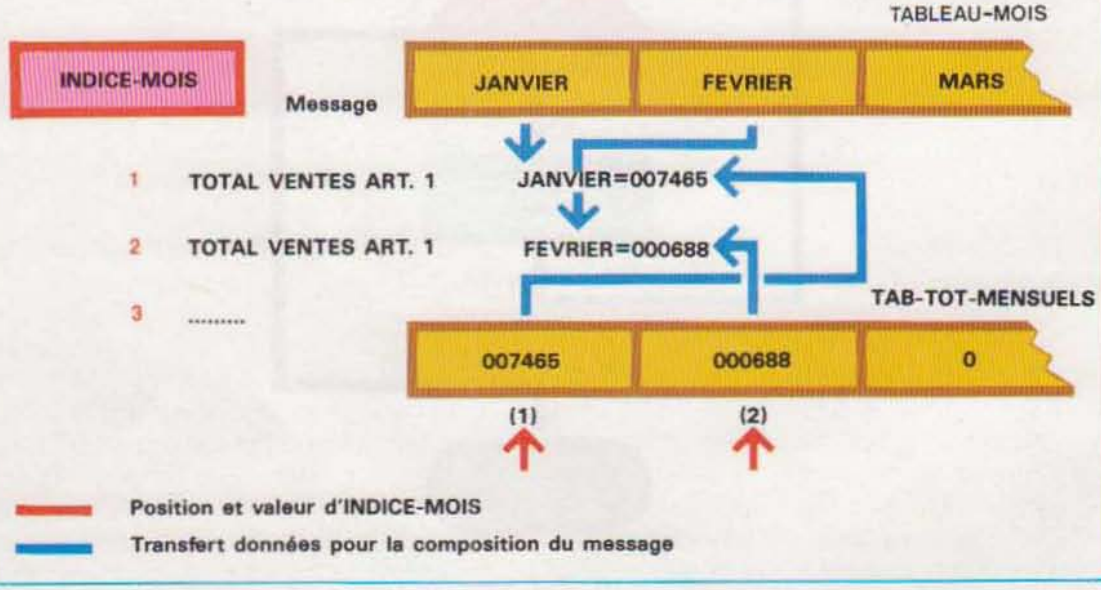
Considérons à nouveau l'exemple exposé au début.

On remarque qu'un programme capable de traiter le total des ventes mensuelles d'un

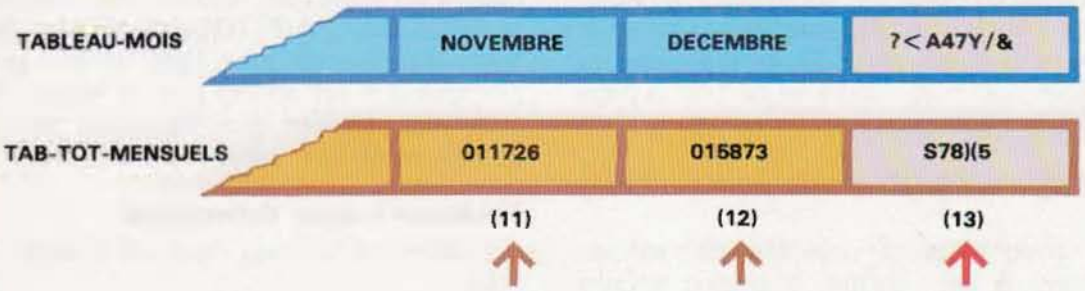
ATTRIBUTION DE VALEURS CONSTANTES A UN TABLEAU



COMPOSITION DU CHAMP D'IMPRESSION



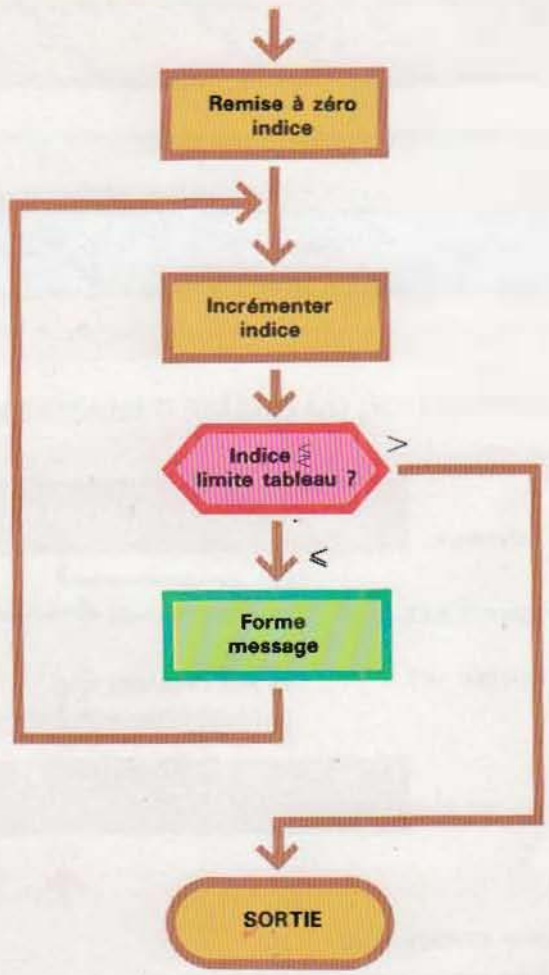
TAILLE D'UN TABLEAU AVEC DES VALEURS D'INDICE NON PERMISES



Champs adjacents aux tableaux mais n'appartenant pas aux tableaux mêmes

Position du pointeur avec INDICE-MOIS=13

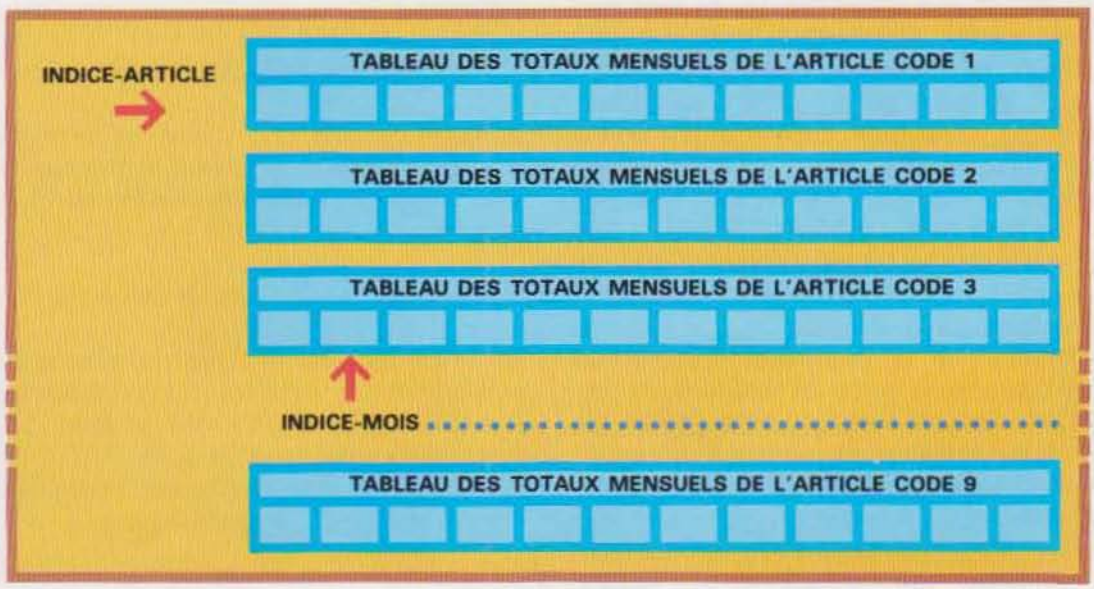
LOGIQUE DE LECTURE D'UN TABLEAU



article unique n'a que peu d'intérêt. Pour simplifier, convenons que le code article est représenté par un seul caractère numérique et extrapolons le type de traitement décrit plus haut (article de code 1) à la gestion de tous les articles codés de 1 à 9. Poursuivant cette généralisation, on peut également gérer la partie de mémoire dédiée aux totalisateurs des quantités mensuelles (voir schéma ci-dessous).
La « feuille mémoire » tout entière est alors assimilée à un tableau dans lequel chaque élément se comporte comme un tableau. Pour lire ou pour modifier le totalisateur du mois de janvier de l'article de code 2, il faudra d'une part se positionner sur la ligne numéro 2 (INDICE-ARTICLE) et d'autre part sur la première position (JANVIER=1) du tableau des totalisateurs mensuels (INDICE-MOIS).
La situation se rapproche beaucoup du jeu de la « bataille navale » dans lequel on communique à l'adversaire sa propre manœuvre en fournissant les codes de la ligne et de la colonne pour « cibler » le bateau ennemi.
Avant de fournir l'organigramme de la nouvelle version du programme, arrêtons-nous un instant sur la manière d'écrire le « tableau des tableaux ». Puisque on a prévu de ne traiter que 9 articles, chacun d'entre eux devra correspondre à un élément du tableau, lui-même ensuite défini comme un tableau de 12 compteurs dans lequel est totalisée la quantité mensuelle vendue par article.
Finalement, l'occupation mémoire sera de 9 lignes × 12 colonnes × 4 octets = 432 octets. Rappelons-nous, en effet, qu'un champ numérique de longueur déclarée entre 4 et 9 caractères occupe en réalité 4 octets en USAGE-COMP. Dans notre cas, chaque compteur est décrit PIC 9(6) COMP, occupant par conséquent 4 octets. Le tableau des totalisateurs des articles TAB-TOT-ART sera décrit :

```
01 TAB-TOT-ART.
   05 TOTAUX-ARTICLE          OCCURS 9.
      10 TAB-TOT-MENSUELS.
         15 TOTAL-MENSUEL PIC 9(6)
                         COMP
                         OCCURS 12.
```

SCHEMA D'UN TABLEAU A DEUX DIMENSIONS TAB-TOT-ART.



Finsiel

Le stylo optique relié à une caisse enregistreuse permet de lire les informations sur l'emballage.

Le programme détaillé se trouve en 1099 et 1100. Bien que, dans cette nouvelle version, le traitement concerne 9 × 12 = 108 totalisateurs et non plus seulement 12, on peut voir que le nombre d'instructions dans la MAIN SECTION n'a pratiquement pas changé.
La seconde remarque concerne la possibilité, offerte par les tableaux, d'initialiser chacun de leurs éléments avec une seule instruction quand ils sont constitués de champs homogènes. La ligne MOVE LOW-VALUES TO TAB-TOT-ART, qui renvoie au plus haut niveau du tableau (01), initialise à LOW-VALUES tous les éléments de niveau inférieur.
La fonction de la procédure AFFICHAGE-TOTAUX, activée en fin de lecture du fichier CARTES, consiste à lire le contenu entier du tableau des totalisateurs, mettant ainsi en évidence les totaux des ventes de chaque mois en face de chaque article. On se reportera au diagramme de lecture page 1101 qui met bien en évidence les instructions de lecture « horizontale » (celle des compteurs mensuels se rapportant au même article) et les instructions de pointage « vertical », (le positionnement sur la ligne). On notera, par exemple, que le positionnement sur le totalisateur de mai pour l'article de code 9 est effectué par l'élément :

TOTAL-MENSUEL (9, 5)

La façon correcte, pour fournir les références d'un élément dans un tableau de tableaux (ou mieux, dans un tableau à deux dimensions), est la suivante :

nom-élément (indice-ligne, ␣ indice-colonne)

Un blanc (espace) doit nécessairement être présent entre la virgule et l'indice de colonne. Dans le listing de la page 1102, seuls les résultats relatifs aux articles de codes 1, 2, 3 et 4 ont été portés, pour des raisons évidentes de compréhension du programme.

Tableaux à trois dimensions

De notre exposé sur les tableaux, il ressort qu'un seul indice suffit dans le TABLEAU-MOIS pour adresser complètement l'élément

UTILISATION D'UN TABLEAU A DEUX DIMENSIONS

```
 WORKING-STORAGE SECTION.
 01  CARTE-WS.
     05  SERIE-ARTICLE            PIC 9.
     05  CODE-ARTICLE             PIC 9.
     05  DATE-VENTE.
         10  ANNEE                PIC 9(2).
         10  MOIS                 PIC 9(2).
         10  JOUR                 PIC 9(2).
     05  QUANTITE                 PIC 9(3).
     05  COUT-UNITAIRE            PIC 9(3).
     05  FILLER                   PIC X(66).
*
*
*    ------------  T A B L E A U X  -----------
*
**** TABLEAU TOTALISATEURS DES QUANTITES MENSUELLES ART.1 ****
*
*
 01  TAB-TOT-ART.
     05  TOTAUX-ARTICLES          OCCURS 9.
         10  TAB-TOT-MENSUELS.
             15  TOTAL-MENSUEL    PIC 9(6) COMP OCCURS 12.
*
*
**** TABLEAU DES NOMS DES MOIS ****
*
*
 01  MOIS.
     05  FILLER                    PIC X(9)  VALUE 'JANVIER   '.
     05  FILLER                    PIC X(9)  VALUE 'FEVRIER   '.
     05  FILLER                    PIC X(9)  VALUE 'MARS      '.
     05  FILLER                    PIC X(9)  VALUE 'AVRIL     '.
     05  FILLER                    PIC X(9)  VALUE 'MAI       '.
     05  FILLER                    PIC X(9)  VALUE 'JUIN      '.
     05  FILLER                    PIC X(9)  VALUE 'JUILLET   '.
     05  FILLER                    PIC X(9)  VALUE 'AOUT      '.
     05  FILLER                    PIC X(9)  VALUE 'SEPTEMBRE '.
     05  FILLER                    PIC X(9)  VALUE 'OCTOBRE   '.
     05  FILLER                    PIC X(9)  VALUE 'NOVEMBRE  '.
     05  FILLER                    PIC X(9)  VALUE 'DECEMBRE  '.
 01  TABLEAU-MOIS  REDEFINES   MOIS.
     05  NOM-MOIS                  PIC X(9)  OCCURS 12.
*
*
 01  INDICE-ARTICLE                PIC 9  COMP.
 01  INDICE-MOIS                   PIC 9(2)  COMP.
*
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 DEBUT.
     MOVE LOW-VALUES TO TAB-TOT-ART.
     OPEN INPUT CARTES.
 PREMIERE-LECTURE.
     READ CARTES INTO CARTE-WS
                 AT END
                 DISPLAY '** FICHIER CARTES VIDE **'
                         UPON PRINTER
                 GO TO FIN-CALCUL.
 SOMME.
     ADD QUANTITE TO TOTAL-MENSUEL (CODE-ARTICLE, MOIS).
     READ CARTES INTO CARTE-WS
                 AT END
                 PERFORM AFFICHAGE-TOTAUX
                 GO TO FIN-CALCUL.
     GO TO SOMME.
 FIN-CALCUL.
     CLOSE CARTES.
     STOP RUN.
*
```

```
* ============  S E C T I O N S  ============

*
*
 AFFICHAGE-TOTAUX SECTION.
 AFFI-TOT.
     MOVE ZEROES TO INDICE-ARTICLE.
 AFFICHAGE-ARTICLE.
     ADD 1 TO INDICE-ARTICLE.
     IF  INDICE-ARTICLE > 9
         GO TO AFFICHAGE-ARTICLE-EX.
     DISPLAY SPACES UPON PRINTER.
     DISPLAY ' ARTICLE COD. '
             INDICE-ARTICLE
             '     '
             '* * * TOTAUX VENTES MENSUELLES * * *'
             UPON PRINTER.
     DISPLAY SPACES UPON PRINTER.
     MOVE ZEROES TO INDICE-MOIS.
 AFFICHAGE-VENTES.
     ADD 1 TO INDICE-MOIS.
     IF  INDICE-MOIS > 12
         GO TO AFFICHAGE-ARTICLE.
     DISPLAY '
             NOM-MOIS (INDICE-MOIS)
             ' = '
             TOTAL-MENSUEL (INDICE-ARTICLE, INDICE-MOIS)
             UPON PRINTER.
     GO TO AFFICHAGE-VENTES.
 AFFICHAGE-ARTICLE-EX. EXIT.
```

contenant la description du mois. Inversement, pour adresser complètement le totalisateur de la quantité vendue de l'article de code 2 au mois de mai, il faut deux indices. Ils permettent le double positionnement, d'abord sur la ligne correspondante à l'article (2) et, ensuite, au milieu de celle-ci, sur la colonne se rapportant au mois de mai (5).

Dans le premier cas (TABLEAU-MOIS), nous avons parlé d'un tableau à une dimension alors que dans l'exemple précédent, il s'agissait d'un tableau à deux dimensions.

Bien noter que le Cobol autorise une définition de tableaux avec un maximum de trois dimensions.

Poursuivons avec le même exemple : nous continuons à totaliser les quantités vendues mensuellement pour chaque article et pouvant appartenir à 9 série différentes.

Pour la clarté de la démonstration, supposons qu'une carte contient les valeurs :

SERIE = 2
ARTICLE = 5
MOIS = 12
QUANTITE = 007215

et une autre, ces valeurs :

SERIE = 4
ARTICLE = 5
MOIS = 12
QUANTITE = 000036

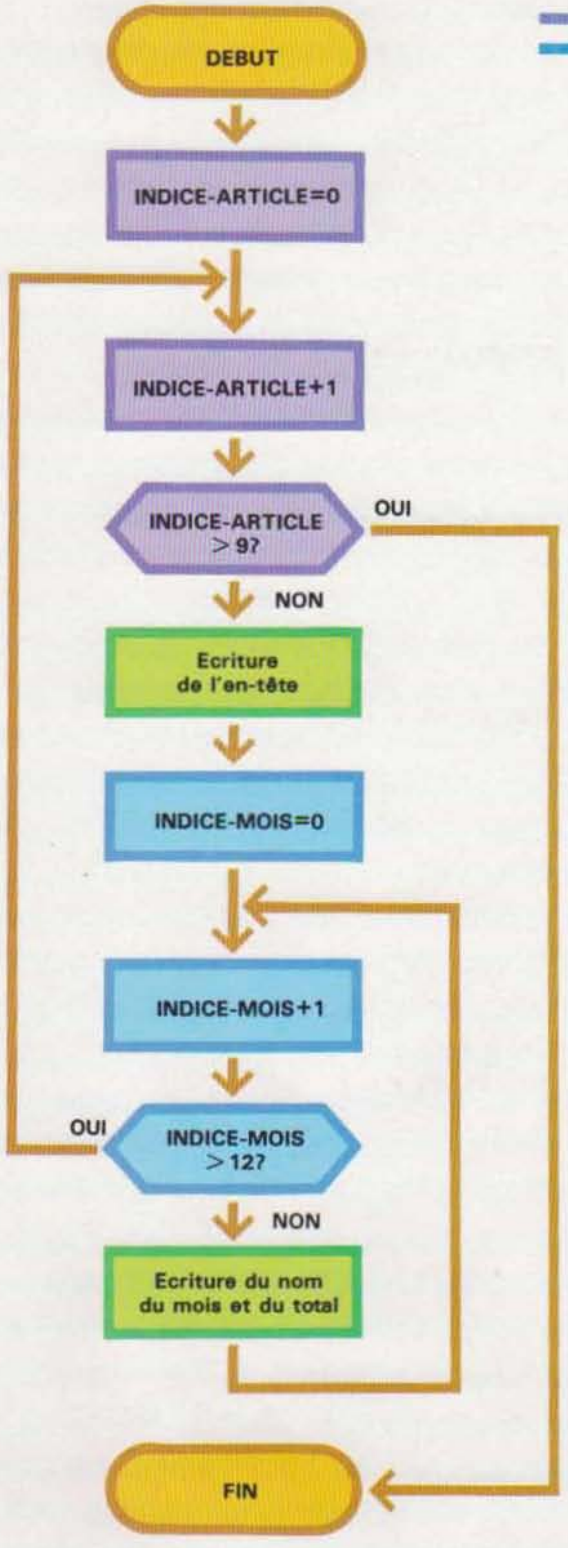
Malgré l'identité des valeurs respectives d'ARTICLES et de MOIS, il est évident que les quantités doivent être totalisées dans deux compteurs distincts.

Nous retrouvons ce découpage de la mémoire réservée aux totalisateurs mensuels page 1103. La mémoire est représentée par une série de « feuilles » adressées par le nombre de SERIE. Chaque feuille est ensuite traitée et adressée comme dans l'exemple précédent : une ligne correspond à un article et un champ de la ligne au contenu (totalisateur) des ventes mensuelles.

En WORKING-STORAGE SECTION, le tableau s'écrit :

```
01 TAB-TOT-VENTES
   05 TOTAUX-SERIE              OCCURS 9.
      10 TOTAUX-ARTICLES        OCCURS 9.
         15 TOTAL-MENSUEL PIC 9 (6)
                         COMP
                         OCCURS 12.
```

ORGANIGRAMME DE LA PROCEDURE DISPLAY-TOTAUX



Instructions pour lecture "verticale"
Instructions pour lecture "horizontale"

Remise à zéro de l'indice de ligne INDICE-ARTICLE

Incrémentation de l'indice de ligne

Contrôle de la position maximale "verticale" atteinte

En-tête
ARTICLE COD. N. *** TOTAUX VENTES MENSUELLES ** où N est le code générique en cours de traitement

Remise à zéro de l'indice de colonne INDICE-MOIS pour la lecture des compteurs mensuels

Incrémentation de l'indice de colonne

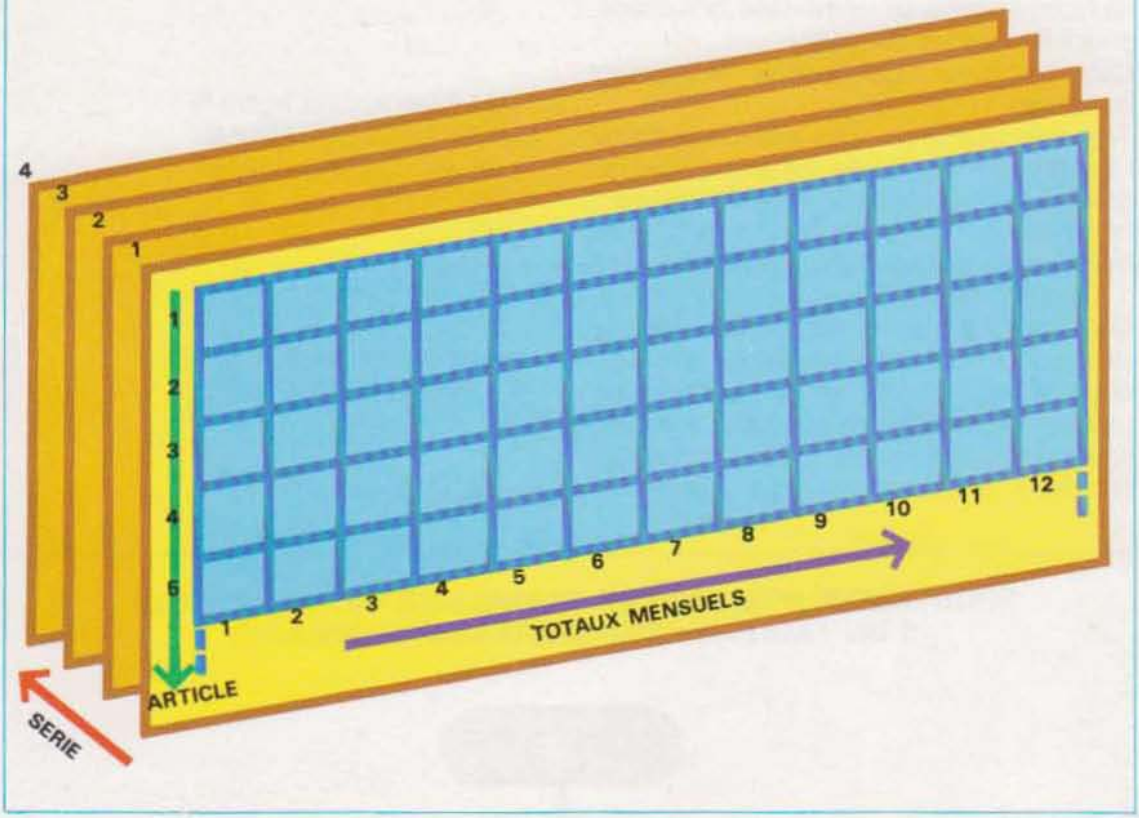
Contrôle de la position maximale "horizontale" atteinte

Le nom du mois et le total des ventes sont affichés pour l'article traité

EXEMPLE D'IMPRESSION DES DONNEES D'UN TABLEAU A DEUX DIMENSIONS

```
ARTICLE COD.1   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006001
                MAI         = 003564
                JUIN        = 001234
                JUILLET     = 001234
                AOUT        = 006574
                SEPTEMBRE   = 012564
                OCTOBRE     = 010084
                NOVEMBRE    = 011284
                DECEMBRE    = 010064

ARTICLE COD.2   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006002
                MAI         = 003564
                JUIN        = 002234
                JUILLET     = 002234
                AOUT        = 006574
                SEPTEMBRE   = 022564
                OCTOBRE     = 020084
                NOVEMBRE    = 021284
                DECEMBRE    = 020064

ARTICLE COD.3   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006001
                MAI         = 003564
                JUIN        = 001334
                JUILLET     = 001334
                AOUT        = 006574
                SEPTEMBRE   = 013564
                OCTOBRE     = 010084
                NOVEMBRE    = 011384
                DECEMBRE    = 010064

ARTICLE COD.4   * * * TOTAUX VENTES MENSUELLES * * *

                JANVIER     = 008974
                FEVRIER     = 004567
                MARS        = 006787
                AVRIL       = 006004
                MAI         = 003564
                JUIN        = 004334
                JUILLET     = 041334
                AOUT        = 006574
                SEPTEMBRE   = 012564
                OCTOBRE     = 010084
                NOVEMBRE    = 011284
                DECEMBRE    = 010064
```

SCHEMA D'UN TABLEAU A TROIS DIMENSIONS
TAB-TOT-VENTES



Chargement d'un tableau

Dans le listing de la page 1102, on remarque que le mot ARTICLE COD. 1 prévoit, côté utilisateur, la consultation ultérieure d'un indice qui mette bien en évidence la description de l'article de code 1.
En réalité, la consultation de l'indice (qui a pour but de vérifier la correspondance entre le code et la description) doit être effectuée par le même programme, ce qui nécessite la production de l'en-tête suivant :

ARTICLE : description-article

La correspondance peut être établie en utilisant un tableau dans lequel chaque élément contient :

1 / le nombre de série (1 caractère)
2 / le code article (1 caractère)
3 / la description de l'article (30 caractères).

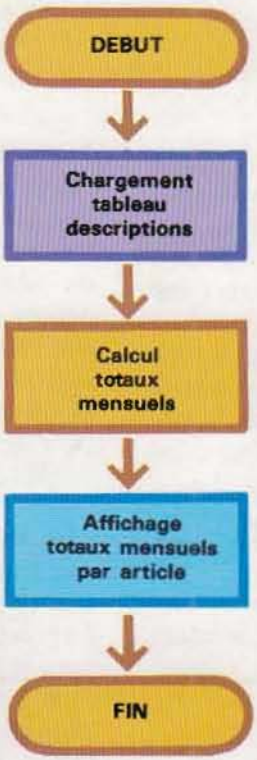
Puisqu'il est logique de penser que de nouveaux articles seront insérés et que certains des paramètres déjà catalogués devront être modifiés, il est préférable que le chargement du tableau de description soit effectué en-dehors de chaque exécution du programme, par l'intermédiaire d'un fichier de cartes, par exemple.
Si l'on adoptait la technique déjà employée pour le tableau des noms de mois, chaque modification du contenu du tableau serait suivie de la correction de la description correspondante dans WORKING-STORAGE-SECTION et de la recompilation du programme.
La procédure est analogue à celle déjà utilisée pour le calcul des totaux des ventes.
Avec une différence, toutefois : alors que, dans ce dernier cas, on pouvait adresser directement le compteur d'article (grâce aux codes de série, article et mois), entreprendre le char-

gement d'un tableau impose un autre type de gestion de l'indice.

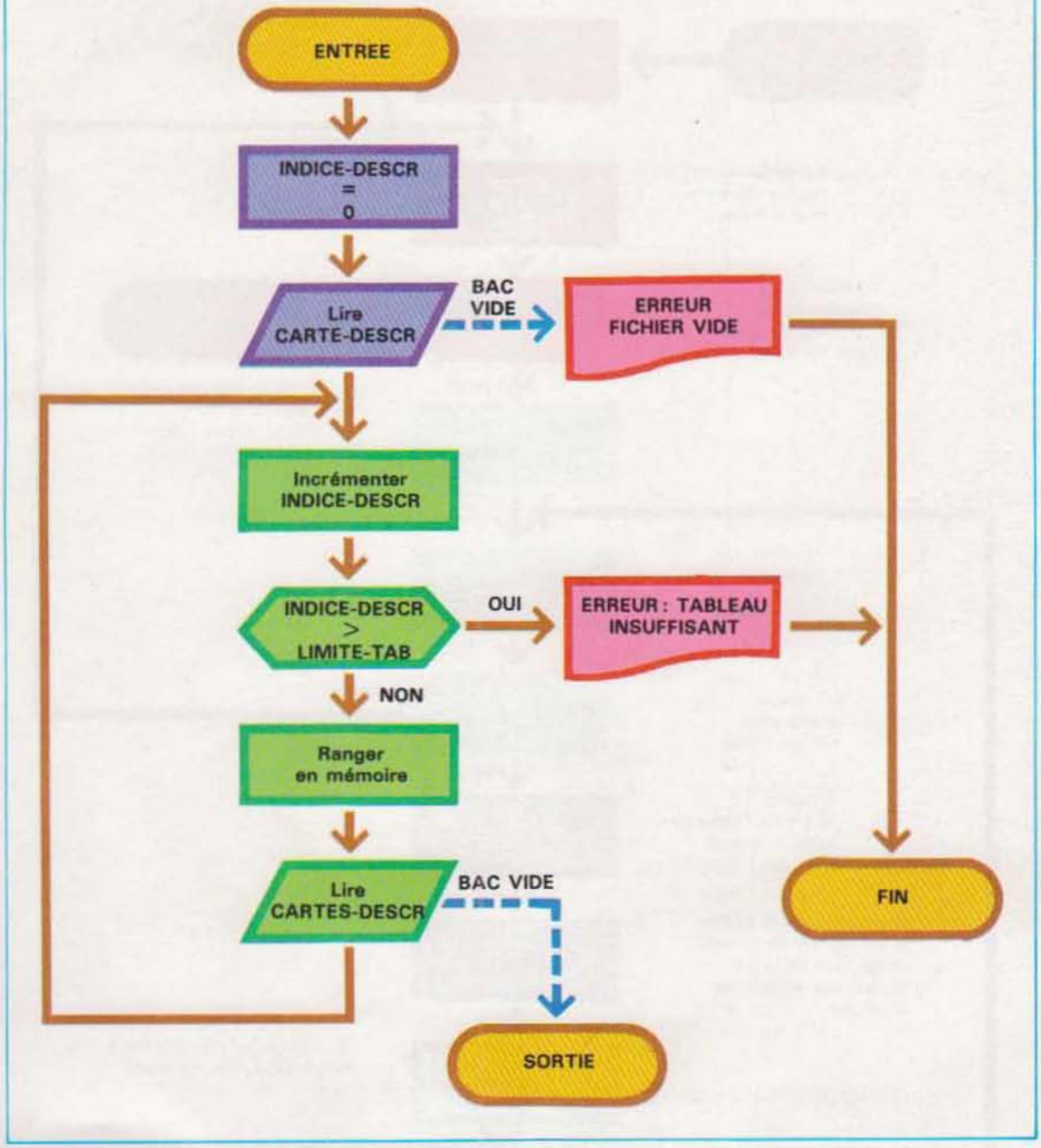
L'indice ou les indices du tableau à charger doivent être gérés et contrôlés à chaque pas, de façon à obtenir un remplissage sans discontinuité de l'espace disponible.
Ceci afin d'éviter notamment un débordement du tableau.
Tout cela paraîtra plus clair après avoir étudié l'exemple suivant. Dans l'organigramme ci-dessous, nous avons représenté le chargement du tableau de description des articles. Chaque bloc peut être, à son tour, détaillé (voir page 1105, 1106 et 1107).
On suppose que les cartes décrivant les articles sont disposées à l'intérieur du fichier par ordre croissant de série et d'article. Dans l'hypothèse où les descriptions des 9 articles (vêtements par exemple) et des 9 series sont présentes, le fichier des cartes-description peut être organisé comme suit :

11 CHEMISES HOMME
12 CHEMISES FEMME
13 CHEMISES ENFANT
14 .....
15 .....
..
..
21 PANTALONS HOMME
22 PANTALONS FEMME
23 PANTALONS ENFANT
24 .....
25 .....
..
..
..
91 CHAUSSURES HOMME
92 CHAUSSURES FEMME
..
..
99 SANDALES ENFANT

PROGRAMME UTILISANT LE CHARGEMENT ET LA GESTION D'UN TABLEAU (ORGANIGRAMME SOMMAIRE)



PHASE DE CHARGEMENT DU TABLEAU DES DESCRIPTIONS

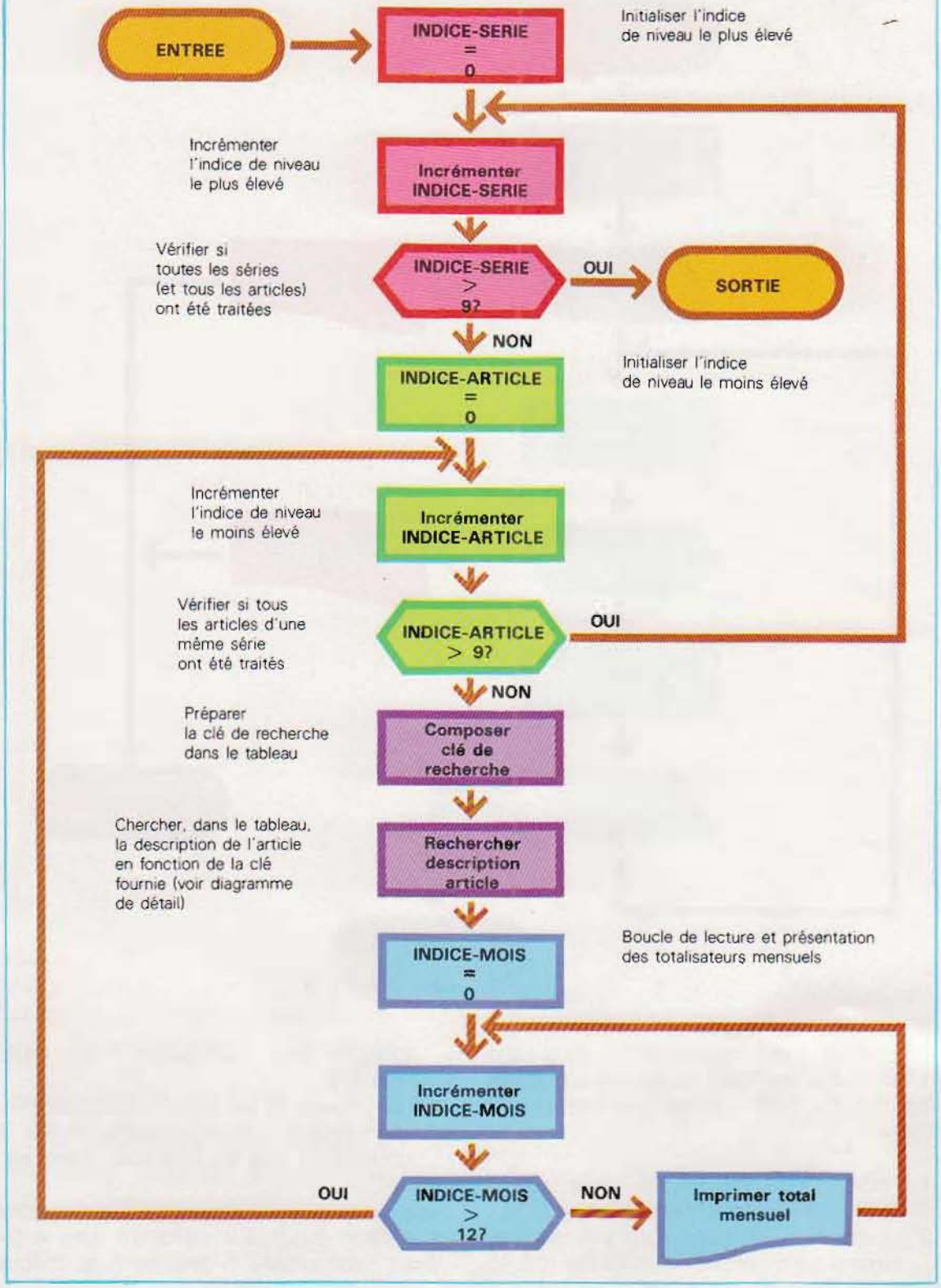


On constate, dans l'organigramme de la page 1106, que la technique de lecture du tableau est encore celle utilisée précédement, à savoir :

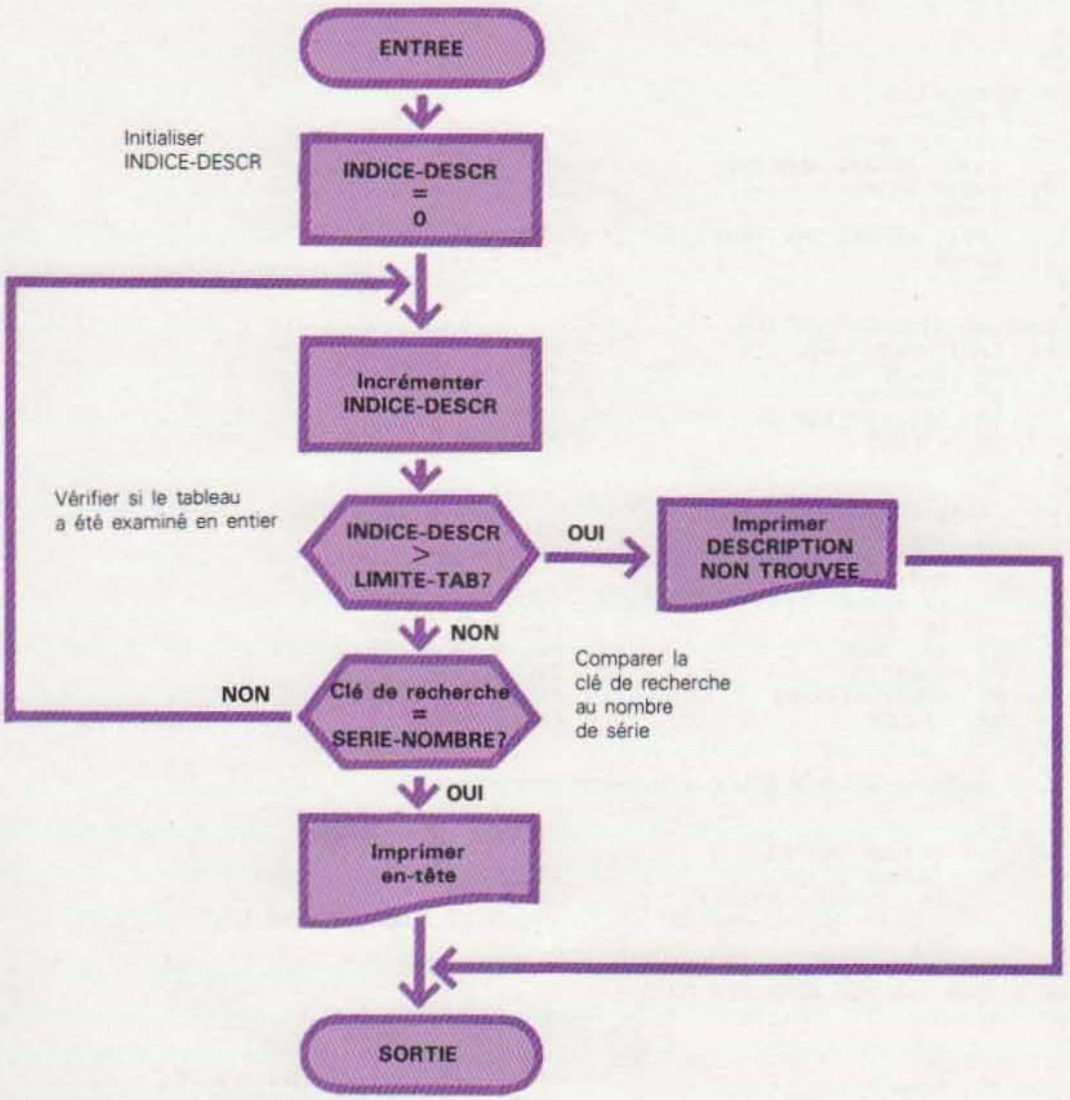
1 / l'indice de niveau le plus élevé (INDICE-SERIE) est incrémenté et contrôlé ;
2 / la valeur de cet indice n'est pas modifiée avant le traitement de la totalité des articles (contrôle pour INDICE-CONTROLE supérieur à 9) ;
3 / Une boucle de test est effectuée pour chaque valeur de l'indice pour lire et afficher le contenu de tous les totaliseurs mensuels.

Une importance particulière doit être accordée à la partie dédiée à la recherche dans le tableau, pour extraire la description de chaque

PHASE DE LECTURE DES CARTES-DESCRIPTION



PHASE DE RECHERCHE DE LA DESCRIPTION D'UN ARTICLE



article (page 1107). Puisqu'il s'agit de mettre en évidence la description de tous les aspects d'une gestion commerciale, un champ utilitaire (clé de recherche), indiquant les valeurs du nombre de série et du code des articles est composé pour chacun d'eux.

Le champ-clé est donc itérativement comparé aux deux premiers caractères de chaque élément du tableau des descriptions.

La boucle s'interrompt lorsque les deux champs comparés s'avèrent égaux : dans ce cas, on prélève la partie DESCRIPTION de l'élément pour composer la ligne du titre.

Il est alors prudent d'envisager le cas où l'égalité décrite n'est pas vérifiée ; dans cette éventualité, il est alors prévu de remplacer la description de l'article par ses codes indentificateurs (série et nombre) accompagnés du commentaire DESCRIPTION NON TROUVEE. On verra le programme détaillé pages 1108 à 1110. Les descriptions de certaines parties, pour lesquelles nous renvoyons le lecteur aux exemples précédents, ont été volontairement omises.

PROGRAMME POUR LE CHARGEMENT ET LA GESTION D'UN TABLEAU

```
*
*
*
 DATE DIVISION.
 FILE SECTION.
 FD  CARTES-DESCR
     LABEL RECORD OMITTED.
 01  CARTE-DESCR                          PIC X(80).
 FD  CARTES
     LABEL RECORD OMITTED.
 01  CARTE                                PIC X(80).
*
*
 WORKING-STORAGE SECTION.
 01  CARTE-DESCR-WS.
     05  SERIE-SK                         PIC 9.
     05  CODE-SK                          PIC 9.
     05  DESCRIPTION-SK                   PIC X(30).
     05  FILLER                           PIC X(48).
*
*
 01  CARTE-WS.
     05  SERIE-ARTICLE                    PIC 9.
     05  CODE-ARTICLE                     PIC 9.
     05  DATE-VENTE.
         10 ANNEE                         PIC 9(2)  COMP.
         10  MOIS                         PIC 9(2)  COMP.
         10  JOUR                         PIC 9(2)  COMP.
     05  QUANTITE                         PIC 9(3).
     05  COUT-UNITAIRE                    PIC 9(3).
     05  FILLER                           PIC X(66).
*
*
* ------------  T A B L E A U  ------------
*
*
 01  TAB-TOTAUX-VENTES.
     05  TOTAUX-SERIE                     OCCURS 9.
         10  TOTAUX-ARTICLE               OCCURS 9.
             15  TOTAL-MENSUEL            PIC 9(6)  COMP OCCURS 12.
*
*
****  TABLEAU DES NOMS DES MOIS ****
*
*
 01  MOIS .
     05  FILLER                           PIC X(9)  VALUE 'JANVIER  '.
     05  FILLER                           PIC X(9)  VALUE 'FEVRIER  '.
     05  FILLER                           PIC X(9)  VALUE 'MARS     '.
     05  FILLER                           PIC X(9)  VALUE 'AVRIL    '.
     05  FILLER                           PIC X(9)  VALUE 'MAI      '.
     05  FILLER                           PIC X(9)  VALUE 'JUIN     '.
     05  FILLER                           PIC X(9)  VALUE 'JUILLET  '.
     05  FILLER                           PIC X(9)  VALUE 'AOUT     '.
     05  FILLER                           PIC X(9)  VALUE 'SEPTEMBRE'.
     05  FILLER                           PIC X(9)  VALUE 'OCTOBRE  '.
     05  FILLER                           PIC X(9)  VALUE 'NOVEMBRE '.
     05  FILLER                           PIC X(9)  VALUE 'DECEMBRE '.
 01  TABLEAU-MOIS  REDEFINES  MOIS.
     05  NOM-MOIS                         PIC X(9)  OCCURS 12.
*
*
*        TABLEAU DES DESCRIPTIONS DES ARTICLES
*
*
 01  TABLEAU-DESCR.
     05  DESCR-ART                        OCCURS 100.
```

```
          10  CLE-ELEMENT.
              15  SERIE-ELEM          PIC 9.
              15  CODE-ELEM           PIC 9.
          10  DESCRIPTION             PIC X(30).
*
*                  = = =  I N D I C E S  = = =
*
*
 01  INDICE-ARTICLE                   PIC 9      COMP VALUE  0.
 01  INDICE-SERIE                     PIC 9      COMP VALUE  0.
 01  INDICE-MOIS                      PIC 9(2)   COMP VALUE  0.
 01  INDICE-DESCR                     PIC 9(2)   COMP VALUE  0.
 01  LIMITE-TAB-DESCR                 PIC 9(3)   COMP VALUE  100.
*
*
*                  = = =  U T I L I T A I R E S  = = =
*
 01  CLE.
     05  SERIE-CLE                    PIC 9.
     05  CODE-CLE                     PIC 9.
*
*
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 OUVRIR-FICHIER-DESCR.
     OPEN INPUT CARTES-DESCR.
 PREMIERE-LECTURE-DESCR.
     READ CARTE-DESCR INTO CARTE-DESCR-WS
                      AT END
                      DISPLAY '** FICHIER DESCRIPTIONS VIDE **'
                              UPON PRINTER
                      CLOSE CARTES-DESCR
                      GO TO FIN-CALCUL.
 BOUCLE-DESCRIPTIONS
     ADD 1 TO INDICE-DESCR.
     IF  INDICE-DESCR > LIMITE-TAB-DESCR
         CLOSE CARTES-DESCR
         DISPLAY '** TABLEAU DESCRIPTIONS INSUFFISANT **'
                 UPON PRINTER
         DISPLAY '** NOMBRE MAXIMUM DE DESCRIPTIONS PREVU = 100'
                 UPON PRINTER
         GO TO FIN-CALCUL.
     MOVE CARTE-DESCR-WS TO DESCR-ART (INDICE-DESCR).
     READ CARTES-DESCR INTO CARTE-DESCR-WS
                       AT END
                       CLOSE CARTES-DESCR
                       DISPLAY '** DESCRIPTIONS CHARGEES = '
                               INDICE-DESCR
                               UPON PRINTER
                       GO TO CHARGER-DONNEES.
     GO TO BOUCLE-DESCRIPTIONS.
 CHARGER-DONNEES.
     OPEN INPUT CARTES.
     MOVE LOW-VALUES TO TAB-TOTAUX-VENTES.
 PREMIERE-LECTURE-DONNEES.
     READ CARTES INTO CARTE-WS
                 AT END
                 CLOSE CARTES
                 DISPLAY '** FICHIER CARTES DONNEES VIDE **'
                         UPON PRINTER
                         FIN-CALCUL.
 SOMME.
     MOVE SERIE-ARTICLE     TO INDICE-SERIE.
     MOVE CODE-ARTICLE      TO INDICE-ARTICLE.
     MOVE MOIS              TO INDICE-MOIS.
     ADD QUANTITE TO TOTAL-MENSUEL (INDICE-SERIE, INDICE-ARTICLE
      , INDICE-MOIS).
 DONNEES SUIVANTES.
     READ CARTES INTO CARTE-WS
                 AT END
```

```
                    CLOSE CARTES
                    PERFORM AFFICHAGE-TOTAUX
                    GO TO FIN-CALCUL.
      GO TO SOMME.
*
*
*
 FIN-CALCUL.
      STOP RUN.
*
*
*
 AFFICHAGE-TOTAUX SECTION.
 AFFI-TOT.
      MOVE ZEROES TO INDICE-SERIE.
 AFFICHAGE SERIE.
      ADD 1 TO INDICE-SERIE.
      IF  INDICE-SERIE > 9
          GO TO AFFI-TOT-EX.
      MOVE ZEROES TO INDICE-ARTICLE.
 AFFICHAGE ARTICLE.
      ADD 1 TO INDICE-ARTICLE.
      IF  INDICE-ARTICLE > 9
          GO TO AFFICHAGE-SERIE.
      MOVE ZEROES TO INDICE-DESCR.
      MOVE INDICE-SERIE   TO SERIE-CLE.
      MOVE INDICE-ARTICLE TO CODE-CLE
 RECHERCHER-DESCRIPTION.
      ADD 1 TO INDICE-DESCR.
      IF  INDICE-DESCR > LIMITE-TAB-DESCR
          DISPLAY 'ARTICLE SERIE    : '
                  INDICE-SERIE
                  ' CODE            : '
                  INDICE-ARTICLE
                  ' ** DESCRIPTION NON TROUVEE **'
                  ' * * * TOTAUX VENTES * * *'
                  UPON PRINTER
          GO TO ECRIRE-TOTAUX.
      MOVE INDICE-SERIE TO SERIE-CLE.
      MOVE CODE-ARTICLE TO CODE-CLE.
      IF CLE NOT = CLE-ELEMENT (INDICE-DESCR)
          GO TO RECHERCHER-DESCRIPTION.
      DISPLAY 'ARTICLE : '
              DESCRIPTION (INDICE-DESCR)
              ' * * *  TOTAUX VENTES * * * '
              UPON PRINTER.
 ECRIRE-TOTAUX.
      MOVE ZEROES TO INDICE-MOIS;
 AFFICHAGE-VENTES.
      ADD 1 TO INDICE-MOIS.
      IF  INDICE-MOIS > 12
          GO TO AFFICHAGE-ARTICLE.
      DISPLAY '
              NOM-MOIS (INDICE-MOIS)
              ' = '
              TOTAL-MENSUEL (INDICE-SEZRIE, INDICE-ARTICLE
     , INDICE-MOIS)
              UPON PRINTER.
      GO TO AFFICHAGE-VENTES.
 AFFI-TOT-EX. EXIT.
*
*
*
```

Les indices des tableaux

Pour adresser un champ dans un tableau, nous avons, jusqu'à présent, utilisé des champs définis dans la WORKING-STORAGE SECTION : pensons, par exemple, à NOM-MOIS et à INDICE-MOIS (tableau des pages 1108 à 1110).Le compilateur traite ainsi les champs en question en fonction de leurs caractéristiques et non en se fondant sur la fonction d'indice que le programmeur lui réserve. Autrement dit, le programmeur est en droit d'utiliser le champs INDICE-MOIS pour n'importe quelle opération autorisée par le Cobol, sans pour autant entraîner d'erreurs de compilation. En effet, la fonction d'indice associée à ce champ ne lui a pas été déclarée.
INDICE-MOIS, INDICE-ARTICLE décrits dans la WORKING-STORAGE SECTION et utilisés par le programmeur comme tels sont appelés **champs souscrits**.
Le Cobol prévoit la possibilité d'associer, à un tableau, un ou plusieurs indices reconnus comme tels par le compilateur et gérés par lui de manière spécifique : **un indice,** en effet, **n'a pas besoin de définition dans la WORKING-STORAGE SECTION**. Sa déclaration au compilateur est réalisée pendant la prescription du tableau auquel l'indice est lié. Considérons le TABLEAU-MOIS de l'exemple précédent. Pour pouvoir lui associer un indice considéré comme tel, il est nécessaire d'adopter la notation indiquée dans le listing ci-dessous.
Le champ IND-MOIS, associé de manière univoque au tableau par la clause INDEXED BY, est utilisable seulement pour adresser les champs de TABLEAU-MOIS. Chacun d'eux peut, du reste, être adressé non seulement par IND-MOIS mais aussi par n'importe quel subscript (champ réservé).
En résumé, quand TABLEAU-MOIS est décrit comme ci-dessus, les trois instructions :

MOVE NOM-MOIS (3) TO .....
MOVE NOM-MOIS (IND-MOIS) TO .....
MOVE NOM-MOIS (INDICE-MOIS) TO .....

sont valables.
L'adressage est obtenu dans la première instruction avec une constante (3), dans la deuxième avec l'indice du tableau (IND-MOIS) et dans la troisième avc un subscript (INDICE-MOIS).
Notons, enfin, que plusieurs indices peuvent être associés à un tableau, les limitations décrites restant bloquées pour chacun d'eux.
On pourrait ainsi écrire :

```
.
01 TABLEAU-MOIS REDEFINES MOIS.
   05 NOM-MOIS  PIC X (9) OCCURS 12
                INDEXED BY
                IND-MOIS
                NOMBRE-MOIS
                POINTEUR
                .....
```

Le verbe SET pour la gestion d'un indice.
L'indice associé à un tableau par la clause INDEXED BY ne demande pas de description dans la WORKING-STORAGE parce que le compilateur réserve, à cet effet, un de ses registres internes. Ce registre assure une gestion automatique lorsque apparaissent des

DESCRIPTION D'UN TABLEAU AVEC INDICE GERE PAR LE COMPILATEUR

```
01  MOIS .
    05  FILLER                  PIC X(9)  VALUE 'JANVIER  '.
    05  FILLER                  PIC X(9)  VALUE 'FEVRIER  '.
    05  FILLER                  PIC X(9)  VALUE 'MARS     '.
    05  FILLER                  PIC X(9)  VALUE 'AVRIL    '.
    05  FILLER                  PIC X(9)  VALUE 'MAI      '.
    05  FILLER                  PIC X(9)  VALUE 'JUIN     '.
    05  FILLER                  PIC X(9)  VALUE 'JUILLET  '.
    05  FILLER                  PIC X(9)  VALUE 'AOUT     '.
    05  FILLER                  PIC X(9)  VALUE 'SEPTEMBRE'.
    05  FILLER                  PIC X(9)  VALUE 'OCTOBRE  '.
    05  FILLER                  PIC X(9)  VALUE 'NOVEMBRE '.
    05  FILLER                  PIC X(9)  VALUE 'DECEMBRE '.
01  TABLEAU-MOIS  REDEFINES  MOIS.
    05  NOM-MOIS                PIC X(9)  OCCURS 12
                                INDEXED BY IND-MOI.
```

instructions dont on parlera un peu plus loin. A cause de sa particularité, le programme ne peut intervenir sur son contenu avec les instructions Cobol employées pour les subscripts (MOVE, ADD...).
Ainsi recourt-il à l'instruction SET qui permet les opérations suivantes :

- positionner l'indice sur une valeur déterminée,
- incrémenter d'un pas la valeur de l'indice,
- décrémenter l'actuelle valeur de l'indice.

Les 3 fonctions décrites sont assumées par les formats du tableau du bas de cette page.
Notons que les valeurs attribuées par un indice dépendent des dimensions du tableau. Dans TABLEAU-MOIS, l'indice IND-MOIS n'est assumé que par les valeurs de 1 à 12, en fonction du nombre d'éléments déclarés dans le tableau. Celles qui se trouvent en dehors de cet intervalle rendent imprévisible le contenu de l'indice.
Sur la base de ce que l'on a vu jusqu'à présent, le verbe SET et les indices qui lui sont associés pourraient paraître comme inutilement redondants dans la gestion d'un tableau. En réalité, il a son utilité.

Le verbe SEARCH pour la recherche dans les tableaux

Considérons à nouveau l'exemple développé jusqu'à présent pour le traitement des tableaux. Une clé de recherche, avec les nombres de série et d'article, a été composée pour pouvoir personnaliser la description d'un article.
La clé a été successivement comparée aux deux premiers caractères des 100 éléments du tableau des descriptions. Si elle avait été égale à la clé de l'élément, la partie description aurait été prélevée ; sinon on augmenterait l'indice en vue d'une nouvelle comparaison.
Ce processus itératif s'interrompt une fois l'élément trouvé, ou bien lorsque l'indice dépasse le nombre d'éléments admis par le tableau même.
Notre exemple peut comporter jusqu'à 9 séries de 9 types d'articles, soit 81 descriptions. Dès lors, les comparaisons du 82e ou 100e sont sans objet ; elles ne feraient que rallonger bien inutilement les temps de calcul.
D'autre part, il a été imposé que le chargement des descriptions de tous les articles et séries possibles soit effectué d'office.
A défaut, le nombre de descriptions aurait été inférieur à 81 et les comparaisons excessivement plus nombreuses que nécessaires.
Parvenir à limiter la recherche aux seuls éléments effectivement chargés (sans balayer le tableau dans sa totalité) est particulièrement avantageux lorsqu'on a affaire à des tableaux volumineux, surtout s'ils ne sont que très partiellement remplis.
Pour limiter la recherche aux seuls éléments chargés, il suffit d'enregistrer leur nombre dans un champs utilitaire approprié au moment du chargement. Le tableau sera alors considéré comme complet, lors de la phase suivante de recherche (quand l'indice aura dépassé le contenu du champ utilitaire).
On peut ainsi réécrire le paragraphe BOUCLE-

FORMATS DE L'INSTRUCTION SET

SET nom-indice TO { constante | nom-donnée | nom-indice-1 }

SET nom-indice UP BY { constante | nom-donnée }

SET nom-indice DOWN BY { constante | nom-donnée }

DESCRIPTIONS de l'exemple précédent dans lequel le chargement des descriptions des articles était effectué par le fichier de cartes CARTES-DESCR. :

```
BOUCLE-DESCRIPTIONS.
   ADD 1 TO INDICE-DESCR.
   IF INDICE-DESCR    LIMITE-TAB-DESCR
      CLOSE CARTES-DESCR
      DISPLAY 'TABLEAU DESCRIPTIONS
                  INSUFFISANT'
                  UPON PRINTER
                  GO TO FIN-TRAITEMENT.
   MOVE CARTE-DESCR-WS
   TO DESCR-ART (INDICE-DESCR).
   READ CARTES-DESCR
   INTO CARTES-DESCR-WS
         AT END
         MOVE INDICE-DESCR
                  TO COMPTER-ELEMENTS
                  CLOSE CARTES-DESCR
                  GO TO CHARGER-DONNEES
   GO TO BOUCLE-DESCRIPTIONS.
CHARGER-DONNEES.
   .....
```

A la fin de la lecture du fichier CARTES-DESCR, INDICE-DESCR (qui contient encore le numéro de position du dernier élément chargé) est sauvegardé dans le champ utilitaire COMPTER-ELEMENTS supposé décrit ainsi :

```
01 COMPTER-ELEMENTS    PIC 9(3) COMP.
```

Avec ces préliminaires, le paragraphe RECHERCHER-DESCRIPTION est écrit ainsi dans la section DISPLAY-TOTAUX :

```
RECHERCHER-DESCRIPTION.
   ADD 1 TO INDICE-DESCR.
   IF INDICE-DESCR    COMPTER-ELEMENTS
      DISPLAY 'ARTICLE SERIE'
                  INDICE-SERIE
                  'CODE'
                  INDICE-ARTICLE
                  '*** DESCRIPTION
                  NON TROUVEE ***'
                  UPON PRINTER
                  GO TO ECRIRE-TOTAUX.
.....
.....
```

La description est considérée comme « non trouvée » dans le tableau soit quand tous les éléments réellements présents ont été balayés, soit quand l'indice a dépassé la valeur contenue dans COMPTER-ELEMENTS. Tout cela constitue un préliminaire à l'examen de l'instruction SEARCH et des tableaux à dimension variable.

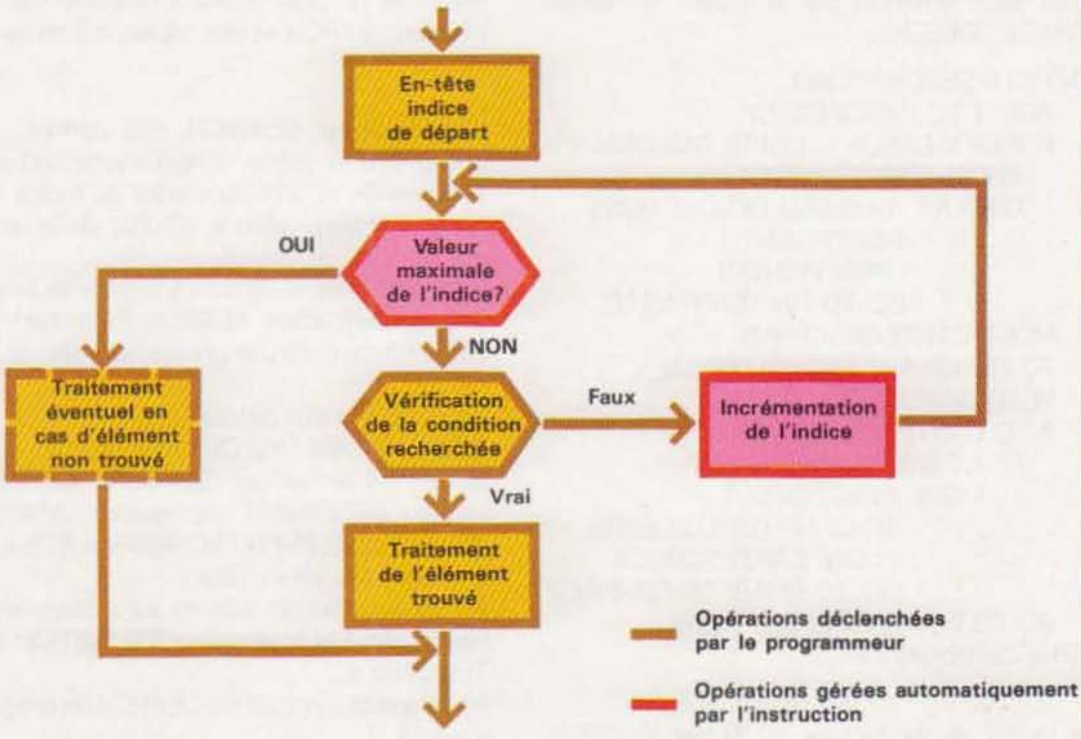
L'instruction SEARCH. Elle permet, dans une première forme, d'opérer une recherche séquentielle et d'entreprendre au moins deux types d'actions selon le résultat de la recherche (négatif ou positif).
En accord avec le caractère familier du langage Cobol, l'instruction SEARCH (chercher) peut être traduite dans une phrase normale, de type humain.
La recherche des descriptions de l'article dans le tableau TABLEAU-DESCR équivaut à l'ordre suivant : « rechercher (SEARCH) l'élément du tableau DESCR-ART et, quand (WHEN) le champ CLE-ELEMENT est égal à la CLE, écrire la description de l'article.
Arrivé à la fin du tableau sans avoir trouvé égalité (AT END), signaler DESCRIPTION NON TROUVEE ».
En adoptant l'instruction SEARCH, le programme doit :

1 / imposer la valeur de l'indice correspondant à la position de départ de la recherche. Normalement, on balaie le tableau à partir du premier élément (SET indice TO 1) ;
2 / indiquer le type d'action à entreprendre quand la condition de recherche est validée ;
3 / spécifier l'action à lancer dans le cas où l'élément recherché est absent du tableau.

Le mécanisme de fonctionnement du SEARCH séquentiel est illustré par le schéma de la page 1114.
Pour éclairer l'emploi de SEARCH, il convient d'examiner les deux versions du paragraphe RECHERCHER-DESCRIPTION (voir listings pages 1114 et 1115) avec, pour définition de TABLEAU-DESCR :

```
01 TABLEAU-DESCR.
   05 DESCR-ART    OCCURS 100
                  INDEXED BY IND-DES.
      10 CLE-ELEMENT.
         15 SERIE-ELEM          PIC 9.
         15 CODE-ELEM           PIC 9.
      10 DESCRIPTION            PIC X(30).
```

EXECUTION D'UN INSTRUCTION SEARCH SEQUENTIELLE



RECHERCHE DANS UN TABLEAU AU MOYEN D'UN SUBSCRIPT

```
      .
      .
      .
      MOVE ZEROES TO INDICE-DESCR.
  RECHERCHER-DESCRIPTION.
      ADD 1 TO INDICE-DESCR.
      IF  INDICE-DESCR GREATER THAN LIMITE-TAB-DESCR
          DISPLAY 'ARTICLE SERIE '
                  INDICE-SERIE
                  'CODE '
                  INDICE-ARTICLE
                  ' *** DESCRIPTION NON TROUVEE ***'
                  UPON PRINTER
                  GO TO ECRIRE-TOTAUX.
  *
  *
   COMPOSER-CLE.
      MOVE INDICE-SERIE    TO SERIE-CLE.
      MOVE INDICE-ARTICLE  TO CODE-CLE.
  COMPARER-CLE.
      IF  CLE NOT = CLE-ELMENT (INDICE-DESCR)
          GO TO RECHERCHER-DESCRIPTION.
      DISPLAY 'ARTICLE : '
              DESCRIPTION (INDICE-DESCR)
              '*** TOTAUX VENTES ***'
              UPON PRINTER.
  *
  *
   ECRIRE-TOTAUX.
      ..................
```

RECHERCHE DANS UN TABLEAU AU MOYEN DE SEARCH SEQUENTIEL

```
 RECHERCHER-DESCRIPTION.
     SET IND-DES TO 1.
     MOVE INDICE-SERIE    TO SERIE-CLE.
     MOVE INDICE-ARTICLE  TO CODE-CLE.
*
*
     SEARCH DESCR-ART
            AT END DISPLAY 'ARTICLE SERIE '
                           INDICE-SERIE
                           ' CODE '
                           INDICE-ARTICLE
                           ' *** DESCRIPTION NON TROUVEE ***'
                           UPON PRINTER
            WHEN CLE-ELEMENT (IND-DES) = CLE
                 DISPLAY 'ARTICLE : '
                           DESCRIPTION (IND-DES)
                           ' *** TOTAUX VENTES ***'
                           UPON PRINTER.
*
*
 ECRIRE-TOTAUX.
     ..................
```

FORMAT GENERAL DE L'INSTRUCTION SEARCH

```
SEARCH nom-élément [ VARYING { nom-indice    } ] [ AT END { phrase-impérative-1 } ]
                             { nom-donnée    }             { NEXT SENTENCE     }

     WHEN condition-1 { phrase-impérative-2 }
                      { NEXT SENTENCE     }

   [ WHEN condition-2 { phrase-impérative-3 } ]  .....
                      { NEXT SENTENCE     }
```

Dans la version recherche gérée par le programmeur (par l'intermédiaire de INDICE-DESCR), on utilise la limite maximale des dimensions du tableau et non pas celle de remplissage (COMPTER-ELEMENTS). De cette manière, et d'un point de vue fonctionnel, les deux versions du paragraphe sont complètement équivalentes.

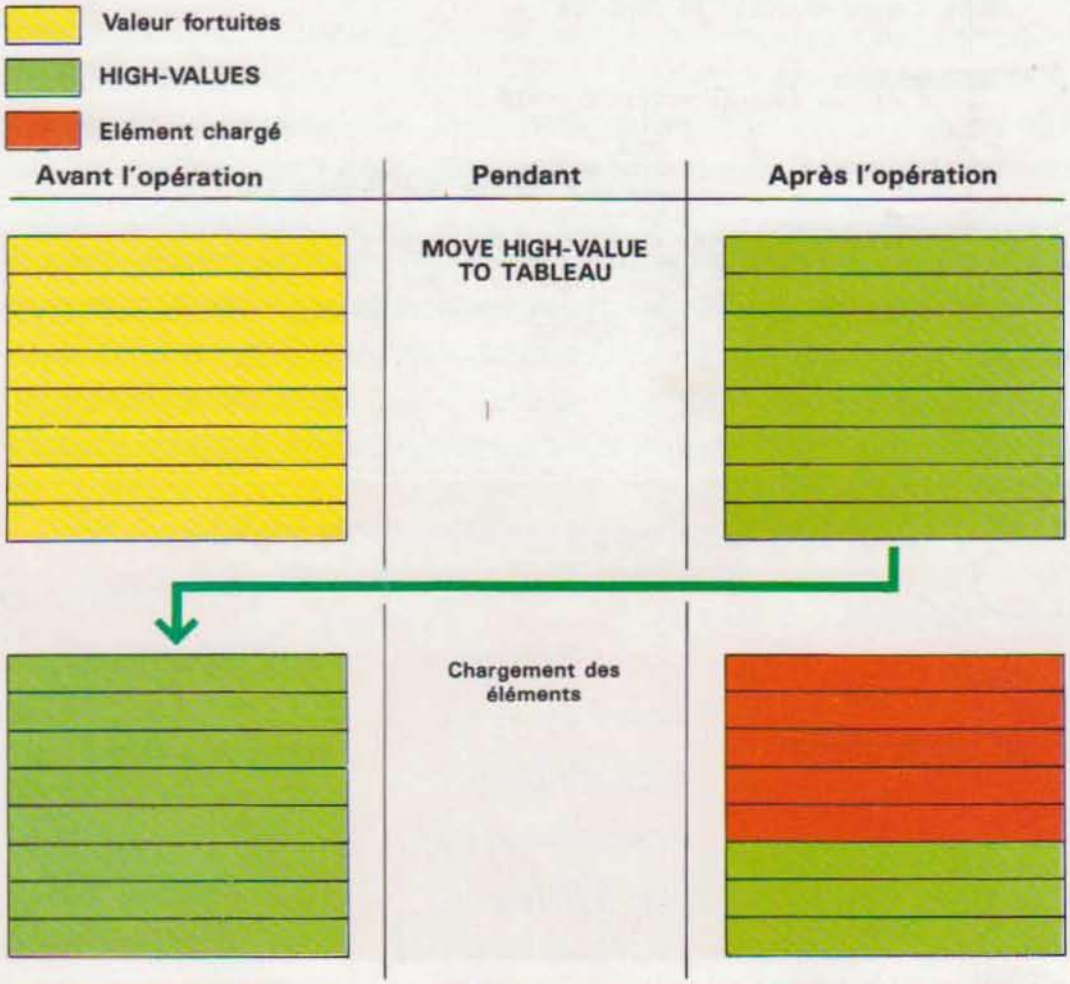
Le format général de l'instruction séquentielle SEARCH est donné en page 1115.

Nom-élément est le terme générique de l'élément dans le tableau où s'effectue la recherche. Quand la clause AT END n'est pas spécifiée et si la condition de l'élément trouvé n'a jamais été vérifiée, le contrôle passe à la première instruction après le point de fermeture du SEARCH à exécuter. La clause NEXT SENTENCE se comporte exactement comme la clause analogue de l'instruction IF, tandis que VARYING permet à chaque hausse de l'indice d'incrémenter automatiquement un autre indice ou nom-donnée.

Emploi du SEARCH séquentiel. Le problème créé par l'arrêt du balayage du tableau au-delà des limites de son remplissage réel (problème déjà affronté dans la recherche par l'intermédiaire de subscripts), peut être résolu en adoptant le SEARCH.

Dans la première méthode (page 1116), on impose HIGH-VALUES à tous les éléments du tableau ; les éléments réels sont ensuite chargés. La condition de fin de recherche sera vérifiée si le tableau est réellement terminé ou si le

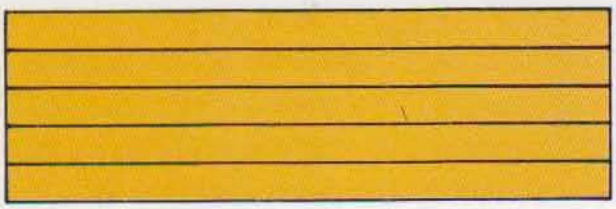
SCHEMATISATION DU TABLEAU MEMOIRE PENDANT LE CHARGEMENT




```
PROCEDURE DIVISION.
    .
    .
    .
     MOVE HIGH-VALUES  TO  TABLEAU.
     PERFORM CHARGER-TABLEAU.
        .
        .
        .
CALCULER.
        .
        .
        .
RECHERCHER-ELEMENT.
     SEARCH ELEMENT
            AT END
            PERFORM NON-TROUVE
     WHEN ELEMENT (INDICE) = HIGH-VALUES
            PERFORM NON-TROUVE
     WHEN CLE-ELEMENT (INDICE) = CLE-RECHERCHE
            PERFORM CALCULER-ELEMENT.
```

DESCRIPTION D'UN TABLEAU A DIMENSION VARIABLE DANS WORKING-STORAGE SECTION

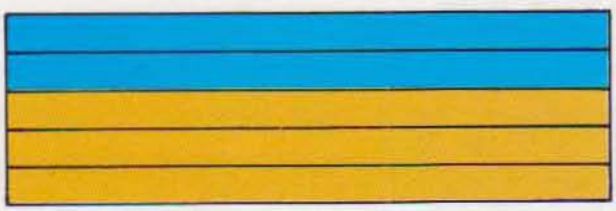
```
 01  DIMENSION                    PIC 9  COMP.
*
 01  TABLEAU.
     05  ELEMENT                  PIC X(10)
                                  OCCURS 1 TO 5
                                  DEPENDING ON DIMENSION
                                  INDEXED BY IND-TAB.
```



Représentation schématique de l'espace physique assigné au tableau

```
MOVE  2  TO  DIMENSIONE.
```

Limitation à 2 éléments de la dimension du tableau



Représentation schématique de l'espace physique assigné au tableau et de la partie (établie par DIMENSION=2) sur laquelle les instructions (par exemple SEARCH) s'activent

dernier élément examiné est égal à HIGH-VALUE.
Les pas à effectuer au cœur du programme pour limiter la recherche aux seuls éléments présents dans le tableau sont réduits, comme indiqué page 1116.
Une seconde méthode, pour minimiser les temps de recherche séquentielle, exploite les possibilités prévues par le Cobol de déclarer au compilateur le nombre d'éléments concernés par la recherche.
Cette dernière méthode exige de définir le tableau comme un tableau à dimensions variables. Le nombre de ses éléments est spécifié par la valeur d'un champ numérique attribuée par le programmeur.
La description de ce type de tableau nécessite les paramètres suivants :

- le nombre minimal d'éléments contenus par le tableau,
- le nombre maximal d'éléments contenus par le tableau,
- le nom du champ numérique dont la valeur établit les dimensions réelles du tableau.

La description d'un tableau à dimensions variables est conforme à la forme :

01 TABLEAU.
05 ELEMENT OCCURS minimum TO maximum TIMES DEPENDING ON nom-champ.

minimum C'est une constante supérieure à 0 sans signe et inférieure à « maximum »

nom-champ C'est le nom d'un champ numérique ne pouvant assumer que des valeurs comprises entre « minimum » et « maximum ».

L'expression « tableau à dimensions variables » ne doit pas être interprétée comme la possibilité d'étendre l'espace occupé en mémoire pendant le chargement.

RECHERCHE SEQUENTIELLE DANS UN TABLEAU

```
.
WORKING-STORAGE SECTION.
.
.
.
01  COMPTEUR-ELEMENTS                    PIC 9(3)  COMP VALUE 0.
01  TABLEAU-DESCR.
    05  DESCR-ART                        OCCURS 1 TO 100
                                         DEPENDING ON COMPTEUR-ELEMENTS
                                         INDEXED BY IND-DES.
        10  CLE-ELEMENT.
            15  SERIE-ELEM               PIC 9.
            15  CODE-ELEM                PIC 9.
        10  DESCRIPTION                  PIC X(30).
01  INDICE-DESCR                         PIC 9(3)  COMP VALUE 0.
01  LIMITE-TAB-DESCR                     PIC 9(3)  COMP VALUE 100.
01  CLE.
    05  SERIE-CLE                        PIC 9.
    05  CODE-CLE                         PIC 9.
*
*
*
PROCEDURE DIVISION.
.
.
CHARGER-DESCRIPTIONS.
    ADD  1  TO  INDICE-DESCR.
    IF  INDICE-DESCR > LIMITE-TAB-DESCR
        CLOSE CARTES-DESCR
        DISPLAY 'TABLEAU DESCRIPTIONS INSUFFISANT'
                UPON PRINTER
        GO TO FIN-CALCUL.
    MOVE CARTE-DESCR-WS TO DESCR-ART (INDICE-DESCR).
    READ CARTES-DESCR INTO CARTE-DESCR-WS
                      AT END
                      MOVE INDICE-DESCR TO COMPTEUR-ELEMENTS
                      CLOSE CARTES-DESCR
                      GO TO CHARGER-DONNEES.
    GO TO CHARGER-DESCRIPTIONS.
.
.
.
.
*
*
*
CHARGER-DONNEES.
.....................
.....................
.....................
.
.
.
*
*
*
RECHERCHER-DESCRIPTION.
    SET   IND-DES          TO 1.
    MOVE INDICE-SERIE      TO SERIE-CLE.
    MOVE SERIE-ARTICLE     TO CODE-CLE.
    SEARCH DESCR-ART
           AT END
                DISPLAY 'ARTICLE SERIE '
                        INDICE SERIE
                        ' CODE '
                        INDICE-ARTICLE
                        '*** DESCRIPTION NON TROUVEE ***'
                        UPON PRINTER
           WHEN
                CLE-ELEMENT (IND-DES) = CLE
```

```
                    DISPLAY 'ARTICLE : '
                            DESCRIPTION (IND-DES)
                            ' *** TOTAUX VENTES ***'
                            UPON PRINTER.
      .
      .
      .
    *
    *
    *
     ECRIRE-TOTAUX.
     ..................
```

En effet, le compilateur assigne un espace suffisant pour contenir le tableau dimensionné au maximum.
Bien noter que la variabilité des dimensions du tableau (en fonction du contenu de nom-champ) est purement logique, en ce sens que n'importe quelle instruction traitée par le tableau, à un moment déterminé, ne voit que la partie déclarée par la valeur de nom-champ. (Voir schéma ci-dessus).
Reprenons maintenant le programme de l'exemple.
A l'évidence, il reproduit la méthode la plus rapide pour rechercher séquentiellement la description d'un article. A noter que, dans le listing, les parties non concernées par l'application ont été omises.

L'instruction SEARCH ALL (recherche binaire). Nous venons de montrer que l'utilisation du SEARCH séquentiel comporte des avantages relativement modestes par rapport à une recherche avec subscripts entièrement gérée par le programmeur.
Nous allons examiner une autre forme de SEARCH ALL.
Le classement ascendant ou descendant des éléments du tableau est établi en fonction d'un champ clé déclaré.
Reprenons le tableau des descriptions des articles.
Les spécificités du programme prévoient que le fichier concernant les cartes de descriptions (CARTES-DESCR) soit rangé en ordre croissant de série et d'article :

Clé de comparaison

Série	Article	Description
11	CHEMISES	HOMME
12	CHEMISES	FEMME
13	CHEMISES	ENFANT
.		
21	PANTALONS	HOMME
22	PANTALONS	FEMME
23	PANTALONS	ENFANT
.		
91	CHAUSSURES	HOMMES
92	CHAUSSURES	FEMME
.		
99	SANDALES	ENFANT

Après chargement, TABLEAU-DESCR reflétera donc ce classement.
Dans une recherche séquentielle, (par exemple sur un article de clé 23), on procède ainsi :
— la clé de recherche CLE est d'abord comparée avec celle du 1er élément (11),
— ensuite avec celle du 2e (12)
— et ainsi de suite jusqu'au 12e élément dont la clé est 23.
Le nombre d'accès à effectuer dans un tableau pour vérifier l'existence d'un élément est en moyenne égal à N/2 avec N représentant le nombre d'éléments à examiner.
Il est clair que le nombre d'accès composés dans le tableau dépend de la position effective de l'élément recherché : une seule fois si l'élément en question est en 1ère position et n fois s'il se trouve en dernière position du tableau.
La logique de recherche du SEARCH ALL s'apparente au jeu dichotomique du « haut et bas » (voir représentation page 1120 et 1121). Le joueur n° 1 doit deviner un nombre (47, par exemple) compris entre 1 et 100 écrit par le joueur n° 2 sur une feuille de papier.
Le joueur n° 1 peut effectuer des tentatives auxquelles le joueur n° 2 ne peut répondre qu'avec les phrases « trop grand » ou « trop petit » ce qui permet d'éliminer des zones de classement par pans entiers. On en devine immédiatement l'avantage. D'ailleurs les chiffres expérimentaux sont là pour le prouver.
Dans notre exemple, alors qu'une recherche

Joueur N° 1	Commentaire	Joueur N° 2
50	Le joueur n° 1 cherche immédiatement à savoir dans quelle moitié de la série 1 à 100 se trouve le nombre, en effectuant une première tentative sur le numéro central.	Trop grand
25	La réponse est "trop grand". Le nombre recherché est donc dans les 50 premiers. On n'effectue alors plus de tentatives entre 51 et 100. En reprenant le raisonnement précédent, on propose 25, de façon à savoir dans quelle moitié de 1-50 se trouve le nombre.	Trop petit
37	Le joueur n° 1 sait maintenant que le nombre recherché est compris entre 25 et 50, c'est pourquoi il divise encore cet intervalle par deux en proposant 37.	Trop petit
44	La dernière réponse du joueur n° 2 permet de déduire que le nombre est compris entre 37 et 50. C'est pourquoi le joueur n° 1, ajoutant 7 à la moitié de l'intervalle 37-50 à 37, divise en deux ce dernier intervalle.	Trop petit
47	Le nombre doit être compris entre 44 et 50. Toujours en suivant le même critère, on ajoute 3 (50-40) à 44 pour proposer 47.	Exact

séquentielle aurait comporté 47 tentatives avant d'arriver au résultat, la **recherche binaire** a nécessité seulement 5 accès. C'est ce type de recherche appelée également dichotomique (du grec, couper en deux) que réalise le SEARCH ALL.

On comprend, du même coup, pourquoi il est indispensable d'ordonnancer le tableau en sens ascendant ou descendant, en fonction de la clé de recherche. On doit aussi préciser que le classement du tableau se trouve entièrement aux mains du programmeur, de façon à permettre au compilateur d'exclure les parties du tableau devenues inopérantes après chaque tentative.

Par conséquent, la forme décrivant un tableau à soumettre au SEARCH ALL est importante. Une telle forme prévoit, en effet, la déclaration sur le type de classement appliqué au tableau par le programmeur (voir ci-dessous).

La clause ASCENDING/DESCENDING KEY IS, clé-recherche, déclare au compilateur le type de classement appliqué aux tableaux et la partie de l'élément à utiliser comme clé de recherche.

Pour fixer les idées, considérons à nouveau l'exemple du tableau TABLEAU-DESCR contenant les descriptions des articles. Pour appliquer l'instruction SEARCH ALL en phase de recherche des descriptions, la définition du tableau et le paragraphe RECHERCHER-DESCRIPTION peuvent être détaillés comme indiqué dans le listing de la page 1122.

Dans le format de l'instruction SEARCH ALL (haut de la page 1121), nous observons que, contrairement au séquentiel, le SEARCH binaire ne nécessite pas de connaître la position initiale de l'indice (SET indice TO...), puisqu'il est directement géré par l'instruction.

De plus, avec le SEARCH ALL, la seule compa-

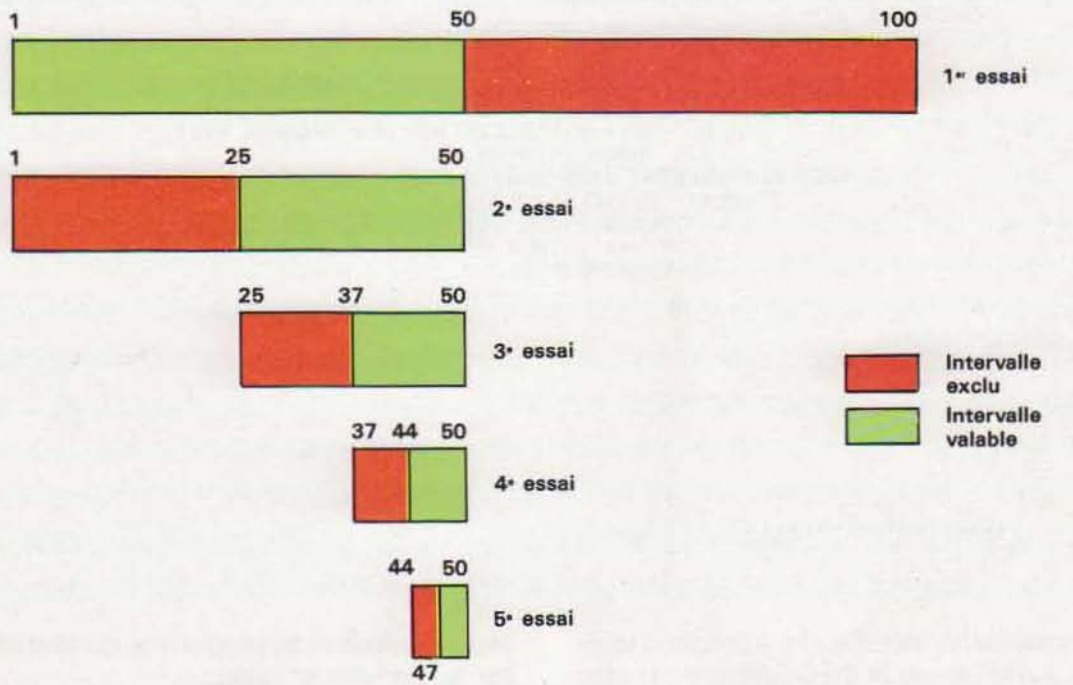
DESCRIPTION D'UN TABLEAU A SOUMETTRE A SEARCH ALL

```
01 nom-tableau
   05 nom-élément    OCCURS minimum TO maximum
                     DEPENDING ON dimensions
                    { ASCENDING  }
                    { DESCENDING } KEY IS clé-recherche
                     INDEXED BY nom-indice
```

FORMAT DE L'INSTRUCTION SEARCH ALL

```
SEARCH ALL nom-élément
           AT END
                  { phrase-impérative-1 }
                  { NEXT-SENTENCE       }
           WHEN   clé-recherche  { constante  }
                                 { nom-champ  }
                                 { expression }
                  { phrase-impérative }
                  { NEXT SENTENCE     }
```

RECHERCHE DICHOTOMIQUE DU NOMBRE 47 DANS L'INTERVALLE 1 ÷ 100



raison admise concerne l'égalité entre la clé de recherche et la clé de l'élément dans le tableau. Une telle limitation n'existe pas en SEARCH séquentiel puisque ce dernier admet également des comparaisons pour supérieur et inférieur.

Classement d'un tableau

En Cobol, la consultation d'un tableau est courante, pour des motifs très variés.
L'exposé sur la recherche dans les tableaux et sur les outils mis en œuvre (SEARCH et SEARCH ALL) a fait ressortir les avantages associés à la manipulation de données par l'intermédiaire de tableaux.
Laissons de côté les cas simples de tableaux contenant des valeurs fixes, comme les noms de mois et de jours de la semaine.

L'utilisation d'un tableau, surtout de grandes dimensions, doit être attentivement conduite, en particulier quand l'ordinateur n'est pas très puissant.

Le SEARCH ALL permet en effet d'accélérer les procédures de recherche en réduisant au

RECHERCHE DICHOTOMIQUE DANS UN TABLEAU

```
 01  TABLEAU-DESCR.
     05  DESCR-ART                      OCCURS 1 TO 100
                                        DEPENDING ON COMPTE-ELEMENTS
                                        ASCENDING KEY IS CLE-ELEMENT
                                        INDEXED BY IND-DES.
         10  CLE-ELEMENT.
             15  SERIE-ELEM            PIC 9.
             15  CODE-ELEM             PIC 9.
         10  DESCRIPTION               PIC X(30).
*
*
*
      .
      .
      .
*
*
 RECHERCHER-DESCRIPTION.
     MOVE INDICE-SERIE    TO SERIE-CLE.
     MOVE INDICE-ARTICLE  TO CODE-CLE.
     SEARCH ALL DESCR-ART
                AT END DISPLAY 'ARTICLE SERIE '
                               INDICE-SERIE
                               ' CODE '
                               INDICE-ARTICLE
                               ' *** DESCRIPTION NON TROUVEE ***'
                               UPON PRINTER
                WHEN CLE-ELEMENT (IND-DES) = CLE
                     DISPLAY 'ARTICLE : '
                             DESCRIPTION (IND-DES)
                             ' *** TOTAUX VENTES ***'
                             UPON PRINTER.
*
*
 ECRIRE-TOTAUX.
     ..................
```

minimum le nombre d'accès, à condition toutefois que le tableau ait été préalablement ordonné, selon une clé déterminée.
Si les données à charger dans le tableau ne sont pas déjà ordonnées selon un certain ordre croissant ou décroissant, le problème de savoir si l'on adopte une recherche de type séquentiel ou binaire se pose.
Avant de passer à l'utilisation de ce dernier mode, il faut en effet prévoir une phase de classement des données, qui peut être conduite de trois manières différentes.

— On peut ordonner le fichier de données selon une clé, grâce à des programmes de tri. L'opération de classement doit être effectuée avant l'exécution du programme qui doit utiliser les données en question.
Pour fixer les idées, prenons l'exemple du fichier de cartes CARTES-DESCR, contenant les descriptions des articles.
Si l'on adopte une sélection externe, l'organisation des données sera celle illustrée dans la page suivante.
— Alternativement, on peut ordonner le fichier données selon la clé désirée, en recourant à la sélection (tri) interne des données (si le compilateur ne l'a pas).
Ce mode de sélection sera plus largement décrit dans le chapitre suivant (voir organigramme page 1124).
— Comme dernière solution, on peut ordonner

directement le tableau en mémoire, tout de suite après son chargement par l'intermédiaire d'un algorithme quelconque géré par le programmeur (voir organigramme page 1125).

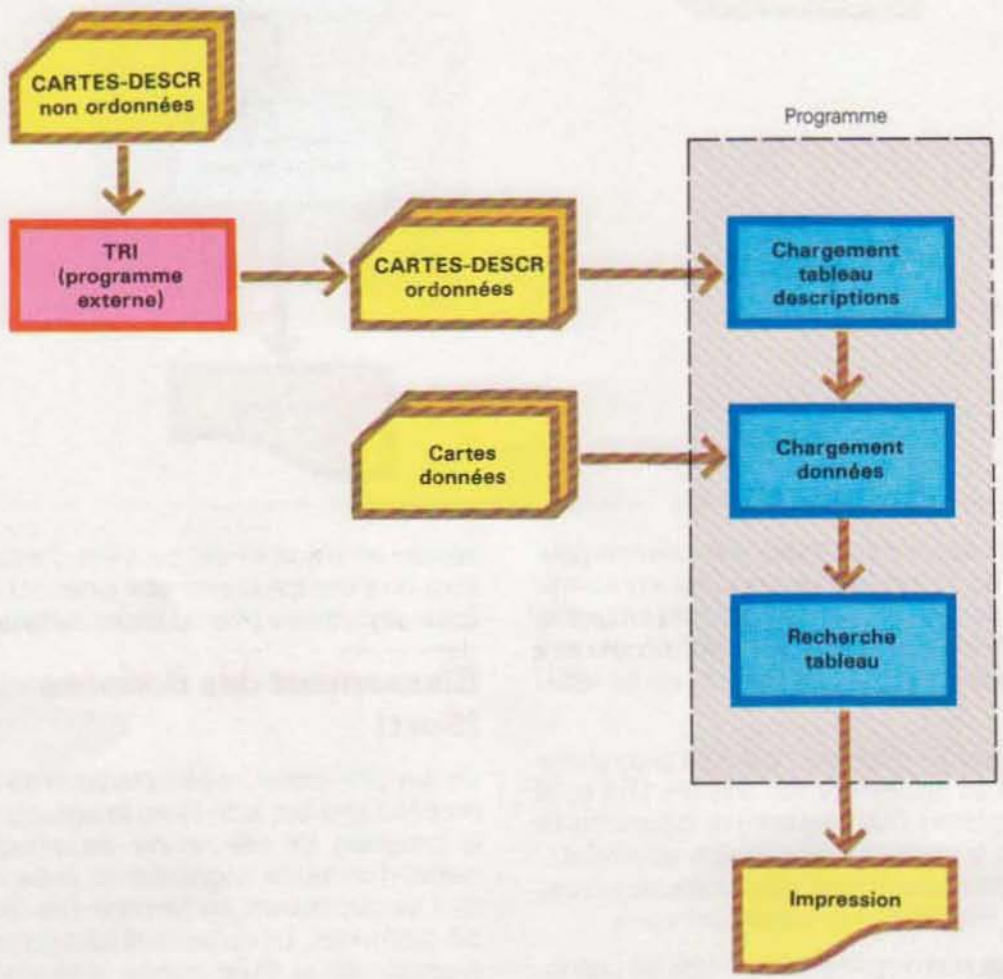
Les trois méthodes décrites présentent des avantages et des inconvénients pour ce qui concerne le temps utilisé et les ressources de calcul demandées (mémoire centrale et mémoire de masse). Indépendamment de la méthode choisie, il est normal de se demander : « est-il juste de pénaliser en temps le programme employé pour sélectionner les données dans l'intention d'utiliser le SEARCH ALL ; ne convient-il pas plutôt d'adopter une recherche plus lente sans être obligé d'ordonner les données d'entrée ? »

Il n'existe pas de réponse univoque à cette question puisque le choix dépend du type de traitement.

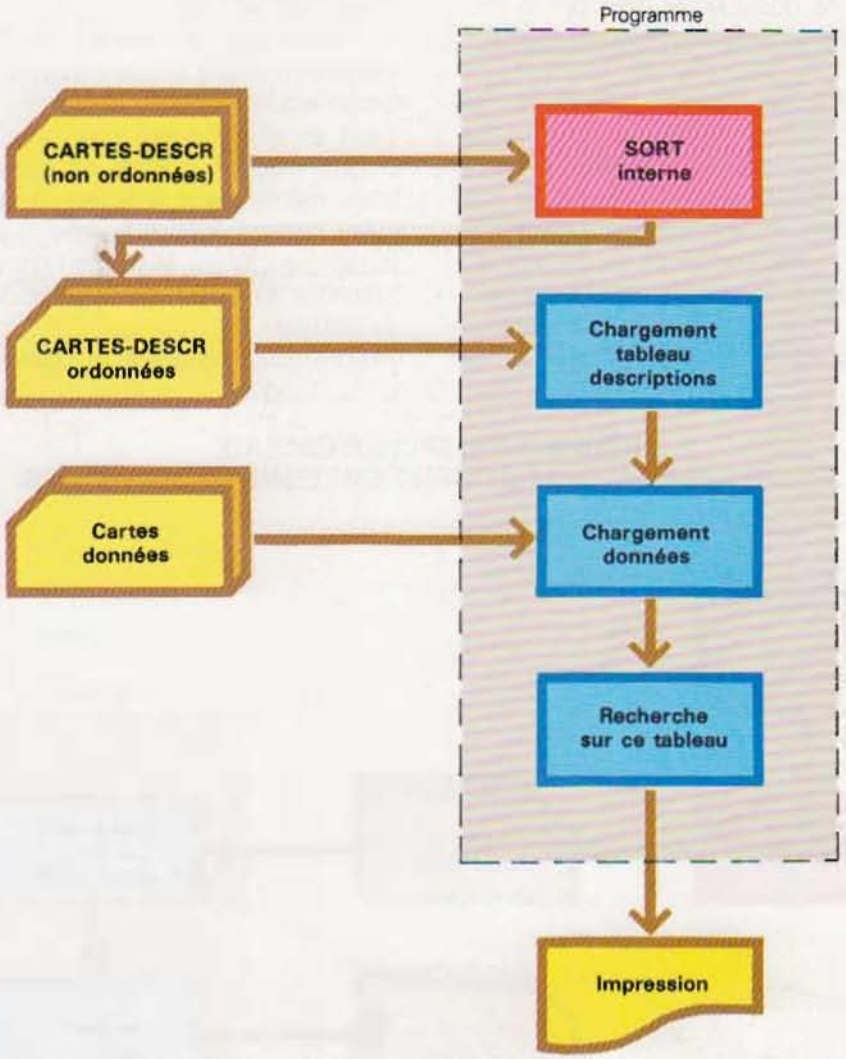
Il est en effet évident que, si au cours d'un programme, les besoins de consulter un tableau, même grand, sont peu nombreux, il vaut mieux recourir à la recherche séquentielle, en évitant de charger le classement des enregistrements du fichier d'entrée ou des éléments du tableau.

Inversement, si le traitement demande d'accé-

RECHERCHE DICHOTOMIQUE
PREALABLE AU CLASSEMENT EXTERNE DES DONNEES



RECHERCHE DICHOTOMIQUE PREALABLE AU CLASSEMENT INTERNE DES DONNEES



der très souvent au tableau (par exemple pour contrôles la validité de tous les enregistrements lus par un fichier d'entrée), la recherche séquentielle s'avère extrêmement pénible et il est donc opportun de recourir au SEARCH ALL.

Si le système n'est pas doté d'un programme destiné au classement des données (Tri) ou si le compilateur Cobol est privé de la fonction Tri interne, la seule solution qui reste, au programmeur, pour sélectionner les données du tableau est de réaliser seul le classement voulu.

Dans les micro-ordinateurs et dans les ordinateurs personnels, le champ du classement d'un tableau en mémoire est peut-être l'argument dont on s'occupe le plus et il existe de nombreux algorithmes plus ou moins rapides.

Classement des données (Sort)

Un des instruments les plus puissants dont dispose le Cobol est le tri (Sort) interne qui offre la possibilité de sélectionner les enregistrements d'un fichier séquentiel en ordre croissant ou décroissant, en fonction des champs clé déterminés. Le classement peut porter sur plusieurs clés et d'une manière différente.
Soit un fichier cartes décrit ainsi :

```
01 SK.
   05 LOCALITE        PIC X(20).
   05 DEPARTEMENT     PIC X(20).
   05 REGION          PIC X(20).
   05 NB-HABIT        PIC 9(8).
   05 POSITION        PIC X(4).
   05 FILLER          PIC X(8).
```

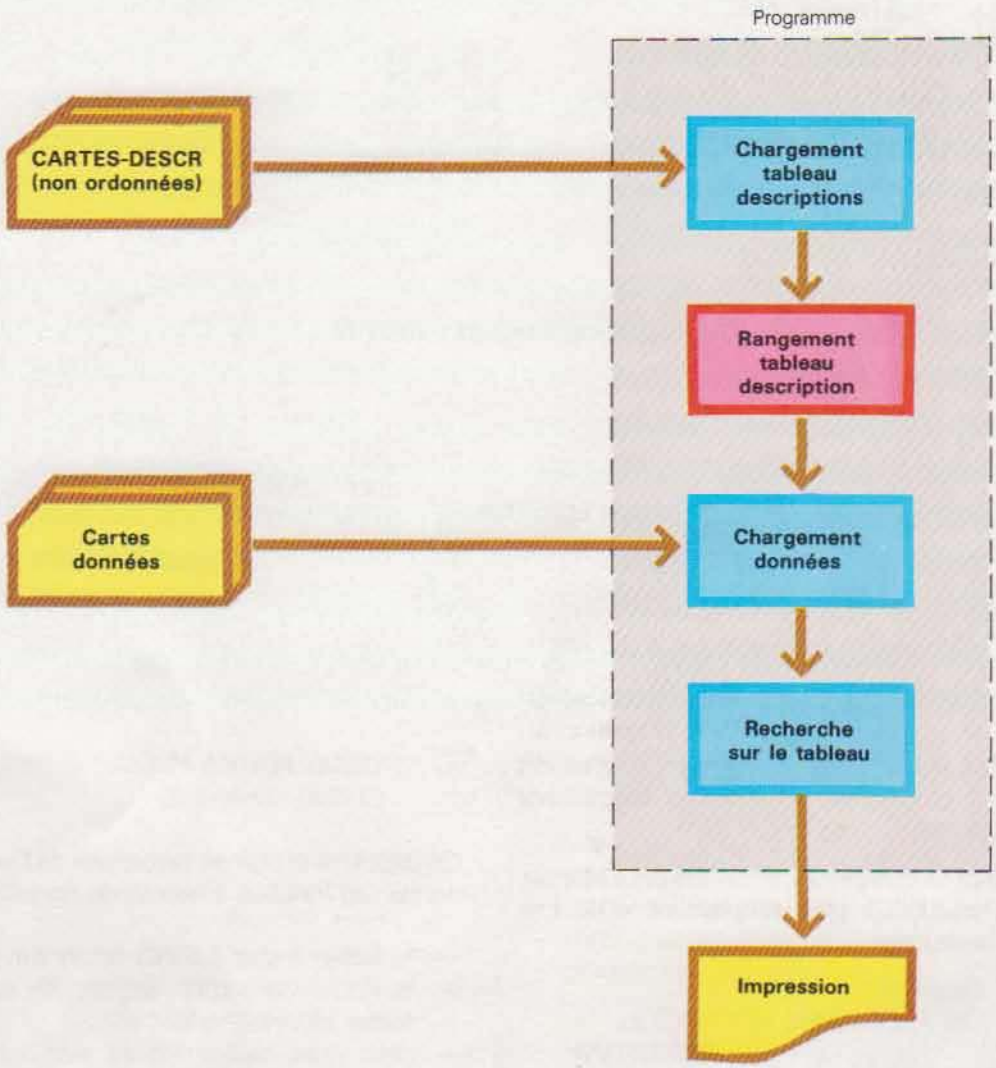
Dans chaque cadre sont reportés :

1 / le nom d'une localité française,
2 / le département d'appartenance,
3 / la région d'appartenance,
4 / le nombre d'habitants de la localité,
5 / la position (Nord ou Sud) dans laquelle on se situe par rapport à Paris.

Nous voulons sélectionner les cartes pour que :

- les régions soient ordonnées par nom ascendant,
- tous les départements de la même région soient ordonnés en série ascendante,

RECHERCHE DICHOTOMIQUE AVEC CLASSEMENT GERE PAR LE PROGRAMMEUR



EXEMPLE DE CLASSEMENT ET DE SELECTION DES DONNEES

```
    IDENTIFICATION DIVISION.
    .
    .
    .
    ENVIRONMENT DIVISION.
    .
    .
    FILE-CONTROL.
        SELECT FICHIER-DE-TRI  ASSIGN TO DISC.
        SELECT CARTES          ASSIGN TO CARD-READER.
        SELECT CARTES-ORD      ASSIGN TO DISC.
  *
  *
    DATA DIVISION.
    FILE SECTION.
    FD   CARTES
             LABEL RECORD OMITTED.
    01   SK                                PIC X(80).


    SD   FICHIER-DE-TRI.
    01   SK-TRI.
         05  LOCALITE                      PIC X(20).
         05  CHEF-LIEU                     PIC X(20).
         05  REGION                        PIC X(20).
         05  NOMBRE-HABIT                  PIC 9(8).
         05  POSITION                      PIC X(4).
         05  FILLER                        PIC X(8).
  *
  *
  *
     PROCEDURE DIVISION.
     SELECTIONNE.
         SORT FICHIER-DE-TRI
                           ON ASCENDING KEY REGION


                                                  CHEF-LIEU
                           ON DESCENDING KEY NOMBRE-HABIT
              USING  CARTES
              GIVING CARTES-ORD.
  *
  *
     FIN.
         STOP RUN.
```

- toutes les localités du même département soient ordonnées par nombre décroissant d'habitants.

La sélection complexe demandée peut être obtenue beaucoup plus simplement avec une seule instruction :

```
SORT fichier-tri
     ON ASCENDING KEY REGION
                          DEPARTEM.
     ON DESCENDING KEY NB-HABIT
     USING fichier-à-trier
     GIVING fichier-trié.
```

On peut tirer certaines indications de l'exemple donné : la fonction a besoin de connaître :

— le fichier à trier (USING fichier-à-trier),
— le fichier où seront rangées les données triées (GIVING fichier-trié),
— dans quel espace travail sera effectuée l'opération de tri (fichier-tri),

— les champs de l'enregistrement et l'ordre de sélection (ON ASCENDING KEY...) adoptés.

Analysons maintenant chacune de ces entités.

Fichier-tri. Cet espace peut être affecté tant à DISC qu'à TAPE en utilisant la clause normale SELECT dans l'ENVIRONMENT DIVISION.
Les particularités dans la définition du fichier de tri concernent essentiellement la DATA DIVISION où :

1 / l'indicateur de niveau SD (Sort Definition) **doit** être utilisé au lieu de FD,
2 / la clause LABEL RECORD **ne doit pas** être utilisée.

Ces deux exceptions précisées, toutes les autres clauses examinées dans la description des fichiers restent valables.
La description, au niveau SD, du fichier de tri ne difinit par un fichier physique. Mais il permet au compilateur de personnaliser les champs de l'enregistrement en fonction duquel s'opère la sélection. De ce fait, on ne décrira que les champs de l'enregistrement qui participent à la sélection. On obtient ainsi des traitements simplifiés.

Fichier à trier. C'est un fichier séquentiel quelconque utilisé en entrée. Il n'exige pas d'ouverture (OPEN) ou de fermeture (CLOSE) de la part du programmeur si la clause USING est utilisée.

Fichier trié. Egalement séquentiel, il est soumis aux opérations normales de définition de fichiers : il est entièrement géré en ouverture et en fermeture par le compilateur avec la clause GIVING.

Ordre de sélection. Chaque préposition ON rencontrée dans l'instruction SORT (tri) spécifie le type de classement à appliquer aux champs-clés classés.
La séquence, selon laquelle ces clés sont disposées à l'intérieur de l'instruction, détermine leur niveau hiérarchique.

Dans l'exemple rapporté, la clé REGION s'avère d'une importance supérieure à celle de DEPARTEMENT, elle-même prioritaire par rapport à NB-HABITANTS.
En guise de conclusion à cette première description du classement des données, nous donnons le programme détaillé qui sélectionne les cartes décrites selon les demandes spécifiques (voir listing ci-contre).

L'INPUT PROCEDURE et le verbe RELEASE

La fonction TRI nécessite généralement un temps de traitement très long. Aussi est-il de bonne règle de ne sélectionner que les enregistrements indispensables au calcul.
Pour permettre au programmeur de choisir ces enregistrements, le Cobol prévoit une procédure entrée (INPUT PROCEDURE) qui représente une alternative à la clause USING.
Dans cette SECTION, l'enregistrement lu par le fichier d'entrée est soumis à des traitements et à des contrôles avant d'être écarté au moment du tri. L'opération TRI consiste à écrire l'enregistrement dans le fichier-tri par l'intermédiaire de l'instruction RELEASE (tableau ci-dessous), analogue au WRITE.
Pour illustrer la fonction introduite, prenons l'exemple d'une sélection selon les critères connus des seules cartes correspondant aux localités de plus de 20 000 habitants.
Nous obtenons le programme de la page suivante, qui ne tient pas compte des parties communes à l'exemple précédent.

L'OUTPUT PROCEDURE et le verbe RETURN

En dehors de la clause GIVING, le Cobol permet de déclarer, au compilateur, le nom d'une SECTION (OUTPUT PROCEDURE) dans laquelle le programmeur soumet les enregistrements sélectionnés (directement lus par le fichier-tri) à n'importe quel type de traitement.
Comme pour la procédure d'entrée, le fichier-tri n'est accessible qu'avec l'instruction RETURN (voir format général page 1128), et non avec l'instruction READ.

FORMAT DE L'INSTRUCTION RELEASE

RELEASE enregistrement-tri [FROM nom-données]

EXEMPLE D'UTILISATION DE L'INSTRUCTION RELEASE

```
      .
      .
      WORKING-STORAGE SECTION.
      01  SK-WS.
          05  LOCALITE-WS                PIC X(20).
          05  CHEF-LIEU-WS               PIC X(20).
          05  REGION-WS                  PIC X(20).
          05  NOMBRE-HABIT-WS            PIC 9(8).
          05  POSITION-WS                PIC X(4).
          05  FILLER                     PIC X(8).
     **
     **
     **
       PROCEDURE DIVISION.
       SELECTIONNE SECTION.
       DEBUT.
            OPEN INPUT CARTES.
            SORT FICHIER-DE-TRI
                              ON ASCENDING KEY REGION
                                               CHEF-LIEU
                              ON DESCENDING KEY NOMBRE-HABIT
                 INPUT PROCEDURE ARRETEE
                 GIVING CARTES-ORD.
       FIN.
            STOP RUN.
       *
       *
       *
        FIN SECTION.
        LIRE-CARTES.
            READ CARTES INTO SK-WS
                        AT END
                        CLOSE CARTES
                        GO TO FIN-EX.
            IF  NOMBRE-HABIT-WS GREATER THAN 20000
                RELEASE SK-TRI FROM SK-WS.




            GO TO LIRE-CARTES.
       FIN-EX.
            EXIT.
```

FORMAT GENERAL DE L'INSTRUCTION RETURN

RETURN fichier-tri [INTO nom-données]
AT END phrase-impérative.

Reprenons le même exemple et supposons que l'on doive lister sur deux fichiers distincts toujours les villes situées respectivement au nord et au sud de Paris.
Le programme convenable se trouve en page 1130 et 1131.

Gestion des fichiers annexes

Tous les fichiers abordés jusqu'ici présentaient une structure de type séquentiel. Passons maintenant aux fichiers indexés. A la différence des premiers (où il est nécessaire de lire linéairement tous les enregistrements précédant l'information recherchée) il suffit, dans les fichiers indexés, d'indiquer la valeur recherchée dans le champ clé de l'enregistrement.
On obtient immédiatement l'enregistrement lui-même.
L'organisation indexée d'un fichier permet donc d'accéder directement à un enregistrement grâce à une clé symbolique.
Supposons par exemple que l'on doive rechercher des informations relatives à un salarié dont on connaît le code (matricule). Toutes les données sur le personnel se trouvent dans un fichier ordonné par matricule. Dans un fichier séquentiel, on aurait été obligé d'employer la procédure exposée dans l'organigramme de la page 1131 ; on lirait donc **tous** les enregistrements du fichier jusqu'au résultat (matricule) recherché. Avec cette méthode, il faudrait, en moyenne, un nombre d'accès égal à la moitié du volume d'enregistrement dans le fichier.
En revanche, dans un fichier indexé, dans lequel le matricule a été défini comme clé d'accès, il suffit d'introduire le code (matricule) dans la clé pour déclencher la lecture. L'enregistrement est immédiatement disponible, sans passer par une boucle de lecture (voir organigramme page 1132).
Cette dernière procédure est opérationnelle grâce à une organisation appropriée du fichier, schématiquement représentée page 1133.
Outre les données, ce fichier contient un index **(tableau des adresses)** où sont répertoriées toutes les clés, avec l'adresse de l'enregistrement correspondant.

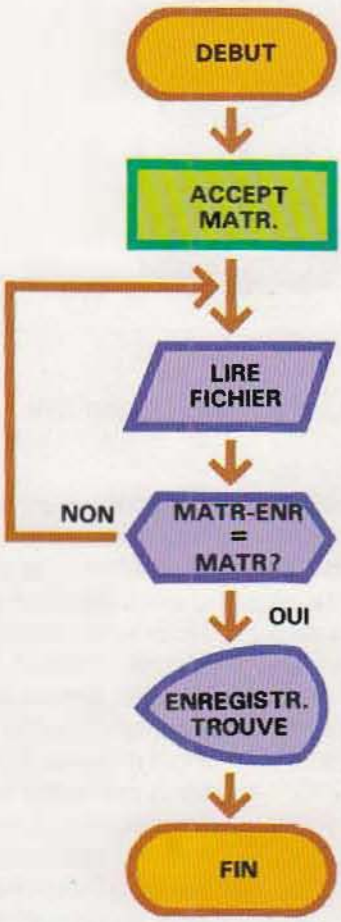
Opération de test et d'entretien d'une imprimante connectée à un gros ordinateur.

EXEMPLE D'UTILISATION DE LA COMMANDE SORT

```
  IDENTIFICATION DIVISION.
  .
  .
  ENVIRONMENT DIVISION.
  .
  .
  FILE-CONTROL.
    SELECT FIC-TRI       ASSIGN TO DISC.
    SELECT CARTE-ENTREE ASSIGN TO CARD-READER.
      SELECT CARTE-NORD   ASSIGN TO DISC.
      SELECT CARTE-SUD    ASSIGN TO DISC.
*
  DATA DIVISION.
    FILE SECTION.
*   ---- fichier d'entrée
    FD CARTE-ENTREE
         LABEL RECORD OMITTED.
  01 C-ENTREE              PIC X(80).
*  ---- premier fichier de sortie (villes du nord > 20000 hab.)
  FD CARTE-NORD
       LABEL RECORD STANDARD.
  01 C-NORD                PIC X(80).
*  ---- second fichier de sortie (villes du sud > 20000 hab.)
  FD CARTE-SUD
       LABEL RECORD STANDARD.
  01 C-SUD                 PIC X(80).
*  ---- fichier servant au tri
  SD FIC-TRI.
  01 C-TRI.
      05 LOCALITE          PIC X(20).
      05 CHEF-LIEU         PIC X(20).
      05 REGION            PIC X(20).
      05 NB-HABIT          PIC 9(8).
      05 POSIT-GEO-WS      PIC X(4).
      05 FILLER            PIC X(8).
*
*
   WORKING-STORAGE SECTION.
   01 VILLE-WS.
      05 LOCALITE-WS       PIC X(20).
      05 CHEF-LIEU-WS      PIC X(20).
      05 REGION-WS         PIC X(20).
      05 NB-HABIT-WS       PIC 9(8).
      05 POSIT-GEO         PIC X(4).
      05 FILLER            PIC X(8).
*
*
*
  PROCEDURE DIVISION.
  PRINCIPALE SECTION.
    DEBUT-PRINCIPALE.
      OPEN INPUT CARTE-ENTREE.
      SORT FIC-TRI
          ON ASCENDING KEY  REGION
                            CHEF-LIEU
          ON DESCENDING KEY NB-HABIT
      INPUT PROCEDURE  ENTREE-TRI
      OUTPUT PROCEDURE SORTIE-TRI.
    FIN-PRINCIPALE.
      STOP RUN.
*
*           sections d'entrée / sortie du tri
*
   ENTREE-TRI SECTION.
     LECTURE-CARTE.
       READ CARTE-ENTREE INTO VILLE-WS
            AT END
               CLOSE CARTE-ENTREE
               GO TO LECTURE-FIN.
```

```
       IF NB-HABIT-WS GREATER THAN 20000
           RELEASE C-TRI FROM VILLE-WS.
       GO TO LECTURE-CARTE.
     LECTURE-FIN.
       EXIT.
*
   SORTIE-TRI SECTION.
     OUV-FIC-SORTIE.
       OPEN OUTPUT CARTE-NORD
                   CARTE-SUD.
     SEPARE-GEO.
       RETURN FIC-TRI INTO VILLE-WS
           AT END
              CLOSE CARTE-NORD
              CLOSE CARTE-SUD
              GO TO SEPARE-FIN.
       IF POSIT-GEO-WS = 'NORD' WRITE CARTE-NORD FROM VILLE-WS.
       IF POSIT-GEO-WS = 'SUD'  WRITE CARTE-SUD  FROM VILLE-WS.
       GO TO SEPARE-GEO.
     SEPARE-FIN.
       EXIT.
```

EXEMPLE DE RECHERCHE DANS UN FICHIER SEQUENTIEL



EXEMPLE DE RECHERCHE DANS UN FICHIER INDEXE

Lorsqu'on demande une lecture d'enregistrement par l'intermédiaire d'une clé, le programme lance une ou plusieurs lectures dans l'index jusqu'à ce qu'il ait trouvé la clé recherchée.
Dans un deuxième temps, en se servant de l'adresse définie dans l'index, il accède directement à l'enregistrement. Donc, pour lire un enregistrement, il faut non pas une, mais deux lectures, quelle que soit la position de cet enregistrement.
C'est une procédure particulièrement avantageuse quand il s'agit d'une recherche aléatoire dans un fichier d'assez grande taille.
Il existe 3 méthodes d'accès à un fichier indexé :

1/ aléatoire (Random)
2/ séquentielle (Sequential)
3/ mixte ou dynamique (Dynamic)

En mode séquentiel, le fichier obéit aux règles habituelles du traitement des fichiers séquentiels : les données sont traitées en séquence physique du premier au dernier.
Nous avons déjà indiqué les caractéristiques de l'accès aléatoire (Random). Quant au mode d'accès dynamique (Dynamic) il s'agit d'une synthèse des deux autres : il permet un traitement mixte de fichier. On peut, en effet, adopter la procédure qui semble la plus appropriée aux besoins du moment : par exemple lire de manière séquentielle une série de données, passer à l'aléatoire pour une autre série et revenir au séquentiel, etc.

REPRESENTATION SIMPLIFIEE DE LA STRUCTURE D'UN FICHIER INDEXE

Index

Clé (matricule	Adresse de l'enregistrement
00015	000001
00038	000002
00088	000003
00133	000004
00134	000005

Enregistrement des données

Matricule	Nom et prénom
00015	DURAND PAUL
00038	DUPONT JEAN
00088	MERCIER JACQUES
00133	PERRIN NATHALIE
00134	LOISEAU CHARLES

C'est pourquoi on parle parfois de fichiers **séquentiel-indexé** ou **IS** (pour Indexed-Sequential).

Voyons maintenant les clauses servant à définir un fichier IS dans les divisions environnement et données, ainsi que les instructions de la division algorithmes (Procédure Division) nécessaires à sa gestion.

Description des fichiers IS dans ENVIRONNEMENT DIVISION

Si un programme utilise un fichier IS au niveau Select, il faut nécessairement assigner le fichier à Disc et en préciser la clause :

ORGANIZATION IS INDEXED

Elle doit être suivie d'une deuxième clause qui définit le mode d'accès :

ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC }

Il faut enfin écrire la clause :

RECORD KEY IS nom-donnée

où « nom-donnée » désigne un champ de l'enregistrement défini dans le Fichier-descriptions (qui contient la clé symbolique de l'enregistrement). Ce champ, élémentaire ou composé, ne dépasse généralement pas 60 caractères. Le champ-clé ne peut contenir une valeur équivalente à LOW-VALUES et doit être déclaré avant toute opération d'accès aléatoire.

Description des fichiers IS dans DATA DIVISION

Dans le Fichier-descriptions, on emploie la clause

LABEL RECORD STANDARD

d'après les règles précédemment établies pour les fichiers séquentiels assignés à DISC.
Toujours au niveau FD, dans la description de l'enregistrement, il faut définir le champ contenant la clé de l'enregistrement au moyen du nom indiqué dans la clause RECORD KEY (au niveau SELECT).

Les instructions dans PROCEDURE DIVISION

Un fichier indexé est ouvert selon l'une des trois méthodes étudiées pour les fichiers à accès séquentiel, à l'exception de la modalité EXTEND.
Le format de l'instruction d'ouverture est donc :

FORMAT DE L'INSTRUCTION OPEN

OPEN { INPUT / OUTPUT / I/O } nom-fichier

Le verbe READ. L'instruction de lecture n'est possible qu'à deux conditions : le fichier a préalablement été ouvert en entrée (INPUT) ou en entrée-sortie (I/O) ; le fichier comporte deux formats en fonction du mode d'accès défini au

1er FORMAT DE L'INSTRUCTION READ APPLIQUEE AUX FICHIERS IS

READ nom-fichier [NEXT] RECORD [INTO nom-donnée-1]

AT END phrase-impérative.

2e FORMAT DE L'INSTRUCTION READ APPLIQUEE AUX FICHIERS IS

READ nom-fichier [INTO nom-donnée-2]

INVALID KEY phrase-impérative.

niveau SELECT (voir page 1134).
Plus précisément, les règles à respecter sont :

1/ Pour tout fichier à accès séquentiel (ACCESS MODE IS SEQUENTIAL), employer le 1er format sans la clause NEXT :

READ nom-fichier RECORD
[INTO nom-donnée-1]
AT END phrase-impérative.

2/ Pour des lectures de type séquentiel sur des fichiers à accès mixte (ACCESS MODE IS DYNAMIC), c'est le 1er format, avec la clause NEXT, que l'on doit employer :

READ nom-fichier NEXT RECORD
[INTO nom-donnée-1]
AT END phrase-impérative.

3/ Pour toutes les opérations de lecture aléatoire, employer le deuxième format lorsqu'au niveau SELECT on a déclaré ACCESS MODE IS RANDOM ou ACCESS MODE IS DYNAMIC.

4/ Avant l'opération de lecture, on assigne, au champ de RECORD KEY, une valeur correspondante à celle de l'enregistrement recherché. S'il n'existe pas, on exécutera les instructions de la « phrase-impérative » qui suit la clause INVALID KEY.

Le verbe WRITE. Cette instruction n'est opérationnelle que sur des fichiers préalablement ouverts en sortie ou en E/S et selon le format ci-dessous.
Si le fichier est ouvert en entrée (phase de création), les enregistrements doivent être écrits en ordre croissant de clé. Autrement dit, on accède au fichier de manière séquentielle, quel que soit le mode d'accès choisi.
Avant toute écriture, il faut organiser le champ du domaine contenant la clé d'enregistrement.
La phrase impérative, spécifiée dans la clause INVALID KEY, est exécutée dans les cas suivants :

- les valeurs des clés ne sont pas en ordre croissant d'une clause WRITE à la suivante
- la valeur de la clé n'est pas unique dans le cadre du fichier
- on a rempli tout l'espace alloué sur le disque.

Instructions spécifiques aux fichiers IS
Trois verbes n'ont pas d'équivalents dans les

FORMAT DE L'INSTRUCTION WRITE APPLIQUEE AUX FICHIERS IS

WRITE nom-enregistrement [FROM nom-donnée]

INVALID KEY phrase-impérative.

fichiers séquentiels :

REWRITE pour mettre à jour des données existantes

DELETE pour écraser des données déjà présentes sur le fichier

START pour pointer un enregistrement donné.

Le verbe REWRITE. L'opération de réécriture n'est autorisée que sur des fichiers ouverts en ES, selon le premier format indiqué en bas de page.

Son emploi est subordonné au respect de quelques règles fondamentales :

— L'instruction n'est exécutable que si la modification d'une donnée n'entraîne pas celle de la clé d'accès.
Sinon, la phrase impérative, spécifiée dans la clause INVALID KEY, serait exécutée.

— En fichiers séquentiels, REWRITE n'est lancé qu'après un READ fructueux.

— En fichiers à accès aléatoire ou dynamique, il n'est pas nécessaire d'exécuter un READ avant un REWRITE.

— La phrase impérative de la clause INVALID KEY s'exécute seulement si la valeur de la clé, attribuée dans RECORD KEY (avant REWRITE), est différente de celle des autres enregistrements du fichier.
Il n'est donc pas possible de réécrire un enregistrement non encore créé.

Le verbe DELETE. Comme pour REWRITE, l'instruction d'effacement (voir le second format ci-dessous) n'opère que sur des enregistrements déjà existants.

Elle suit les règles suivantes :

— Le fichier doit être ouvert en ES.

— En fichiers séquentiels, l'instruction READ doit précéder l'effacement.

— En fichiers à accès aléatoire ou dynamique, il faut réorganiser RECORD KEY avant l'opération d'effacement (...).

— Si la clé définie ne correspond à aucun enregistrement, c'est la phrase impérative de la clause INVALID KEY qui s'exécute.

Phase de test de terminaux dans une unité de production.

FORMAT DE L'INSTRUCTION REWRITE

REWRITE nom-enregistrement [FROM nom-donnée]
INVALID KEY phrase impérative.

FORMAT DE L'INSTRUCTION DELETE

DELETE nom-fichier RECORD
INVALIDE KEY phrase-impérative.

Le verbe START. Il initialise la lecture séquentielle en pointant sur un enregistrement du fichier.
Supposons que l'on ait à imprimer tous les enregistrements du fichier du personnel défini plus haut, de telle sorte que le code matricule, donc la clé d'enregistrement (RECORD KEY), soit supérieur à une valeur donnée.
Grâce à l'instruction START, on peut se positionner directement sur le premier matricule valable.
On peut aussi sortir tous les enregistrements suivants pour une lecture séquentielle du fichier.
La comparaison entre deux organigrammes, l'un avec un test et l'autre avec START, montre que cette dernière instruction ne rend pas effectivement disponible l'enregistrement mais qu'elle réalise un pointage à la position voulue.

L'instruction START (voir format page 1137) doit être suivi d'un :

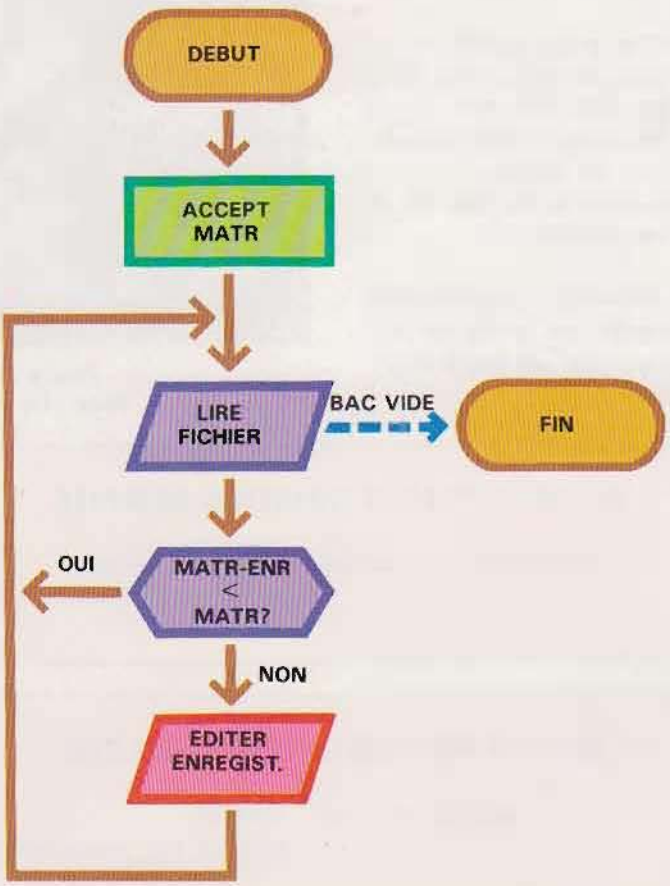
READ nom-fichier AT END...

ou d'un :

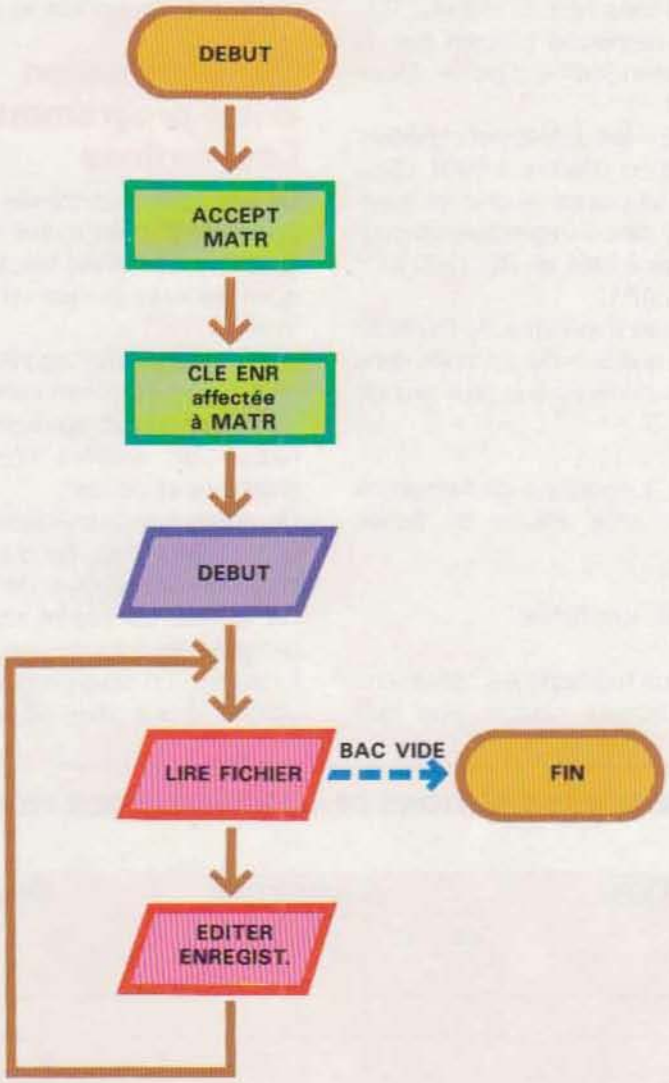
READ nom-fichier NEXT AT END...

selon le type de déclaration du fichier, respectivement en fichier séquentiel ou dynamique.
L'opération de positionnement n'est réalisable que sur des fichiers ouverts en entrée ou en E/S et après avoir défini l'accès séquentiel ou dynamique.
Le format général de l'instruction se trouve en bas de la page ci-contre.
Le « nom-donnée » désigne la clé d'enregistrement auquel il faut attribuer une valeur avant

POSITIONNEMENT SUR UN ENREGISTREMENT DU FICHIER IS SANS START



POSITIONNEMENT SUR UN ENREGISTREMENT DU FICHIER IS AVEC START



FORMAT GENERAL DE L'INSTRUCTION START

START nom-fichier KEY IS { EQUAL TO / = / GREATER THAN / > / NOT LESS THAN / NOT < } nom-donnée

INVALIDE KEY phrase-impérative.

l'exécution de START. En cas d'omission de la clause KEY, le compilateur supposera que l'enregistrement recherché possède la même clé que RECORD KEY. On reviendrait alors à la spécification de la forme KEY IS EQUAL TO.
Si l'enregistrement recherché n'existe pas, la phrase spécifiée est exécutée dans la clause INVALID KEY.
Lorsque la clause KEY est définie par supérieur à (GREATER THAN) ou inférieur à (NOT LESS THAN), le système va pointer le premier enregistrement de valeur de clé respectivement supérieure ou inférieure à celle de RECORD KEY avant l'opération START.
Dans ce cas, la phrase impérative de INVALID KEY n'est exécutée que si la clé attribuée dans RECORD KEY est supérieure à la plus grande clé du fichier.

Le verbe CLOSE. L'opération de fermeture d'un fichier IS est celle utilisée en fichier séquentiel :

CLOSE nom-fichier

Le tableau ci-dessous regroupe les opérations possibles sur des fichiers indexés ainsi que leurs conditions d'ouverture et d'accès.
Le tableau ci-contre dresse la liste des causes de sortie de fichier par INVALID KEY confrontées aux opérations exécutées en fonction des modalités d'ouverture et d'accès.

Communication entre programmes. Les routines

Un programme en format exécutable peut être composé d'un ensemble de sous-programmes (anglais subroutines) liés, logiquement et physiquement entre eux par un programme principal (main).
Avantage de cette organisation : chaque sous-programme peut être compilé et testé de manière tout à fait autonome. Cela permet de réduire de manière considérable le temps d'écriture et de test.
Un autre avantage, résultant de l'emploi des sous-programmes, est que l'on peut également créer une bibliothèque de fonctions simples ou complexes, qui seront réutilisées par d'autres programmes.
Exemple : un sous-programme qui contrôle la validité d'une date (CONTRDATE), ou qui

RESUME DES OPERATIONS REALISEES SUR DES FICHIERS IS

OPERATION	OUVERTURE			MODE D'ACCES		
	ENTREE	SORTIE	E/S	SEQ	DYN	ALE
OPEN INPUT ...	*			*	*	*
OPEN OUTPUT ...		*		*		
OPEN I-O ...			*	*	*	*
READ ... AT END ...	*		*	*		
READ ... NEXT AT END ...	*		*		*	
READ ... INVALIDE KEY ...	*		*			*
WRITE ... INVALIDE KEY ...		*		*		
WRITE ... INVALIDE KEY ...			*	*	*	*
REWRITE ... INVALIDE KEY ...			*	*	*	*
DELETE ...			*	*		
DELETE ... INVALIDE KEY ...			*		*	*
START ... INVALIDE KEY ...	*		*	*	*	

SEQ = SEQUENTIEL, DYN = DYNAMIQUE, ALE = ALEATOIRE

LISTE DES CAUSES D'INVALIDATION DES CLES DANS UN FICHIER IS

OPERATION	OUVERTURE	MODE D'ACCES	CLE INVALIDEE
READ	E-E/S	ALE	Pas de donnée identique à celle définie.
WRITE	S	SEQ	Manque espace sur disque.
WRITE	E/S	—	Donnée avec même clef déjà existante. Manque espace sur disque.
REWRITE	E/S	SEQ	Clef correspondant à celle de la donnée précédemment lue.
REWRITE	E/S	DYN/ALE	Il n'existe pas de donnée avec une clef égale à celle définie.
DELETE	E/S	DYN/ALE	Il n'existe pas de donnée avec une clef égale à celle définie.
START	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef égale à celle définie.
START ... KEY EQUAL ...	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef égale à celle définie.
START ... KEY GREATER ...	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef supérieure à celle définie.
START ... KEY NOT LESS ...	E/S	SEQ/DYN	Il n'existe pas de donnée avec une clef supérieure ou égale à celle définie.

SEQ = SEQUENTIEL DYN = DYNAMIQUE ALE = ALEATOIRE
E = ENTREE (Input) S = SORTIE (Output) E/S = ENTREE/SORTIE (I/O)

donne le jour de semaine d'une date (calendrier).
Ces deux sous-programmes (CONTRDATE ET CALENDRIER) sont étudiés en détail plus loin (voir listings des pages 1148 à 1151).
Rappelons que le programme principal et les sous-programmes sont reliés physiquement pendant l'opération de connexion (linking). De ce fait, l'occupation en mémoire réelle est bien égale à la somme des programmes.
Lorsqu'un programme, principal ou routine, appelle un sous-programme pour exécuter une fonction, il y a en général transfert des données à traiter.
Prenons un exemple.
Dans le cas du sous-programme de contrôle de date, le programme « appelant » doit fournir la date pour permettre au programme appelé de lancer ses opérations de vérification. Il transmettra ensuite, au programme principal, le résultat du contrôle effectué.

Cet échange d'information est obtenu en utilisant des champs de mémoire, identifiés par des noms symboliques décrits au niveau 01 ou 77 de la WORKING-STORAGE SECTION du programme d'appel.
Les champs correspondants du sous-programme sont définis, en respectant les règles, dans une section spéciale de la division données. Il s'agit de la LINKAGE SECTION, décrite à la fin de la WORKING-STORAGE SECTION.

Dans la LINKAGE SECTION, la clause VALUE n'est admise que si elle correspond aux niveaux 88.

Il n'est pas nécessaire que les champs de passage soient identiques dans le programme appelant et dans le programme appelé, mais ils doivent globalement avoir les mêmes PICTURE et les mêmes USAGE.

Les prévisions de l'ordinateur

Voilà plus de dix ans déjà que les météorologues emploient des satellites. Incontestablement, les données transmises au sol ont beaucoup contribué à faire évoluer la météorologie comme science théorico-expérimentale. Et surtout, elles ont beaucoup aidé à la formulation de prévisions plus fiables et à un degré de finesse insoupçonné il y a quelques années.

Ces résultats sont indéniablement dûs au rôle joué par les satellites. Mais aussi, et en grande partie, aux ordinateurs et aux techniques modernes de traitement de l'information.

Prenons un exemple européen, celui du nouveau centre italien pour l'acquisition et le traitement des données recueillies par les ***satellites météorologiques.*** *Il fait partie du Centre de Météorologie nationale et de climatologie de l'aviation (CNMCA) dépendant de l'Armée de l'air. Sa mission est de traiter les informations en provenance du satellite géostationnaire européen Meteosat.*

Le réseau national de saisie et de traitement des données est composé d'une station réceptrice primaire, de cinq stations secondaires, d'un site informatique et de vingt stations d'enregistrement automatiques. La station primaire (PDUS pour Primary Data User Station) reçoit les données numériques transmises par Meteosat ; les cinq autres stations, dites secondaires (SDUS pour Secondary Data User Station), réceptionnent les informations analogiques.

En effet, le satellite envoie vers la terre deux types de signaux : analogique et numérique. Le premier est une entité qui varie de manière pratiquement continue en se maintenant rigoureusement proportionnel à la grandeur mesurée ; le signal numérique, lui, est une série d'impulsions de type allumé/éteint comme celles traitées par les ordinateurs. Dans le second cas, la proportionnalité avec la grandeur observée se traduit en termes numériques.

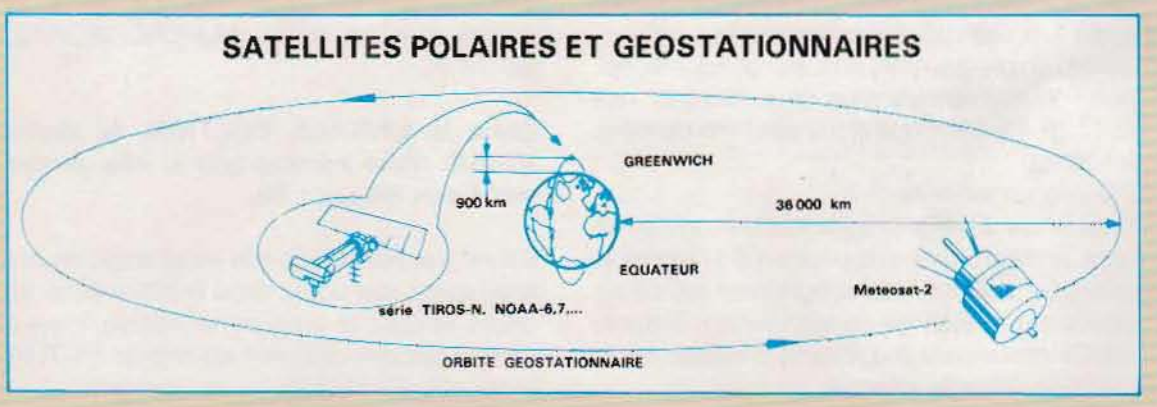
*La variable observée par Meteosat est la « **radiance** » de la surface terrestre. C'est la quantité d'énergie irradiée vers l'espace par la croûte terrestre, par les surfaces liquides et par le système nuageux. Le sous-système de calcul en mode autonome (off-line) traite localement les données contenues dans les signaux numériques ou analogiques les traduit en une représentation graphique.*

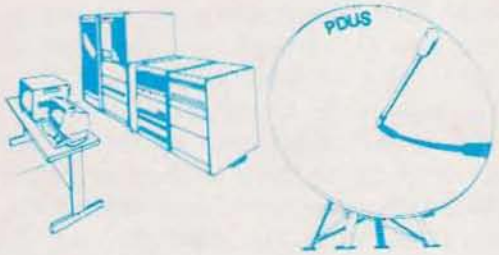
Les vingt stations automatiques de mesure des données de type DCP (Data Collection Platform) utilisent le ***Meteosat*** *comme plateforme de retransmission des données vers le* ***Réseau mondial des télécommunications météorologiques.***

La PDUS est composée d'un sous-système de réception, équipé d'une antenne parabolique d'environ 5 mètres de diamètre, d'un sous-système de calcul, d'équipements sophistiqués pour la représentation graphique et alphanumérique des images et des données.

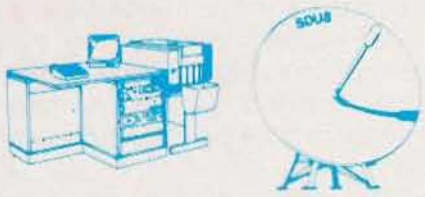
L'antenne reçoit, directement de Meteosat 2, des images à haute résolution. Après un prétraitement, les signaux sont renvoyés de Darmstadt (où se trouve le Centre opérationnel qui gère le satellite) à Meteosat.

Le cœur de la station primaire PDUS est composé d'un ordinateur de 2 Mo de mémoire RAM avec une mémoire de masse (disques magnétiques) d'environ 250 Mo. Pour la sauvegarde, deux unités de bandes magnétiques permettent la conservation, provisoire et historique, des données alors que la gestion opérationnelle des procédures de traitement des données est réalisée sur terminaux interactifs.

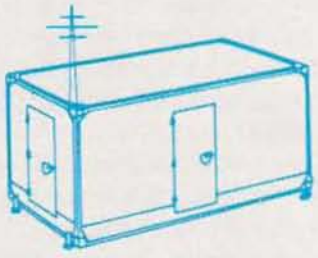




PDUS
PRIMARY DATA USER STATION



SDUS
SECONDARY DATA USER STATION



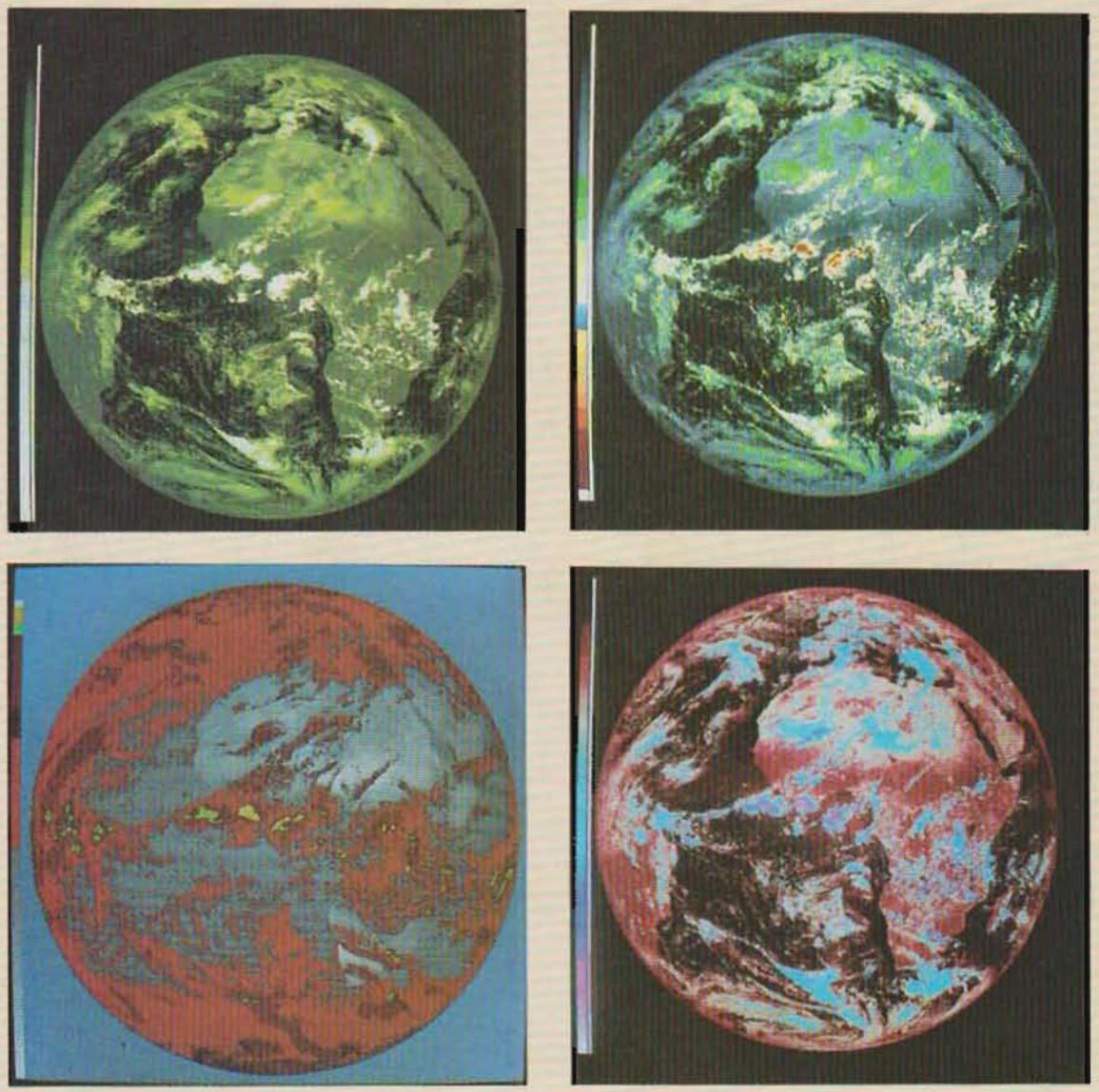
DCP
DATA COLLECTION PLATFORM

Représentation schématique de sous-systèmes de saisie et de traitement de données transmises par le satellite Meteosat.
En haut : station de travail du système ISIDE.
Au milieu : poste de travail dans une station réceptrice secondaire (SDUS) ; en bas, l'ordinateur central d'une station réceptrice primaire PDUS.

*Reliés à l'ordinateur, des appareils de photoreproduction numérique et graphique permettent également l'**impression de détails** d'une image (zoom) présentant un intérêt météorologique particulier.*

Un réseau de micro-ordinateurs (travaillant parallèlement à l'UC), d'interfaces et d'écrans graphiques permet de conserver en continu jusqu'à 32 images résolution pour chacune des trois bandes (visible, infrarouge et vapeur

d'eau) sur lesquelles le radiomètre de Meteosat travaille.
Les résultats peuvent également être des films élaborés par un système d'animation complexe reposant sur une structure de multiprocesseurs et sur une mémoire de 2 Mo.
Ainsi, les masses nuageuses et les perturbations défilent-elles sous les yeux du météorologue, reconstituant un historique du mouvement et de l'évolution sur une période de 16 heures (et plus) entre deux images, avec des intervalles de 30 minutes. De tels films sont également réalisables sur bande ou sur vidéocassette et reproductibles à loisir.
On peut employer le zoom interactif et la technique de la **fausse couleur**, fondée sur l'attribution d'une couleur à des valeurs données de la radiance, mesurée par Meteosat. Cela permet d'affiner sensiblement les possibilités d'interprétation physique des phénomènes atmosphériques en cours. La couleur est également utilisée en amont d'un traitement des données de **radiance** grâce à des techniques **multispectrales**. En effet, chacune des 32 images visualisées se compose d'un nombre considérable d'éléments d'image (pixels) : 1250 lignes × 2500 pixels par ligne dans la bande de la lumière visible et 650 × 1250 pour l'infrarouge et la vapeur d'eau. Le système mémorise la valeur numérique de la radiance correspondante à chaque pixel. Le traitement des données numériques, recueillies par la station primaire, fournit des mesures météorologiques très utiles à la prévision du temps.
L'une d'elles est la mesure de la vitesse des vents en altitude, déduite du déplacement des nuages et mesurable à partir de l'observation d'images successives d'une même zone. L'information est très importante puisqu'elle comble une lacune des observations traditionnelles en altitude au moyen de radio-sondes au-dessus des zones désertiques et des océans. La direction et la vitesse des vents, tirées des données de Meteosat, sont introduites tous les jours pour construire des **modèles numériques de prévision** du temps à court et à moyen terme. Une deuxième série importante d'informations concerne la mesure des températures à la surface des mers. Elle est obtenue par un traitement des données de la radiance dans la bande de l'infra-rouge (qui est proportionnelle aux températures du corps radiant selon le principe de Stefan-Boltzmann). Il faut également tenir compte de certaines corrections dûes à l'atténuation de la radiation quand elle traverse l'atmosphère et de l'angle d'inclinaison de la lumière solaire.
Ces valeurs de température de surface des océans sont couramment employées dans les modèles prévisionnels et pour les **bilans de radiation** du système Terre-Atmosphère.
L'analyse des nuages détermine, sur d'énormes surfaces, le pourcentage de ciel couvert et les différents types de nuages formant la nébulosité globale. Effectuée toutes les 6 heures, elle est directement exploitable pour l'assistance à la navigation aérienne. Des techniques statistiques permettent de classer les divers types de nuages et de fournir, par temps clair, des informations très utiles sur les courants marins.
L'altitude du plafond des nuages est une autre donnée que l'on peut extrapoler de l'analyse des radiances pour des zones importantes de l'Europe, de la Méditerranée et de l'Afrique. La température du plafond des corps nuageux étant inversement proportionnelle à leur altitude, on peut appliquer une couleur à des intervalles de température et afficher sur écran de télévision une carte représentant la surface externe des systèmes nuageux. Il s'agit bien d'une information précieuse pour l'assistance à la navigation aérienne, civile et militaire. Les données les plus intéressantes pour la climatologie sont principalement les **cartes de couverture du ciel**, de l'enneigement et des valeurs de radiation pour les océans et les déserts. Avec ces informations, on peut calculer la différence entre le total d'énergie qui entre et qui sort du système Terre-Atmosphère. Naturellement, ce « solde » détermine la température moyenne de la terre et conditionne le climat à l'échelle de la planète.
Les stations secondaires SDUS sont installées au CNMCA de Rome, dans les aéroports de Milan, de Cagliari, de Brindisi et de Palerme. Elles reçoivent les images analogiques des transmissions WEFAX (Weather Facsimile) de Meteosat dans les trois bandes de la lumière visible, de l'infra-rouge et de la vapeur d'eau, selon des formats et un programme définis par le centre principal de Darmstadt, en Allemagne Fédérale.
Les images sont composées de 800 lignes ×



Images en fausses couleurs fournies par une station PDUS.

800 pixels et couramment utilisées pour l'analyse du temps sur 24 heures. Contrairement à celles reçues par PDUS, ces images sont destinées à des usages de type non numérique.

Chaque station est munie d'un module récepteur (antenne, down-converter, récepteur) qui capte les signaux analogiques envoyés sur deux canaux de télécommunication (dans la bande S à 1691 et 1694,5 MHz) par un micro-ordinateur qui règle le flux des informations et qui accepte un ***dialogue interactif*** *avec l'utilisateur. Elle dispose également d'un système de visualisation avec des écrans en noir et blanc, d'un système d'enregistrement sur bande magnétique et d'un système de reproduction sur papier photographique (un reproducteur photo à rayon laser).*

Pourtant, le système de contrôle et de commande de la station est très simple. Ainsi, grâce à des ***procédures élémentaires de dialogue****, l'opérateur peut programmer le fonctionnement automatique de la station entière. Les signaux, captés par chacune des cinq SDUS, sont envoyés, par téléphone, à des terminaux de type facsimile ou vidéo (un maximum de dix par station). Ainsi, 50 organismes, répartis sur le territoire italien reçoivent-ils, en temps réel, les images Meteosat.*

Image, fournie par une SDUS, d'un système nuageux au-dessus de l'Europe.

La possibilité d'enregistrement des images sur bande magnétique (cassette) constitue un excellent moyen de créer un système d'archives locales, peu encombrant et relativement bon marché. Le système de traitement en mode local (off-line) a d'abord été utilisé par le service météorologique de l'aviation militaire, pour étudier les caractéristiques des images de Meteosat et développer des programmes de traitement des informations numériques en provenance du satellite. Ces programmes, après mise au point, sont progressivement intégrés à la PDUS.

Le système traite les images reçues presque en temps réel par la station primaire, grâce à une liaison à grande vitesse et à un accès direct à la mémoire des deux ordinateurs. Le système, qui s'appelle ISIDE (Interactive System for Image Data Elaboration) se compose d'un ordinateur analogue à celui de la PDUS, avec une mémoire centrale d'une capacité de 1 Mo et une mémoire de masse sur disques de 20 Mo. Il est complété par deux unités de bandes magnétiques, un terminal de contrôle, une table traçante et un écran graphique couleur.

Naturellement, le système ISIDE fournit des images en provenance de Meteosat, mais aussi d'autres satellites. Il s'agit d'images radar numériques ou de cartes météorologiques que l'on peut superposer à d'autres images visualisées sur écran.

Chacune des 20 stations automatiques DCP est munie de 8 capteurs qui contrôlent et mesurent les variables fondamentales au sol : intensité du vent, pression, température de l'air, humidité relative, précipitations, température au sol, radiation solaire.

Les stations, entièrement automatisées, déterminent les valeurs des paramètres reçus chaque heure, convertissent les signaux électriques des capteurs en paramètres physiques, au moyen de tableaux de correspondance et codifient les informations en leur attribuant le format adéquat.

Les messages sont enregistrés au niveau local et transmis à Meteosat. Le satellite renvoie le message de chaque DCP au centre opérationnel de Darmstadt qui le rediffuse sur le réseau mondial de télécommunications météorologiques. Les stations sont également reliées au CNMCA de Rome au moyen de lignes téléphoniques commutées qui permettent la transmission du message en cas de panne du satellite.

Le nouveau centre du Service météorologique pour le traitement de données provenant du satellite sera complété par des équipements destinés à la réception de données numériques ou sous forme d'images envoyées à Terre par des satellites polaires de la série TIROS-N.

Munis d'une résolution spatiale de l'ordre de 1,2 km (environ quatre fois celle de Meteosat), ces derniers fournissent des images de qualité supérieure. De plus, disposant d'un nombre élevé de capteurs radiométriques dans la bande de l'infra-rouge et dans celle des micro-ondes, ils sont en mesure d'effectuer des sondages sur les profils verticaux de température, avec une résolution spatiale d'environ 50 km.

Le Centre que nous venons de décrire est le résultat de la collaboration entre la DATAMAT, une entreprise italienne d'ingénierie des systèmes qui a fourni la station primaire PDUS, les stations secondaires SDUS et le système autonome, et la SIAP, une autre entreprise italienne d'équipements de précision qui a construit les stations automatiques DCP. Le Service météorologique de l'Armée de l'air d'Italie emploie le système de traitement des informations provenant du satellite en mode continu (24 heures sur 24). Les premiers résultats montrent que les objectifs fixés seront bientôt atteints.

D'après le Service Météorologique de l'Armée de l'air et de la Datamat Ingegnieria dei Sistemi S.p.A., en Italie.

En d'autres termes, le champ d'échange du programme principal et son équivalent du sous-programme peuvent avoir des descriptions différentes, à condition que la longueur globale et le type d'USAGE employé restent inchangés. Si, dans le programme appelant, on définit ainsi un champ :

```
01 A.
   05 A1        PIC X(2).
   05 A2        PIC X(4).
```

dans la LINKAGE SECTION du programme appelé, le même champ est décrit :

```
01 B
   05 B1        PIC X.
   05 B2        PIC X(3).
   05 B3        PIC X(2).
```

La définition des champs, dans la LINKAGE SECTION, n'implique pas d'allocation de la mémoire correspondante. En effet, le sous-programme prévoit que les champs seront déjà physiquement définis dans la mémoire du programme appelant. Le champs B de l'exemple précédent, défini dans la LINKAGE SECTION du programme appelé, occupe physiquement la même zone mémoire que le champ A défini dans la WORKING-STORAGE SECTION du programme appelant.

En résumé, le champ de LINKAGE SECTION d'un sous-programme peut être comparé à une REDEFINES du champ de WORKING-STORAGE SECTION du programme appelant.

Le verbe CALL

L'appel d'un sous-programme est réalisé par l'instruction CALL (premier format ci-dessous).

« Literal » représente une chaîne de caractères, entre guillemets, correspondant au nom défini dans la clause PROGRAM-ID du programme appelé ; « nom-donnée-1, nom-donnée-2... » ; ils désignent les champs de passage définis dans la WORKING-STORAGE SECTION.
Le nom du programme appelé est nécessairement unique : un programme ne peut appeler deux sous-programmes différents avec la même clause PROGRAM-ID.

PROCEDURE DIVISION d'un sous-programme

Pour compléter la description des conditions d'emploi d'un sous-programme, il faut se rappeler, qu'outre la LINKAGE SECTION, il est indispensable d'introduire la clause USING dans l'instruction PROCEDURE DIVISION selon le second format ci-dessous.
Les « nom-donnée-3, nom-donnée-4... » désignent des champs de passage définis dans la LINKAGE SECTION. L'ordre dans lequel ces noms sont définis doit refléter fidèlement l'ordre de définition des données dans la clause USING du CALL correspondant.
L'association entre les champs de passage du programme appelant et ceux définis dans la LINKAGE SECTION du programme appelé n'est en effet accomplie qu'au moyen de cette correspondance.

L'instruction EXIT PROGRAM

Cette instruction transfère le contrôle du sous-programme au programme appelant.
Elle a le format suivant :

```
EXIT PROGRAM
```

Un sous-programme peut comporter plusieurs

FORMAT DE L'INSTRUCTION CALL

```
CALL literal USING nom-donnée-1 nom-donnée-2 ...
```

PROCEDURE DIVISION D'UN SOUS-PROGRAMME

```
PROCEDURE DIVISION USING nom-donnée-3 nom-donnée-4 ...
```

STRUCTURE D'UN PROGRAMME APPELANT

```
IDENTIFICATION DIVISION
.
.
.
DATA DIVISION.
.
.
  WORKING-STORAGE SECTION.
  01 A            PIC X(8).
  01 B.
     05 B1        PIC 9(6).
     05 B2        PIC X(10).
.
.
.
*
*
PROCEDURE DIVISION.

PRINCIPALE SECTION.
  EXEMPLE-APPELANT.
     .
     .
     .
     CALL 'ROUTINE' USING B, A.
     .
     .
     .
```

STRUCTURE D'UN PROGRAMME APPELE

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.   ROUTINE.
.
.
.
DATA DIVISION.
.
.
  WORKING-STORAGE SECTION.
  .
  .
  LINKAGE SECTION.
  01 C            PIC X(8).
  01 D.
     05 D1        PIC 9(6).
     05 D2        PIC X(4).
     05 D3        PIC X(6).
*
*
PROCEDURE DIVISION USING D, C.

PRINCIPALE SECTION.
  DEBUT.
  .
  .
  .
  SORTIE.
   EXIT PROGRAM.
```

Opération de chargement d'une bande magnétique dans un dérouleur.

EXIT PROGRAM en fonction des diverses sorties prévues. La structure schématique d'un programme principal (APPELANT) utilisant un sous-programme de PROGRAM-ID VERIFICATION se trouve sur le listing commençant en page 1148. Le sous-programme VERIFICATION, lui, se trouve dans le second listing commençant en page 1151. En suivant l'énoncé du listing, on peut vérifier l'application de certaines règles précédemment exposées :

— Les champs A et B, décrits dans la WORKING-STORAGE SECTION du programme appelant, sont passés au sous-programme au moyen de l'instruction CALL'VERIFICATION'USING B A.
— Le LITERAL employé dans l'instruction appelante contient le PROGRAM-ID du sous-programme (VERIFICATION).
— Au moyen de la ligne symbolique du sous-programme 'PROCEDURE DIVISION USING D C', on établit une correspondance entre les champs des programmes appelant et appelé.

Remarquons l'ordre de définition des champs. Aussi bien dans l'instruction d'appel :

CALL'VERIFICATION' USING B A

que dans la :

PROCEDURE DIVISION USING D C

on fait d'abord référence au champ de 16 caractères, puis à celui de 8 caractères.

Exemples d'application

Pour conclure cet exposé, nous présentons deux exemples de sous-programmes d'un emploi très fréquent :

1 / CONTRDATE, pour le contrôle et la conversion d'une date,
2 / CALENDRIER, pour le calcul du jour de la semaine correspondant à une date donnée.

Les fonctions accomplies par les deux programmes sont explicitées au niveau des REMARKS...
Observons qu'avant de réaliser le calcul demandé, la routine CALENDRIER appelle CONTRDATE et lui demande de contrôler la date transmise par le programme principal.

EXEMPLE D'APPLICATION : CONTROLE DU FORMATAGE D'UNE DATE

```
 IDENTIFICATION DIVISION.
   PROGRAM-ID.   CONTRDATE.
   REMARKS. ================================================
            ==     SOUS ROUTINE POUR CONVERTIR UNE DATE     ==
            ================================================
            LE FORMAT DE PRESENTATION D'UNE DATE EST CONTROLE
            PAR LA VARIABLE OPTION :
            - OPTION = 1 : CONTROLE DE LA VALIDITE D'UNE DATE
            - OPTION = 2 : CONTROLE ET CONVERSION
                           JJMMAA -> AAMMJJ
            - OPTION = 3 : CONTROLE ET CONVERSION
                           JJMMAA -> AAJJJ (DATE JULIENNE)
                            ====
            LA DATE EVENTUELLEMENT CONVERTIE REMPLACE LA DATE
            D'ENTREE (VARIABLE DATE-ENTREE)
            SI LA DATE EST CORRECTE ERREUR =0 SINON ERREUR =1.
 *
 *
 *
  ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
 *
 *
    WORKING-STORAGE SECTION.
 *
    01 DATE-JJMMAA.
       05 JOUR.
          10 JJ            PIC 99.
       05 MOIS.
          10 MM            PIC 99.
       05 ANNEE.
          10 AA            PIC 99.
 *
    01 DATE-JJAAA REDEFINES DATE-JJMMAA.
       05 ALIGN-JULIEN     PIC 9(1).
       05 AA-JULIEN        PIC 9(2).
       05 JJJ-JULIEN       PIC 9(3).
 *
    01 TAMPON-99           PIC 99.
    01 RESTE               PIC 9 COMP.
       88 BISSEXTILE  VALUE 0.
 *
 *      tableaux de conversion.
```

```
*
    01 JOURS-DES-MOIS.
       05 FILLER           PIC X(24)       VALUE
          '312831303130313130313031'.
    01 TAB-JOURS-MOIS  REDEFINES JOURS-DES-MOIS.
       05 NB-JOURS         PIC 99  OCCURS 12.
    01 BASE-JULIEN.
       05 FILLER           PIC X(36)       VALUE
          '000031059090120151181212243273304334'.
    01 TAB-BASE-JUL REDEFINES BASE-JULIEN
       05 BASE             PIC 9(3) OCCURS 12.
*
*
    LINKAGE SECTION.
    01 OPTION              PIC 9.
    01 DATE-ENTREE.
       05 JJ-ENTREE        PIC 99.
       05 MM-ENTREE        PIC 99.
       05 AA-ENTREE        PIC 99.
    01 ERREUR              PIC 9.
*
*
*
 PROCEDURE DIVISION USING OPTION
                          DATE-ENTREE
                          ERREUR.
*
    INITIALISATION.
      MOVE 0 TO ERREUR.
    CONTROLE-OPTION.
      IF OPTION NOT = 1
                AND   2
                AND   3
          MOVE 1 TO ERREUR
          DISPLAY '*** OPTION', OPTION, 'NON PREVUE' UPON PRINTER
          GO TO FIN-PROGRAMME.
      MOVE DATE-ENTREE TO DATE-JJMMAA.
    CONTROLE-DATE-ENTREE.
      IF DATE-ENTREE = SPACES
          MOVE 1 TO ERREUR
          DISPLAY '*** DATE NON INTRODUITE' UPON PRINTER
          GO TO FIN-PROGRAMME.
       INSPECT JOUR   REPLACING ALL LEADING SPACES BY ZEROES.
       INSPECT MOIS   REPLACING ALL LEADING SPACES BY ZEROES.
       IF    JOUR  NOT NUMERIC
```

```
          OR  MOIS  NOT NUMERIC
          OR  ANNEE NOT NUMERIC
            MOVE 1 TO ERREUR
            DISPLAY '*** DATE NON NUMERIQUE' UPON PRINTER
            GO TO FIN-PROGRAMME.
      CONTROLE-MOIS.
        IF MM GREATER THAN 12 OR LESS THAN 1
            MOVE 1 TO ERREUR
            DISPLAY '*** NUMERO DU MOIS IMPOSSIBLE ' UPON PRINTER
            GO TO FIN-PROGRAMME.
      CONTROLE-BISSEXTILE.
        DIVIDE AA BY 4
                  GIVING TAMPON-99
                  REMAINDER RESTE.
        IF BISSEXTILE MOVE 29 TO NB-JOURS (2).
      CONTROLE-JOUR.
        IF JJ LESS THAN 1 OR GREATER THAN NB-JOURS (MM)
            MOVE 1 TO ERREUR
            DISPLAY '*** JOUR IMPOSSIBLE POUR CE MOIS' UPON PRINTER
            GO TO FIN-PROGRAMME.
      CONVERSION.
        IF OPTION = 1
            GO TO FIN-PROGRAMME.
        IF OPTION = 2
            MOVE JJ TO TAMPON-99
            MOVE AA TO JJ
            MOVE TAMPON-99 TO AA
            MOVE DATE-JJMMAA TO DATE-ENTREE
            GO TO FIN-PROGRAMME.
        MOVE 0 TO ALIGN-JULIEN, JJ-JULIEN.
        MOVE AA-ENTREE TO AA-JULIEN.
        IF BISSEXTILE AND MM-ENTREE GREATER THAN 2
            MOVE 1 TO JJ-JULIEN.
        ADD JJ-ENTREE
            JJ-JULIEN
            BASE (MM-ENTREE)   GIVING JJ-JULIEN.
        MOVE DATE-JULIEN TO DATE-ENTREE.
    *
    *
      FIN-PROGRAMME.
        EXIT PROGRAM.
```

EXEMPLE D'APPLICATION : CALCUL DU JOUR DE LA SEMAINE

```
 IDENTIFICATION DIVISION.
   PROGRAM-ID.   CALENDRIER.
   REMARKS.  ==========================================
             ==      SOUS ROUTINE  CALENDRIER        ==
             ==========================================
               CETTE ROUTINE ACCEPTE EN ENTREE UNE DATE
             (DATE-ENTREE) DANS LE FORMAT JJMMAA; ELLE
             RESTITUE LE JOUR DE LA SEMAINE CORRESPONDANT
               UTILISE LA SOUS ROUTINE CONTRDATE
               LE CALCUL EST EFFECTUE EN PRENANT COMME
             BASE LE 1 JANVIER 1900 (LUNDI)
               EN CAS D'ERREUR LE CALCUL EST INTERROMPU
             ==========================================
*
*
*
 ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
*
*
   WORKING-STORAGE SECTION.
*
   01 OPTION                PIC 9.
   01 ERREUR                PIC 9.
   01 DATE-JJMMAA.
      05 JOUR.
         10 JJ              PIC 99.
      05 MOIS.
         10 MM              PIC 99.
      05 ANNEE.
         10 AA              PIC 99.
*
   01 DATE-JJAAA REDEFINES DATE-JJMMAA.
      05 ALIGN-JUL          PIC 9.
      05 AA-JULIEN          PIC 9(2).
      05 JJJ-JULIEN         PIC 9(3).
*
   01 QUOTIENT              PIC 9(5).
   01 RESTE                 PIC 9(5).
   01 NB-JOURS-BASE         PIC 9(5).
   01 INDICE                PIC 9.
*
*         TABLEAUX DES JOURS DE LA SEMAINE
```

```
*
   01 TABLE-NOM.
      05 FILLER          PIC X(3)        VALUE 'DIM'.
      05 FILLER          PIC X(3)        VALUE 'LUN'.
      05 FILLER          PIC X(3)        VALUE 'MAR'.
      05 FILLER          PIC X(3)        VALUE 'MER'.
      05 FILLER          PIC X(3)        VALUE 'JEU'.
      05 FILLER          PIC X(3)        VALUE 'VEN'.
      05 FILLER          PIC X(3)        VALUE 'SAM'.
   01 NOM REDEFINES TABLE-NOM.
      05 NOM-JOUR        PIC X(3) OCCURS 7 TIMES.
*
*
   LINKAGE SECTION.
   01 DATE-ENTREE        PIC X(6).
   01 J-SEM              PIC X(3).
*
*
 PROCEDURE DIVISION USING DATE-ENTREE, J-SEM.
*
 INITIALISATION.
   MOVE DATE-ENTREE TO DATE-JJMMAA.
   MOVE 3 TO OPTION.
   CALL 'CONTRDATE' USING OPTION, DATE-JJMMAA, ERREUR.
   IF ERREUR NOT = 0
       DISPLAY '*** ERREUR DATE ENTREE' UPON PRINTER
       STOP RUN.
*
*
 CALCUL.
   DIVIDE AA-JULIEN BY 4 GIVING QUOTIENT REMAINDER RESTE.
   IF RESTE = 0 SUBTRACT 1 FROM QUOTIENT.
   COMPUTE NB-JOURS-BASE = AA-JULIEN  * 365 +
                           JJJ-JULIEN +
                           QUOTIENT.
   DIVIDE NB-JOURS-BASE BY 7 GIVING QUOTIENT REMAINDER RESTE.
   COMPUTE INDICE = RESTE + 1.
   MOVE NOM-JOUR (IND) TO J-SEM.
*
*
 FIN-PROGRAMME.
   EXIT PROGRAM.
```

Le langage Fortran

Le Fortran (**FOR**mula **TRAN**slator) a été un des premiers langages de haut niveau tout spécialement destiné à l'usage scientifique. La première version officielle remonte à 1957 et, depuis cette date, de nombreuses modifications et adjonctions ont abouti au Fortran ANSI 77, version moderne assez proche du Basic. Le sigle ANSI vient des initiales de American National Standard Institute, l'équivalent de notre Afnor pour la normalisation.

En 1977, ANSI a mis la main aux dernières modifications. La version ANSI 77 est, de ce fait, un standard qui garde une certaine souplesse, car il existe d'autres formes de Fortran qui s'en écartent plus ou moins.

A l'époque où ont été mis au point le Fortran et le Cobol, l'ordinateur recevait les instructions au moyen d'un seul support physique : la carte perforée. En périphérique d'entrée, la machine disposait d'un lecteur de cartes et non d'un clavier.

Chaque instruction était tapée sur le clavier d'une perforatrice de cartes (sans aucune connexion à l'ordinateur), en respectant un format précis. Ainsi, le lecteur de cartes, puis le compilateur pouvaient interpréter de manière univoque le sens de l'instruction.

Le format de perforation des instructions sur les cartes, qui sont encore de nos jours utilisées mais plus rarement, est donc pris en considération par les compilateurs Fortran.

Durant les dix dernières années, l'arrivée du support magnétique a bouleversé les techniques de saisie, éliminant peu à peu les cartes perforées au profit des stockages des programmes sur disque et sur bande. Cependant, le format des instructions reconnu par les compilateurs n'a pas changé. Les diverses parties (champs) composant une instruction sont toujours tapées sur une longueur totale de 80 colonnes, puis découpées dans le format des cartes perforées.

Un lecteur de cartes IBM 3504. Il y a quelques années, les lecteurs de cartes étaient encore les seuls organes d'entrée pour tous les ordinateurs.

IBM

Pour cette raison, et comme pour le Cobol, nous ferons référence aux différents caractères d'une instruction en indiquant leur position sur une carte (colonne 1, colonne 2, etc.), même s'il s'agit d'une saisie au clavier. Nous présenterons le Fortran en nous appuyant largement sur les analogies qu'il présente avec le Basic, quitte à signaler leurs différences

Caractéristiques fondamentales du Fortran comparé au Basic

Tout d'abord, il s'agit d'un langage compilé. Il n'offre donc pas la facilité d'emploi et de mise au point du Basic interprété. Deuxième différence importante : la numérotation des lignes. En Basic, chaque ligne est numérotée dans un ordre croissant ; en Fortran, la numérotation est facultative : elle permet au programmeur d'identifier une ligne particulière (emploi de l'instruction GO TO, par exemple).
Conséquence : les numéros de ligne n'étant pas obligatoires, les sous-programmes Fortran seront identifiés différemment du Basic. Alors qu'en Basic, les sous-programmes sont adressés en se référant au numéro de la ligne initiale (par exemple GOSUB 1000), en Fortran ils seront « marqués » d'un symbole choisi par le programmeur.
Exemple : un sous-programme calculant la somme A = B + C sera en Basic :

```
1000 REM Somme
1010 A = B + C
1020 RETURN
```

et en Fortran :

```
SUBROUTINE SOMME
A = B + C
RETURN
```

En Fortran, la première ligne du sous-programme doit contenir le mot SUBROUTINE, suivi du nom symbolique (ici : SOMME) qui sera employé pour appeler le sous-programme lui-même.
En Basic standard, les instructions (y compris celles des sous-programmes) accèdent à toutes les variables. On peut ainsi attribuer des valeurs aux variables B et C dans n'importe quelle ligne du programme, calculer leur somme dans un sous-programme (le 1000) et utiliser le résultat (A) dans une autre zone du programme.
En Fortran, cela n'est possible qu'avec l'aide d'instruction spéciales :
chaque sous-programme utilise localement ses propres variables qui sont donc inconnues ailleurs.
Le transfert des variables entre sous-programmes se réalise de deux manières :

1 / en transférant les valeurs comme paramètre,
2 / en faisant une déclaration COMMON.

L'appel d'un sous-programme Fortran utilise l'instruction CALL... à la place de GOSUB... en Basic.
Par exemple, l'instruction d'appel du sous-programme précédent est CALL SOMME (en Basic on aurait eu GOSUB 1000). Pour transférer des variables en tant que paramètres, il faut en spécifier le nom symbolique, aussi bien à l'appel qu'à la première instruction du sous-programme qui s'en sert.
Dans l'exemple précédent, l'appel devient :

```
CALL SOMME (A, B, C)
```

et le sous-programme (qui s'appelle SOMME) doit commencer par l'instruction :

```
SUBROUTINE SOMME (A, B, C).
```

La comparaison entre les versions Basic et Fortran d'un même programme est présentée dans la tableau ci-contre.
En Fortran, tous les numéros de ligne ont disparu, sauf pour la première (ligne 10) qui est le point d'arrivée du GO TO (ligne 40 Basic et équivalent Fortran). Pour être identifiée, la ligne d'arrivée doit avoir un nom ou étiquette (label).
En Fortran, l'éventuelle numérotation des lignes n'a qu'une signification d'étiquette et ne suit aucune logique particulière.
La ligne 10 peut précéder la 5 et se trouver après la 1000. En Basic aussi, il est possible d'utiliser une numérotation avec des intervalles variables, mais l'ordre des valeurs est toujours croissant.
La seconde méthode de transfert des valeurs consiste à déclarer « commune » les para-

COMPARAISON DES VERSIONS FORTRAN ET BASIC D'UN MEME PROGRAMME

Version Basic

```
10 C=10
20 B=4.7
30 GOSUB 1000
40 GOTO 10
1000 REM Somme
1010 A=B+C
1020 RETURN
```

Version Fortran

```
10 C=10
   B=4.7
   CALL SOMME (A,B,C)
   GOTO 10
   SUBROUTINE SOMME (A,B,C)
   A=B+C
   RETURN
```

mètres à transférer grâce à l'instruction COMMON, analogue à celle du Basic. Le programme précédent devient (en Fortran) :

```
      COMMON A, B, C
10    C = 10
      B = 4.7
      CALL SOMME
      GOTO 10
      SUBROUTINE SOMME
      COMMON A, B, C
      A = B + C
      RETURN
```

En réalité, ce programme contient une erreur. Le compilateur est informé que le point de départ du sous-programme (SUBROUTINE SOMME) et sa fin, mais il ignore le début et la fin du programme.
En Basic, ce début correspond à la ligne ayant la numérotation la plus basse. En Fortran, il faut spécifier le point de départ avec l'instruction PROGRAM suivie d'un nom symbolique affecté au programme :

```
1       PROGRAM ESSAI
2     C EXEMPLE
3       COMMON A, B, C
4    10 C = 10
5       B = 4.7
6       CALL SOMME
7       GOTO 10
8       STOP
9       END
10      SUBROUTINE SOMME
11      COMMON A, B, C
12      A = B + C
13      RETURN
14      END
```

La numérotation croissante, qui apparaît à gauche des instructions, ne fait pas partie du programme et ne sert qu'à repérer les lignes pour d'éventuelles corrections.
La programmation commence avec l'instruction PROGRAM... (ligne 1) ; la ligne 2 contient un commentaire signalé de diverses manières. Les symboles les plus employés (comme le REM ou le symbole du Basic) sont C ! * tapés comme premier caractère de l'instruction.
Le Basic n'a pas de format fixe pour introduire les instructions : chaque ligne commence par son propre numéro et on poursuit avec l'instruction.
Par exemple, les instructions suivantes sont valables :

```
   10 A = B + C
10    A = B + C
1000  A = B + C
```

Noter : en Fortran, les instructions commencent à la colonne 7 (d'une carte hypothétique). Les commentaires doivent comporter le symbole indicatif lu en colonne 1, les lignes de suite (quand une instruction dépasse le cadre d'une seule ligne), un caractère quelconque (excepté le zéro et le blanc) en colonne 6. Les colonnes 1 à 5 sont réservées aux numéros de ligne.

Opérateurs arithmétiques, logiques et relationnels

S'agissant des opérateurs arithmétiques, le calcul des expressions s'effectue comme en Basic, sauf pour l'élévation à la puissance indiquée

QUELQUES DIFFERENCES ENTRE FORTRAN ET BASIC

	FORTRAN	BASIC
Numéro de ligne	Optionnels ; ne suivent aucun ordre précis	Nécessaires et en ordre croissant
Début programme	Nécessaire et identifié par un nom avec l'instruction PROGRAM nom	Commence toujours par la ligne affectée du plus petit numéro
Sous-programmes	Identifiés par un nom symbolique précédé d'un mot réservé SUBROUTINE. Finissent sur les instrutions RETURN et END	Identifiés par un numéro de ligne. S'arrêtent à l'instruction RETURN
Appel des routines	Instruction CALL suivie par le nom symbolique : CALL ESSAI	Instruction GOSUB suivie du n° de ligne : GOSUB 1000
Transferts de valeurs	Comme paramètre CALL ESSAI (A,B,C). Comme déclaration commune COMMON A,B,C	Pas nécessaire en termes explicités. Chaque point du programme accède à toutes les variables

par le symbole ** (comme en Cobol) : A ** 2 équivaut à A ↑ 2 ou à A ∧ 2.
Les opérateurs de relation sont également les mêmes, mises à part quelques différences de syntaxe avec le Basic.
Alors qu'en Basic, ces opérateurs sont indiqués symboliquement (>, <, =), en Fortran on les présente par un sigle normalement composé de deux lettres, précédé et suivi par un point.
Même constatation, enfin, pour les opérateurs logiques, sauf que la syntaxe du Fortran prévoit des points à droite et à gauche du code mnémonique (voir tableau du bas de la page 1157).

Variables et constantes en Fortran

Le langage Fortran accepte au moins les types de données suivantes :

— Entiers
— Réels en simple précision
— Réels en double précision
— Complexes en simple précision
— Complexes en double précision
— Logiques (booléennes)
— Caractères

D'autres formes prévoient des « entiers condensés » (short integer) et des « logiques condensés » (short logical) occupant un nombre de bits inférieur et économisant de l'espace mémoire.
Leur signification est proche du Basic, à l'exception des nombres complexes (qui n'ont pas d'équivalents en Basic standard) et des entiers (pour lesquels le Fortran prévoit une déclaration implicite). En Fortran, toutes les variables commençant par I, J, K, L, M, N sont supposées entières par défaut.
Cette différence, par rapport au Basic, n'a qu'une importance relative. Le programmeur acquiert rapidement l'habitude d'utiliser les lettres mentionnées pour des variables entières et, souvent, il les emploie aussi en Basic. Ceci explique pourquoi de nombreux programmes Basic font usage de variables tels que I, J, comme indices de boucles, même si ce langage ne les reconnaît pas, a priori, comme des entiers.
Les variables complexes constituent, elles, une particularité du Fortran (même si certaines formes de Basic les prévoient), en raison surtout de son usage scientifique. Limité à des applications bien précises, l'emploi de variables complexes implique d'importantes connaissances techniques.
Les données de type logique ont les mêmes fonctions qu'en Basic ; elles prennent les

valeurs **vrai** (TRUE) et **faux** (FALSE) représentées par les valeurs 1 et 0, comme en Basic. Cependant la syntaxe est différente puisque, en Fortran, TRUE et FALSE doivent être précédés et suivis d'un point : .TRUE. et .FALSE. En plus des variables et des constantes numériques, le Fortran prévoit l'emploi de constantes :

— hexadécimales
— octales
— Hollerith

Les constantes octales et hexadécimales ont le même sens qu'en Basic tout en employant une symbolique différente.
La notation la plus employée pour les octales utilise la lettre B après le nombre :

10B = 8 décimal

et, pour les hexadécimales :

Z'valeur'

où (valeur) indique la valeur hexadécimale de la constante (Z'A' = 10 décimal).
Les constantes d'Hollerith sont une forme particulière de chaîne, avec une syntaxe :

nH XXXX

où n indique le nombre de caractères de la chaîne (XXXX) et H (abréviation de Hollerith) le symbole indiquant le type de constante.

Par exemple :

19HCECI EST UN ESSAI

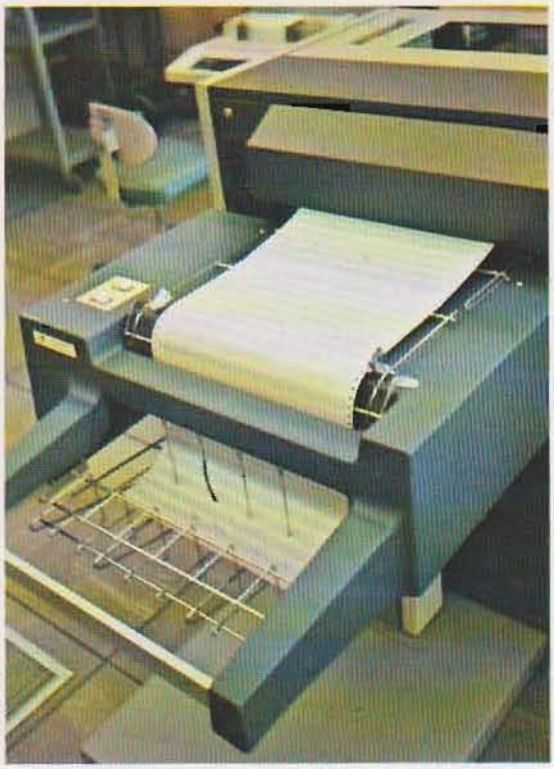
est une forme valable de constante de Hollerith.
Remarquons que les espaces entre les mots sont comptés.
Le traitement des chaînes en Fortran est assez différent du Basic. Précisons qu'il n'existe pas en Fortran de variable de type chaîne ; on n'utilisera donc pas de noms avec le symbole $. Les caractères d'une chaîne sont mémorisés en variables entières, occupant 8 bits chacun. Dans une variable entière (normalement 16 bits), on peut ranger 2 caractères, l'un dans la moitié gauche (bits 0 à 7) et l'autre dans la moitié droite (bits 8 à 15). Pour mémoriser plus de 2 caractères, il faut dimensionner un tableau composé d'autant d'éléments qu'il existe de couples de caractères.
Outre les entiers, d'autres types de tableaux (par exemple en double précision) sont parfois utilisés. Dans ce cas, le nombre de caractères contenu dans un élément est supérieur. Certains compilateurs admettent une méthode typique du Basic qui consiste à désigner un tableau avec un nom de variable. La gestion des tableaux est alors analogue à celle du Basic, sans passer par $.
Par exemple, sont identiques* les attributions :

NOM = « Essai » (Basic).
NOM : = 'Essai' (Fortran)

(*) En Fortran, on utilise normalement les guillemets simples.

OPERATIONS LOGIQUES ET DE RELATION

Fortran	Basic	Abrév. angl.	Signification	Exemple Fortran	Exemple Basic
.EQ.	=	Equal	égal	IF (A.EQ.B)	IF A=B
.NE.	<>	Not equal	différent	IF (A.NE.B)	IF A<>B...
.LT.	<	Less than	inférieur	IF (A.LT.B)	IF A < B...
.GT.	>	Greater than	supérieur	IF (A.GT.B)	IF A > =B...
.LE.	< =	Less-equal	inférieur ou égal	IF (A.LE.B)	IF A < =B...
.GE.	> =	Greater-equal	supérieur ou égal	IF (A.GE.B)	IF A > =B...
.NOT.	NOT		négation	.NOT.B	NOT B
.AND.	AND		ET logique	A.AND.B	A AND B
.OR.	OR		OU logique	A.OR.B	A OR B



Marka/Infophoto

Alimentation en continu d'une imprimante.

L'opération d'extraction d'une partie de la chaîne exige d'en spécifier les limites (caractères extrêmes).
Dans NOM = « Essai », la ligne :

```
NOM 1 = NOM (2 :3)
```

extrait et attribue à NOM 1 la valeur « ss », elle équivaut en Basic à NOM 1$ = MID$(NOM$ (2,2)).
En outre, on peut concaténer 2 chaînes avec le symbole. Ainsi, les lignes.

```
N1 = 'Essai'.
N2 = 'de concaténation'
N = N1 / / N2
```

attribuent à la variable N la valeur « Essai de concaténation ». L'équivalent Basic emploie le symbole + (N1$ + N2$) ou &.
En Fortran, si on attribue à une variable la valeur résultant d'un calcul quelconque, le compilateur lui affecte un type lié aux opérateurs. Exemple : on définit A et B comme des entiers ; le calcul A = B + C mémorise le résultat dans la variable entière A.
Le tableau ci-contre liste les différentes combinaisons possibles et permet d'éviter les erreurs de troncage, provoquées par des déclarations erronées. Exemple : si on multiplie un réel en simple précision par un entier, le résultat sera un réel en simple précision. Mémorisé dans un entier, le résultat induirait des erreurs de troncage. Aussi est-il recommandé d'utiliser une variable d'un type plus long que celui des différents opérateurs du calcul.

Déclaration du type de variables

En Fortran, le type est défini par le programmeur (entier, réel...), avec une syntaxe de déclaration :

```
type nom-1, nom-2, ...
```

où « type » désigne un type de variables et « nom-1, nom-2, ... » des noms symboliques de variables auxquelles on désire attribuer le type précisé.
Les 6 types principaux prévus en Fortran sont :

INTEGER	Entier
REAL	Réel en simple précision
DOUBLE PRECISION	Réel en double précision
COMPLEX	Complexe
LOGICAL	Logique
CHARACTER	Caractère

Ainsi, les lignes :

```
INTEGER A, NOM, K
REAL V
```

définissent comme entières les variables A, NOM et K et comme réelle la variable V.
Ce type de déclaration présente des formes et des fonctions analogues en Basic ;

```
INTEGER A
DOUBLE PRECISION Z
```

équivaut à :

```
DEFINT A
DEFDBL Z
```

Notons, qu'en Basic, la déclaration de type réel n'est pas nécessaire (elle existe toutefois avec

pour syntaxe DEFSNF) dans la mesure où les variables sont supposées réelles par défaut, sauf indication contraire. En Fortran, il se produit la même chose mais la déclaration de type REAL est à faire, ne serait-ce que pour définir comme réelles des variables dont le nom commence par I, J, K, L, M, N.
Dans certaines formes de Fortran, il est possible de définir un type en précision étendue. Ainsi, la déclaration INTEGER devient INTEGER * 2 (précision double) ou INTEGER * 4 (précision quadruple). INTEGER * 2 déclare toujours un nombre entier, mais avec une précision double par rapport à celle que donne INTEGER (à ne pas confondre avec DOUBLE PRECISION qui concerne les réels).
La déclaration CHARACTER définit une variable de chaîne avec la syntaxe :

CHARACTER * n NOM

La variable appelée nom contiendra n caractères. Si on omet n, le compilateur supposera qu'il est égal à un.
Le résultat est semblable à celui obtenu en Basic en écrivant le nom de la variable suivi du symbole S ; mais dans ce cas, il faut définir la longueur de la variable comme nombre de caractères. En Basic, c'est le système qui adapte l'espace réservé à la chaîne, au fur et à mesure que le contenu varie ; en Fortran, la longueur est posée à priori.
La déclaration CHARACTER concerne également un tableau. Ainsi, la déclaration :

CHARACTER * 3V(10)

définit 10 chaînes (V) des caractères chacune.
L'instruction DIMENSION (dimensionnement des tableaux) est alors inutile, dans la mesure où l'espace mémoire est déjà réservé par la déclaration de type.
Les déclarations de type sont « explicites »

TYPES DE VARIABLES RESULTANT DES CALCULS

PREMIER TERME

SECOND TERME	**Entier réduit**	**Entier**	**Réel en simple précision**	**Réel en double précision**	**Complexe en simple précision**	**Complexe en double précision**
Entier réduit	ER	E	R	DP	C	CD
Entier	E	E	R	DP	C	CD
Réel en simple précision	R	R	R	DP	C	CD
Réel en double précision	DP	DP	DP	DP	CD	CD
Complexe en simple précision	C	C	C	CD	C	CD
Complexe en double précision	CD	CD	CD	CD	CD	CD

ER = entier réduit
E = entier
R = réel en simple précision
DP = réel en double précision
C = complexe en simple précision
CD = complexe en double précision

dans la mesure où chaque variable est indiquée avec son propre nom. On peut définir en même temps le type de plusieurs variables regroupées autour de l'initiale du nom : on parle alors de déclaration « implicite ».
La syntaxe est la suivante :

```
IMPLICIT type (X-Y)
```

où X et Y indiquent l'intervalle considéré. La déclaration implicite :

```
IMPLICIT INTEGER (A-L)
```

définit comme **entières** toutes les variables comprises entre A et L. Sa signification est identique au Basic :

```
DEFINT A-L
```

Certains compilateurs Fortran possèdent l'option IMPLICIT NONE, qui élimine toutes les déclarations implicites.

Dimensionnement des variables structurées

Les instructions qui réservent des zones de mémoire à des variables dimensionnées sont les mêmes qu'en Basic : DIMENSION (en Basic DIM) et COMMON.
DIMENSION est employée de la même manière que DIM Basic. Mais COMMON a une autre signification, elle sert aussi à déclarer commun un tableau et, en même temps, à le dimensionner. La déclaration de type offre également cette opportunité puisqu'il est possible de dimensionner un tableau.

Les instruction COMMON et EQUIVALENCE

Les programmes Fortran et Basic ont la même structure : un programme principal appelle des sous-programmes accomplissant chacun sa propre tâche.
Ils ne se différencient que par le transfert des variables. En Fortran, chaque routine a ses propres variables, dans son sein, et qui restent, à moins qu'elles n'aient été déclarées communes par COMMON. Dans la déclaration COMMON, comme dans celle du type, des variables structurées peuvent être dimensionnées avec, pour double effet, de rendre communs certains tableaux et d'en permettre le dimensionnement.
Exemple : si la variable A(20) doit être utilisée à divers endroit du programme, avec l'instruction :

```
COMMON A(20)
```

on définit la dimension et, simultanément, on impose l'usage commun du tableau.
Dans le graphique de la page 1161 on trouve, représentées schématiquement, certaines situations typiques de l'emploi des instructions COMMON et DIMENSION. L'instruction COMMON peut être employée pour transférer simultanément un bloc de variables désigné par un nom-étiquette **(labelled common).**
Dans :

```
COMMON/ESSAI/V1,R,K,M
```

ESSAI nomme le bloc de variables V1,R,K,M. L'usage du bloc permet de modifier les noms des variables d'un sous-programme à l'autre. Si l'instruction précédente est insérée dans le programme principal et dans un sous-programme, par exemple, nous aurons :

```
COMMON/ESSAI/A1,A2,B(2)
```

qui mettaient en correspondance les significations suivantes :

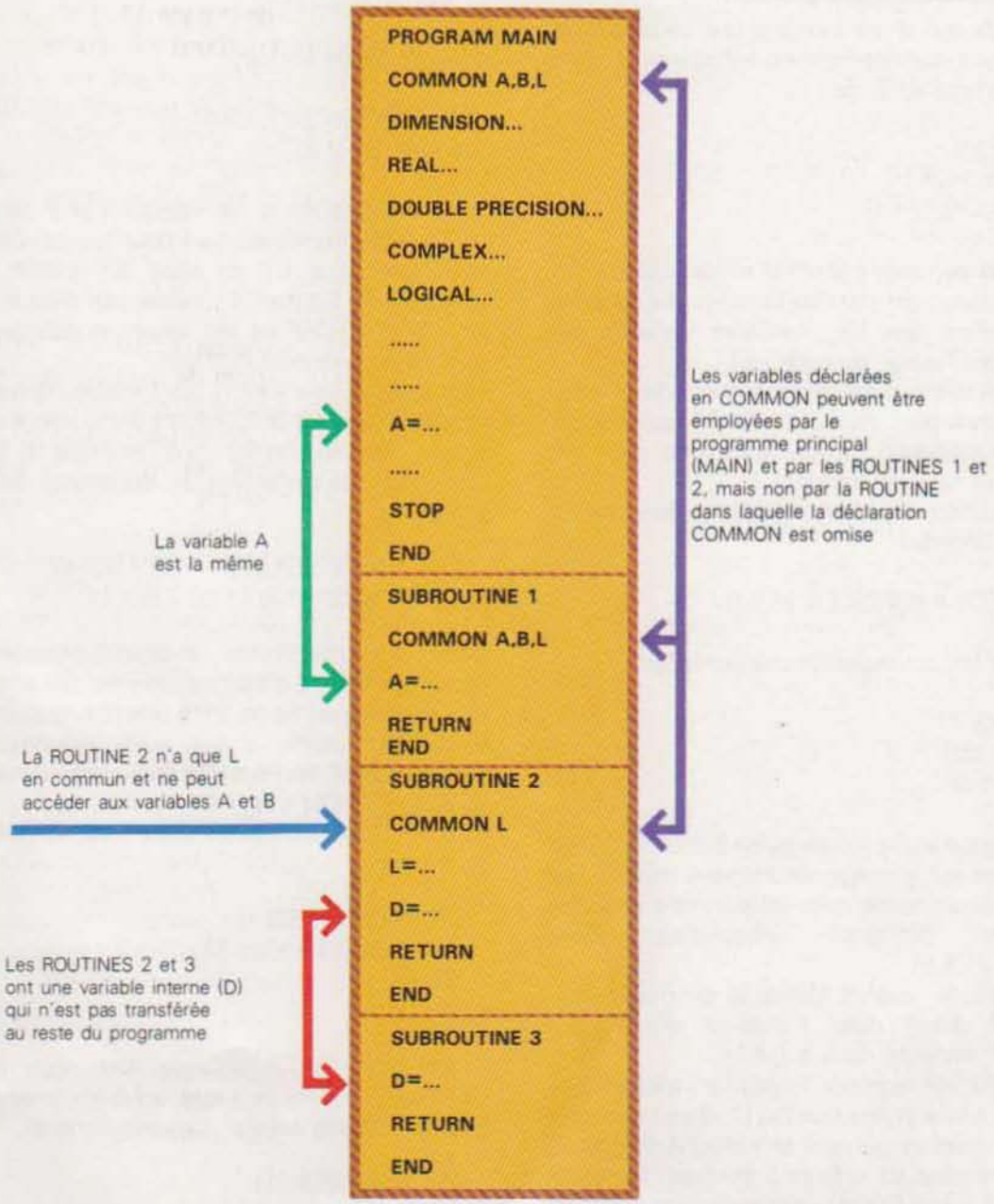
```
A1=V1, A2=R, B(1)=K, B(2)=M
```

Faire appel à une étiquette commune revient à assigner une zone de mémoire aux valeurs d'un bloc de variables. Quand, dans l'instruction, le nom de bloc n'apparaît pas, on parle alors de **blank common.** L'instruction COMMON A(20) est un exemple. Contrairement au Basic, en Fortran, on peut déclarer « équivalents » des noms de variables (même dimensionnées). L'instruction EQUIVALENCE (A,B) assigne, aux deux noms A et B, la même zone mémoire. Pour des variables dimensionnées, il faut s'assurer que l'EQUIVALENCE concerne les éléments contigus. Les instructions :

```
DIMENSION    C(3)
EQUIVALENCE (C(1)A), (C(2)B)
```

sont correctes, dans la mesure où les éléments C(1) et C(2) sont contigus.
En revanche, l'instruction :

TRANSFERTS DE VALEURS AVEC "COMMON"



```
EQUIVALENCE (C(1),A),(C(3),D)
```

comporte une erreur : C(1) et C(3) appartiennent au même tableau sans être contigus.

Instructions d'assignation

L'attribution d'une valeur à une variable s'obtient automatiquement en utilisant le symbole =, comme en Basic.

```
V=3.5
A=7+V*2
B=A*3/(2+1)
```

Ces formes sont correctes et identiques à celles du Basic dans ses versions les plus récentes (rappelons que les premières versions prévoyaient l'usage du code LET).
Un deuxième type d'assignation des valeurs est donné par l'instruction DATA, qui a la même signification qu'en Basic, mais avec une syntaxe bien différente.
L'instruction Fortran (notation américaine du point décimal) :

```
DATA A,R,V / 3.7,4.21,5.9 /
```

établit les correspondances suivantes :

```
A=3.7
R=4.21
V=5.9
```

La logique est la même qu'en Basic : à chaque variable est affectée, de manière linéaire, une des valeurs numériques listées, mais selon des formes distinctes (l'équivalent Basic 3.7,4.21,5.9).
Au-delà de l'aspect formel, la principale différence réside dans l'absence d'instruction READ, implicite dans la DATA.
En Basic, on regroupe toutes les variables assignées à une instruction DATA dans un endroit du programme qui peut être éloigné de celui où sont rangées les valeurs à attribuer. L'instruction READ incrémente le pointeur et va extraire les valeurs dans la zone DATA.
Telle qu'elle est structurée, l'instruction Fortran n'autorise pas cet éclatement géographique : variable et valeur à attribuer doivent apparaître dans la même instruction.

Ainsi les instructions Basic :

```
10 READ A,B,C
20 READ D
30 RESTORE
40 READ E,F
50 DATA 1,2,3,4
```

deviendront en Fortran :

```
DATA A,B,C,D/1,2,3,4/
DATA E,F/1,2/
```

Dans la forme Basic, les valeurs 1 et 2 de la DATA sont employées tant pour les variables A, B que pour E,F (la ligne 30 rétablit le pointeur). En Fortran, il n'existe pas d'équivalent de RESTORE et les valeurs numériques sont nécessairement répétées.
Pour assigner des valeurs aux variables dimensionnées (tableau) on a recours à une forme de boucle implicite comme dans l'exemple du tableau V(3) (de dimension 3), qui admet deux expressions :

```
DATA V(1), V(2),V(3) / 11.6,7.9,3.1 /
DATA (V(I),I=1,3) / 11.6,7.9,3.1 /
```

Observons qu'en Fortran, le dimensionnement implicite en phase d'assignation n'est pas autorisé. Alors qu'en Basic notre exemple est sans erreur (la variable V est automatiquement dimensionnée), en Fortran il exige une instruction DIMENSION V(3).
La forme complète serait alors (avec le point décimal) :

```
DIMENSION V(3)
DATA (V(I),1=1,3) / 11.6,7.9,3.1 /
STOP
END
```

L'instruction DATA accepte tout type de constante, pourvu qu'il soit cohérent avec le type de variable associé. Les instructions :

```
CHARACTERS CH
DATA CH/'A'/
```

définissent CH comme variable contenant des caractères et lui attribuent le caractère A (notons que, dans la DATA, les caractères sont entre guillemets).

Option fréquemment rencontrée dans l'instruction DATA : le symbole * qui signifie répétition.
Par exemple, la ligne :

```
DATA (V(I),1=1,3) / 3*5 /
```

assigne la valeur 5 aux trois éléments du tableau V(3) : la formule 3* signifie « trois fois la valeur qui suit ».
Pour certains compilateurs Fortran, la boucle implicite peut être omise ; ainsi l'instruction précédente s'écrira aussi :

```
DATA V / 3*5 /
```

la variable V ayant été préalablement définie comme un tableau à 3 éléments (DIMENSION V(3).
Les formes principales de DATA sont résumées dans le tableau ci-dessous. La dernière méthode d'assignation des valeurs numériques typique du Fortran consiste à utiliser l'instruction PARAMETER, dont la syntaxe est :

```
PARAMETER (nom-1=const-1, nom-2=...)
```

Celle-ci définit des constantes avec un nom symbolique. Les constantes peuvent appartenir à n'importe quel type, à condition de rester homogènes avec le type associé au nom (par exemple, constantes entières avec noms commençant par I,J,K,L,M,N).

L'instruction :

```
PARAMETER (PI=3.14)
```

assigne, au nom symbolique PI, la valeur 3.14. Dans ce cas, on n'a besoin d'aucune déclaration de type, dans la mesure où PI est un nom auquel on peut associer une valeur réelle (3.14). Si la compatibilité n'existe pas, il faut définir le type de variable avant PARAMETER. Par exemple :

```
CHARACTER*5 D
PARAMETER (D='ESSAI')
```

définit D comme chaîne de 5 caractères et lui attribue la valeur ESSAI.

Les instructions Fortran

Examinons maintenant les principales instructions Fortran, en insistant sur les différences avec le Basic. En effet, la remarquable similitude entre les deux langages permet de simplifier l'exposé en se limitant à l'énumération des différences fondamentales.

L'instruction GOTO

En Fortran, il existe 3 types de GOTO(*) :

- GOTO inconditionnel
- GOTO calculé
- GOTO assigné

(*) La présence d'un espace entre les mots GO et TO n'est pas significatif, comme c'est le cas en Basic.

FORMAT DE L'INSTRUCTION DATA

CHARACTER CH DATA CH/'A'/	Déclaration de type DATA variable, variable,... / valeur, valeur,.../
DIMENSION V(5) DATA (V(I),I=1,3)/4,2,7,1,8,3/	Dimensionnement DATA boucle/valeurs/
DIMENSION V(5) DATA (V(I),I=1,3)/3*5,1/	Dimensionnement DATA boucle/facteur de répétition*valeur/
DIMENSION V(5) DATA V/5*3,7/	Dimensionnement DATA variable dimensionnée/facteur de rép.*valeur/

Le mode inconditionnel est commun au Fortran et au Basic : même spécification d'adresse du saut (ligne identifiée par un numéro) et même syntaxe :

GOTO numéro de ligne

Bien qu'il ait son équivalent Basic, le GOTO calculé a un format différent. En Basic, l'instruction est ON K GOTO... ; en Fortran la syntaxe est :

GOTO (n° -ligne-1, n°-ligne-2,...), K

Signification : pour K=1, sauter au premier numéro de ligne indiqué (n°-ligne-1), pour K=2 au deuxième, etc. La forme exacte prévoit une virgule avant le paramètre, mais elle est généralement omise. L'indice (paramètre K) peut également être le résultat d'un calcul, ainsi :

GOTO (100, 10, 1240), R+4

déclenchera un saut à une des instructions 100, 10 ou 1240, selon le résultat du calcul R+4 (ou mieux : selon la valeur de la partie entière de ce résultat).
Le GOTO assigné, enfin, n'a pas d'équivalent Basic. Il permet d'employer un symbole à la place du numéro de ligne et doit être précédé d'une instruction. Celle-ci attribue le nom symbolique à la ligne.
L'instruction GOTO 150 devient ainsi GOTO ARRIVEE après avoir assigné, à la ligne 150, le nom ARRIVEE. L'instruction effectuant cette assignation est :

ASSIGN n°-ligne TO nom

Pour l'exemple précédent, on aurait :

```
ASSIGN 150 TO ARRIVEE
. . . .
GOTO ARRIVEE
```

Au moment d'exécuter l'instruction GOTO ARRIVEE, le programme transfère le contrôle à la ligne n° 150.

Les boucles

Le déroulement d'une boucle suit les règles décrites dans la présentation du Basic, avec une syntaxe Fortran particulière :

DO n indice=valeur initiale, valeur finale, pas

DO est le code qui déclenche la boucle (équivalent au FOR) ; « n » est le n° de ligne fin du bloc d'instructions à exécuter itérativement dans la boucle ; « indice » est l'indice de la boucle qui varie de la « valeur-initiale » à la « valeur-finale » avec des progressions égales au « pas » (intervalle constant entre deux indices).
Si la valeur du pas n'est pas donnée, le compilateur la lira par défaut comme égale à 1.
L'instruction :

```
DO 100 I=1,22,3
```

active l'itération de la partie du programme comprise entre cette ligne et celle numérotée 100, en faisant varier l'indice entre les valeurs 1 et 22 avec un pas égal à 3. La ligne d'arrêt de la boucle ne doit contenir aucune référence à l'indice (en Basic, on doit écrire l'instruction NEXT I) et il pourrait s'agir de n'importe quelle instruction Fortran. En réalité, on termine généralement par l'instruction CONTINUE « poursuivre ». Celle-ci n'a aucun effet sur le programme ; son seul but est d'identifier un n° de ligne marquant la fin d'une boucle.
La boucle précédente, complétée, devient :

```
    DO 100 I=1,22,3
    . . . .
    . . . .      Instruction exécutées
    . . . .      de manière itérative
    . . . .
100 CONTINUE
```

En outre, certaines formes de Fortran ont adopté un code plus explicite : END DO. Le programme devient alors :

```
    DO 100 I=1,22,3
    (itération)
100 END DO
```

Boucles implicites. En Fortran, il est possible d'activer une boucle de manière implicite en omettant l'instruction DO. Un exemple illustre le mode d'attribution avec une DATA appliquée à un tableau. En général, la boucle impli-

IBM

Perforatrice de cartes IBM 3325, un des périphétiques standard.

cite (appelée aussi DO implicite) a le format :

XX (tableau indice=val-init, val-fin, pas)...

La partie centrale de l'instruction (entre parenthèses) est la boucle proprement dite ; XX représente une instruction admettant ce type de boucle et les points, une suite éventuelle. Dans le cas d'une assignation avec DATA, la partie XX est remplacée par DATA, et la suite de l'instruction est composée des valeurs numériques qu'il faut attribuer (11.6,7.9,.../)
Il n'existe que 2 types d'instructions admettant la boucle implicite :

— DATA
— Instructions d'E/S

Pour éditer le contenu d'un tableau avec 10 éléments, on peut adopter l'instruction :

```
WRITE (p,n)(V(I),I=1,10)
```

WRITE (équivalent du PRINT Basic) est le code de l'instruction ; « p » et « n » sont des paramètres qui indiquent où (sur quel périphérique) et comment (avec quel format) écrire ; l'expression entre parenthèses indique les variables à écrire.
Le DO implicite est formellement semblable à un FOR... NEXT... exprimé sur une seule ligne. Par exemple, pour écrire les éléments du tableau précédent en Basic, on pourrait avoir :

```
FOR I=1 TO 10 : PRINT V(I) : NEXT I
```

équivalente à l'instruction Fortran :

```
WRITE (p,n)(V(I),I=1,10)
```

Les DO implicites peuvent être emboités, c'est-à-dire contenus l'un dans l'autre.
L'impression d'un tableau à deux dimensions s'obtient avec les instructions :

```
DIMENSION A(5,3)
. . . . . .
WRITE (p,n) ((A(I,J),J=1,3),I=1,5)
```

équivalentes au Basic :

```
10 FOR I=1 TO 5
20 FOR J=1 TO 3
30 PRINT A(I,J)
40 NEXT J
50 NEXT I
```

La forme DO... WHILE... (FAIRE... JUSQU'A CE QUE...)
Ce type de boucle n'est pas prévu dans le Fortran ANSI 77 ; il n'existe que sur certaines machines. Il est semblable au Basic WHILE... END, avec la syntaxe :

DO n WHILE (condition)

où « n » est le n° de ligne où s'arrête le DO et « condition » une condition quelconque qui provoque la répétition du cycle si elle se vérifie. Dans l'instruction :

```
DO 100 WHILE (A.GT.B)
```

la boucle est répétée jusqu'à ce que (WHILE) la valeur de la variable A soit supérieure (.GT.) à la valeur B. Normalement, le n° de ligne est facultatif et la fin du DO est indiqué par END DO. Dans ce cas, l'exemple précédent devient :

```
DO WHILE (A.GT.B)
. . . .
. . . .
END DO
```

sans aucun n° de ligne.

Instructions conditionnelles

Comme en Basic, la condition est obtenue avec IF. En Fortran, il existe trois types de IF :

- IF arithmétique
- IF logique
- IF par blocs

Les deux premiers types sont communs à toutes les versions du langage, alors que le dernier est caractéristique des versions les plus récentes.
La syntaxe de l'IF arithmétique est la suivante :

IF (expression) ligne-1, ligne-2, ligne-3

L'expression est évaluée et le contrôle est transféré à la ligne correspondante en fonction du signe du résultat. Par exemple, si on exécute l'instruction :

```
IF (A+B) 10,100,1000
```

le programme saute à la ligne 10, 100 ou 1000 selon le résultat (négatif, nul, positif) de la somme A+B.
Le contenu de l'instruction est limité à 3 adresses seulement, puisque la sélection s'opère selon le signe : le contrôle passe à la 1ère ligne (10) si le résultat est négatif, à la 2e ligne (100), s'il est égal à zéro et à la 3e (1000) s'il est positif (voir graphique ci-contre).
L'IF arithmétique du Fortran équivaut à 3 IF logiques en série. Par exemple, l'instruction

```
IF (K) 10,20,30
```

signifie

```
IF (K.LT.0) GOTO 10
IF (K.EQ.0) GOTO 20
IF (K.GT.0) GOTO 30
```

Hormis quelques écarts symboliques (.LT.,.EQ.), ces IF correspondent à ceux du Basic (IF K=0 GOTO 10, etc.).
Dans l'IF arithmétique, les nos de lignes peuvent être répétés :

IF (K) 10,10,10 transfère dans tous les cas le contrôle à la ligne 10
IF (K) 10,20,10 transfère le contrôle à la ligne 20 si K=0
IF (K) 10,10,20 transfère le contrôle à la ligne 20 si K > 0

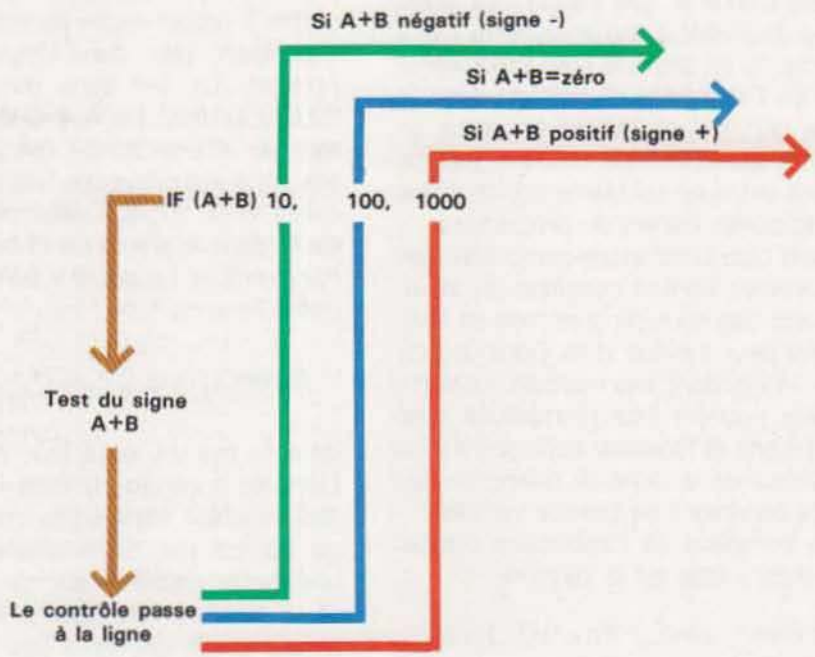
A la différence de l'arithmétique, l'IF logique a les mêmes fonctions que son homologue Basic.
Sa syntaxe est :

IF (condition) instruction

« Instruction » est exécutée seulement si « condition » est vraie. Mais en Fortran, il n'y a pas de THEN. Alors, l'instruction Basic :

```
IF A > B THEN K=1
```

DIAGRAMME SYNTAXIQUE DE UN IF ARITHMETIQUE



devient

```
IF (A.GT.B) K=1
```

« L'instruction » prend la forme d'un type quelconque, à l'exception de DO,END ou d'un autre IF logique.
L'IF par blocs est une extension de l'IF logique : il active des blocs entiers d'instructions au lieu d'une seule. La syntaxe est :

```
IF (expression) THEN
. . . . . .
(bloc d'instructions à exécuter si l'expression est vraie)
. . . . . .
END IF (ou ENDIF)
```

Dans l'IF à bloc apparaît (comme en Basic) le mot THEN et on peut employer la précision ELSE (sinon).
Par exemple, la séquence d'instructions :

```
IF (A.RQ.B) THEN
C=3*K
D=C+2
ELSE IF (A.G.T.B) THEN
C=0
D=1
ENDIF
```

attribue les valeurs C=3*K et D=C+2 à la condition que A=B, alors que pour A B, elle attribue C=0 et D=1. Le nombre de précisions ELSE dans un même IF est à la discrétion des programmeurs. On dispose donc d'un moyen de sélection bien plus puissant, même s'il n'est pas simple à appliquer.
Le langage Fortran, surtout dans sa version ANSI 77, est bien plus structuré que le Basic mais demande une période d'adaptation de la part du programmeur. Le programme est plus complet et souvent plus aisé à modifier.
Le thème de la programmation structurée sera abordé plus loin, à l'occasion de l'exposé sur le langage Pascal qui a justement été conçu pour mieux programmer.
L'emploi de la clause ELSE, à l'intérieur d'un IF, est également prévu dans certaines formes de Basic mais, sauf cas particulier, il n'est pas aussi simple qu'en Fortran.

L'emploi des sous-programmes (ou routines)

On retrouve, en Fortran, la notion de sous-programmes même si, par rapport au Basic standard, ils découlent d'une philosophie tout à fait différente. Si, en Basic, le sous-programme fait partie de l'ensemble et partage avec le programme principal toutes les variables, en Fortran, en revanche, chaque sous-programme construit une entité en soi qui ne communique pas avec les autres parties du programme.
Dans certains cas, la structure particulière des sous-programmes Fortran constitue un atout. Le découpage des sous-programmes en entités distinctes peut s'avérer utile. Dans un programme très long, donc très morcelé, les noms des variables peuvent être réemployés avec des significations différentes, soulageant ainsi le programmeur de la tâche de mémoriser les noms et les conditions de chaque variable.
La syntaxe complète de l'instruction d'appel d'un sous-programme est la suivante :

CALL nom (var-1, var-2,... *n1,*n2...)

où « nom » est le nom symbolique du sous-programme (point d'entrée équivalent au n° de ligne en Basic) ; « var-1, var-2... » les variables éventuelles à transférer au sous-programme appelant ; « n1, n2,... » les points de retour alternatifs.
Les versions Fortran les plus complètes prévoient, en effet, la possibilité d'effectuer un saut de la routine vers un point du programme suivant l'instruction appelante.
Par exemple, l'instruction :

CALL ESSAI (*100,*200,*300)

active le processus de retour paramétré. Selon la sortie sélectionnée dans le sous-programme ESSAI, le contrôle est transféré à l'une des trois lignes indiquées (100,200,300) du programme principal. Naturellement, le paramètre sera mentionné dans les 3 instructions RETURN insérées dans la routine.
Dans le programme ESSAI, au moins trois retours seront prévus : RETURN 1 pour la ligne 100, RETURN 2 pour la ligne 200, RETURN 3 pour la ligne 300.
Ce type de RETURN est paramétrable. Ainsi, en écrivant RETURN K, on déclenche le retour à 100, 200, 300 selon la valeur de K, qui peut être calculée dans le même sous-programme. Si elle n'est pas comprise dans les limites imposées, le contrôle reviendra à la ligne consécutive à l'appel (cette caractéristique n'existe cependant pas dans toutes les versions Fortran). La 1ère ligne du sous-programme (SUBROUTINE...) contient nécessairement autant de références de retour qu'il existe de solutions alternatives en fonction du résultat du calcul lancé (logique déterminée par la valeur de K, dans le graphique ci-contre).
Par exemple, l'appel précédent exige une première ligne du type :

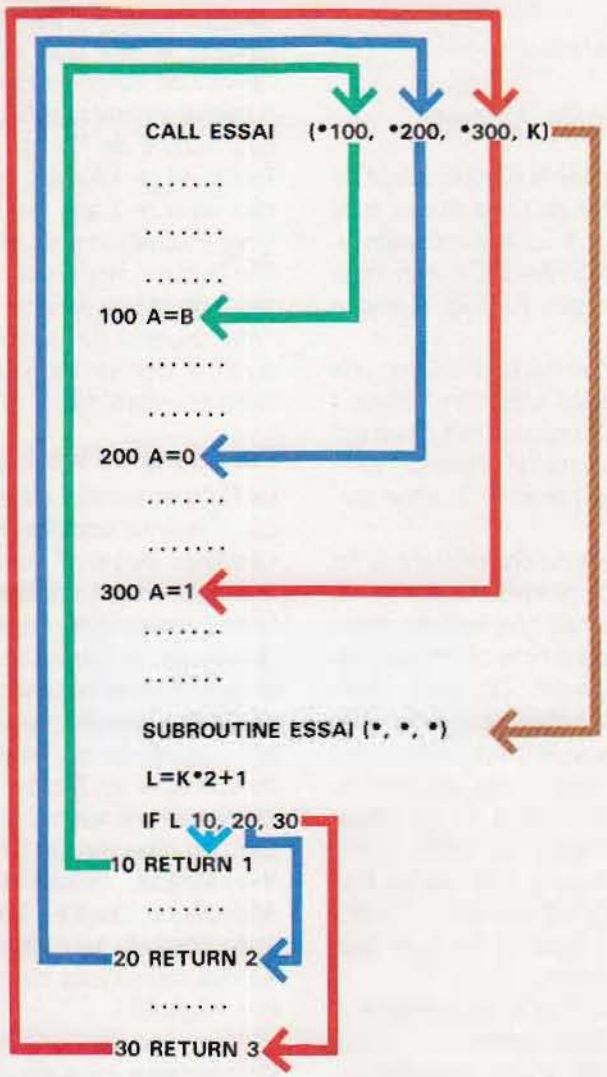
SUBROUTINE ESSAI (*,*,*)

dans la mesure où il faut prévoir 3 retours.
Dans les sous-programmes Fortran, les noms des variables transférées comme paramètres ne doivent pas nécessairement coïncider. Si une routine calcule la somme de deux nombres (A + B) et qu'elle place le résultat dans une troisième (C), elle devra être appelée en citant trois paramètres : les deux nombres à additionner et le résultat. Sa première ligne sera du genre SUBROUTINE SOMME (X, Y, R) en définissant X et Y comme nombres à additionner et R comme résultat.
Dans l'appel, on peut utiliser d'autres noms, à la seule condition de respecter le même ordre : les deux premiers pour les termes de la somme et le troisième pour le résultat.
Dans le programme :

```
      PROGRAM ESSAI
      A = 1
      B = 2
   10 CALL SOMME (A,B,S)
       .......
       .......
      X = 3
      Y = 5
   20 CALL SOMME (X,Y,T)
       .......
      SUBROUTINE SOMME (X,Y,R)
      R = X + Y
      RETURN
```

La ligne 10 (la numérotation 10, 20 n'a été adoptée que pour faciliter la lecture) appelle la

DIAGRAMME SYNTAXIQUE DES RETOURS ALTERNATIFS



routine SOMME avec les paramètres A, B, S ; au retour, la somme est contenue dans S. La ligne 20 appelle la même routine avec les paramètres X, Y, T et, au retour, la somme est contenue dans T.

En d'autres termes, la signification de chaque paramètre dépend essentiellement de sa position.

Dans la routine, on exécute le calcul R = X + Y, donc la calcul du 3e paramètre, de même que la somme des 2 autres. Quel que soit le nom symbolique des paramètres, au retour du sous-programme la 3e variable contient toujours la somme des deux premières.

Instructions de fin et d'arrêt de l'exécution

Dans ce paragraphe, nous allons examiner trois instructions : une sans équivalent Basic, les deux autres avec un sens différent :

PAUSE
STOP
END

PAUSE suspend momentanément l'exécution du programme pour, par exemple, donner le temps à l'opérateur de commander certaines opérations externes, comme de placer le papier sur l'imprimante.
La syntaxe de l'instruction est :

PAUSE n ou PAUSE 'message'

Dans les deux cas, le système signale, avant de suspendre le déroulement du programme, qu'il a rencontré la pause « n » ou le « message ». L'exécution de l'instruction PAUSE « Aligner le papier » génère le message « PAUSE Aligner le papier » à l'écran.
Le programme interrompu est réactivé par une instruction adéquate codée selon la machine ; la plus fréquente est GO. Lorsque cet ordre est introduit au clavier, le système reprend l'exécution du programme à partir de la ligne suivant PAUSE.
L'instruction **END** signale au compilateur la fin de chaque « unité de programme ».
Le Fortran est plus structuré que le Basic standard et chaque sous-programme constitue une entité compilée séparément. On peut donc rencontrer plusieurs END dans le même programme. Un seul concerne le programme principal et indique la fin logique du traitement, alors que les autres signalent la fin de sous-programmes. Normalement, le code END, dans une routine, a le même effet qu'un RETURN mais le contraire n'est pas vrai : si END est omis à la fin d'une routine, on aura une erreur lors de la compilation.
Le **STOP** produit le même effet qu'en Basic : il arrête l'exécution du programme.
En Fortran, on peut l'associer à un message ou à une valeur numérique ; l'un ou l'autre apparaissent sur l'écran lors d'un STOP.
Remarquons que PAUSE et STOP peuvent contenir un message mais doivent utiliser au moins une valeur numérique ; en général, on n'admet pas l'usage de ces instructions sans d'autres éléments.

Les fonctions du Fortran

En Fortran comme en Basic, il existe un ensemble de fonctions particulières (appelé « bibliothèque ») et des fonctions définies par l'utilisateur.

La bibliothèque du Fortran est plus riche en fonctions mathématiques que le Basic. inversement, elle dispose de très peu de fonctions pour le traitement des chaînes. Cette spécialisation s'explique aisément par l'origine délibérément scientifique du Fortran. Cependant, l'évolution parallèle des deux langages les a notablement rapprochés, à tel point que certaines formes de Fortran disposent des mêmes fonctions de chaînes que le Basic, tandis que des versions Basic possèdent toutes les fonctions mathématiques du Fortran.
Par la suite, nous examinerons quelques-unes des principales fonctions du Fortran. Leur variété dépend de la version du compilateur ; aussi ne donnerons-nous que la liste des fonctions standard ANSI 77.

Fonctions de bibliothèque

Le Fortran est très riche en fonctions de calcul car c'était sa vocation première. Les dernières versions incluent aussi certaines fonctions d'enchaînement qui rendent le Fortran apte à toutes sortes d'applications. Par la suite, nous diviserons les fonctions de la version ANSI 77 en groupes homogènes. Cette subdivision n'a pas, à proprement parler, la valeur d'une véritable classification ; elle ne constitue qu'un moyen pour en faciliter la lecture et certaines comparaisons avec Basic.
La compréhension et l'emploi de fonctions mathématiques nécessitant une bonne base scientifique, nous ne les évoquerons que pour être complets. Leur assimilation ne fait pas partie des techniques de programmation et peut être négligée.

Conversion de type. A ce groupe appartiennent les fonctions qui permettent de changer le type d'une variable. Par la suite, on opèrera une subdivision selon le type de variable obtenue.

Les fonctions de **conversion en entier** transforment en entier la valeur de variables réelles ou en double précision, selon la syntaxe :

```
I = INT (R)
I = IFIX (R)
```

(R = Réel, E = entier soit INTEGER en anglais)

Pour transformer en entier une valeur en dou-

B. Coleman/Marka

Micro-ordinateur à base de microprocesseur Intel capable d'exécuter les ordres oralement.

ble précision, il faut écrire :

I = IDINT (B)

A noter que les fonctions équivalentes en Basic sont INT (ou CINT) et FIX.

Les fonctions de **conversion en réel** (simple précision) ont comme syntaxe :

d'entier vers réel R = FLOAT (I)
de double précision vers réel R = SNGL (D)
(R = Réel, I = entier, D = Double précision)

L'équivalent Basic, est CSNG.

La conversion en **double précision** a une syntaxe unique, quel qu'en soit le type :

D = DBLE (X) (X = entier, réel, complexe)

L'équivalent Basic, sauf pour le X complexe, est CDBL (X).

La conversion caractère-nombre (et inversement nombre-caractère) s'obtient avec la syntaxe suivante :

de caractère vers entier I = ICHAR (C)
d'entier vers caractère C = CHAR (I)
(I = entier, C = caractère)

La valeur de E est celle qui est associée au caractère dans le code ASCII.
Les fonctions équivalentes en Basic sont ASC et CHR$.

Pour la **conversion d'un réel en complexe,** la fonction à utiliser (sans équivalent Basic) est CMPLX (R). Elle génère un nombre complexe, avec une partie réelle égale à R et une partie imaginaire égale à zéro.

Fonctions de calcul. Elles sont destinées à certains calculs arithmétiques immédiats.

Pour le **troncage dans la partie entière** de la valeur d'une variable, on utilise les fonctions :

A=AINT(B) A et B réels en simple précision

C=DINT(D) C et D en double précision.

Pour calculer le **le plus grand entier** contenu dans la valeur d'une variable, on utilise :

A=ANINT(B) A et B réels en simple précision
C=DNINT(D) C et D en double précision
I=NINT(B) I entier, B réel en simple précision
I=IDNINT(D) I entier, D en double précision

La **valeur absolue** d'une variable (sans le signe), ou la partie réelle d'un nombre complexe, s'obtient avec les fonctions :

I=IABS(K) I et K entier
A=ABS(B) A et B réels en simple précision
C=DABS(D) C et D en double précision
C=CABS(Z) C réel, Z complexe

La fonction de **transfert de signe** travaille sur deux nombres (A et B) et restitue (-A) si B inférieur à zéro, (A) si B supérieur ou égal à zéro. Les nombres sont nécessairement du même type, avec une syntaxe différente, suivant le type :

ISIGN(A,B) entiers
SIGN(A,B) réels en simple précision
DSIGN(A,B) nombres en double précision

La fonction **différence positive** exige également deux nombres (A, B) et restitue (A-B) si A supérieur à B et 0 si A inférieur ou égal à B :

IDIM(A,B) entiers
DIM(A,B) réels en simple précision
DDIM(A,B) nombres en double précision

La fonction **produit en double précision** calcule le produit de deux nombres réels en tranformant le résultat en double précision :

D=DPROD(A,B) A,B réels en simple précision
D en double précision

La fonction **reste** effectue le calcul :

A-INT(A/B)*B

En donnant ainsi le reste de la division entre A et B.
Son équivalent Basic présente quelques différences de noms :

MOD pour les entiers
AMOD pour les réels
DMOD pour les nombres en double précision

Les fonctions **maximum** et **minimum** permettent d'extraire, respectivement, la valeur maximum et la valeur minimum d'une série de nombres :

R=MAX 0 (A,B,C...) entiers
R=AMAX (A,B,C...) réels en simple précision
R=DMAX (A,B,C...) nombres en double précision
R=MIN 0 (A,B,C...) entiers
R=AMIN (A,B,C...) réels en simple précision
R=DMIN (A,B,C...) nombres en double précision

D'autres formes permettent un usage de type mixte.

Fonctions de chaîne. Le Fortran ANSI 77 ne comporte que deux fonctions pour le traitement des chaînes :

LEN fournit la longueur (en caractères) d'une chaîne
INDEX restitue la position initiale d'une sous-chaîne à l'intérieur d'une chaîne

Fonctions lexicales. Elles permettent de comparer deux chaînes (en suivant l'ordre l'alphabétique) avec un résultat « vrai » ou « faux » selon que la condition de comparaison est vérifiée ou non.
Les fonctions prévues sont :

C=LGE (A,B) vérifie si A = B
C=LGT (A,B) vérifie si A B
C=LLE (A,B) vérifie si A = B
C=LLT (A,B) vérifie si A B

En retour, on aura, dans C, la valeur logique TRUE et FALSE.

Fonctions mathématiques. Pour des raisons de clarté, nous ne présentons, dans ce

paragraphe, que celles qui se rapportent aux réels. Pour la double précision, le plus souvent, il suffit de remplacer la première lettre de la fonction par D. Mais il ne s'agit pas d'une règle générale : pour connaître la syntaxe exacte de la fonction, il suffit de se rapporter au manuel d'utilisation de votre version Fortran.

Description	Fonction	Exemple
Racine carrée	SQRT	B = SQRT(R)
Expotentielle	EXP	B = EXP(R)
Logarithmes	LOG	B = LOG(R)
Logarithmes décimaux	LOG 10	B = LOG10(R)
Sinus	SIN	B = SIN(A)
Cosinus	COS	B = COS(A)
Tangente	TAN	B = TAN(A)
Arcsinus	ASIN	B = ASIN(R)
Arcosinus	ACOS	B = ACOS(R)
Arctangente	ATAN	B = ATAN(R)

Les fonctions trigonométriques utilisent des angles (dans les exemples identifiés par la variable A) exprimés en radians. En Fortran, on prévoit, en outre, les fonctions hyperboliques, dont les noms s'obtiennent en ajoutant la lettre H au terme trigonométrique correspondant (par exemple SINH indique le sinus hyperbolique).

Fonctions programmables par l'utilisateur

Les fonctions définies par l'usager suivent, d'une certaine manière, des règles analogues à celles que nous avons vues en Basic. Reste la différence de fond liée au fait, qu'en Fortran, chaque module (sous-programme ou fonction) est logiquement séparé du reste du programme.
La syntaxe qui permet de définit une fonction particulière est la suivante :

```
type FUNCTION nom (liste des arguments)
```

type	Il définit le type de fonction (réelle, entière, etc.) et prend une des valeurs décrites dans les déclarations (REAL, INTEGER, etc.)
FUNCTION	Mot réservé indiquant la fonction.
nom	Nom attribué à la fonction ; contrairement au Basic, où il n'est composé que d'un caractère (DEF FN A), en Fortran on peut utiliser n'importe quel nom (comme pour les variables)
listes d'arguments	Comme en Basic, elle est composée des arguments que la fonction doit utiliser

Par exemple, la ligne :

```
INTEGER FUNCTION ESSAI ( )
```

définit la fonction entière ESSAI, sans arguments ; la ligne :

```
FUNCTION X (A,B,C)
```

définit la fonction X, qui utilise les arguments A,B,C ; et enfin la ligne :

```
CHARACTER* 4 (L)
```

définit une fonction de chaîne A, de 4 caractères (longueur) avec l'argument L.
En Fortran, les fonctions définies par l'usager présentent une structure semblable à celle d'un sous-programme ; elles peuvent, par exemple, comporter plusieurs lignes, en incluant plusieurs calculs, contrairement au Basic où une seule ligne est utilisée, avec les limites qui en résultent. Une fonction Fortran commence avec la déclaration FUNCTION et se poursuit sur un nombre quelconque de lignes pour se terminer avec RETURN.
Ainsi, la fonction :

```
FUNCTION HYPO (A,B)
HYPO-SQRT((A**2) + (B**2))
RETURN
END
```

calcule l'hypoténuse d'un triangle rectangle de côtés A et B. Pour l'utiliser, il suffit de l'appeler avec le nom symbolique (HYPO).

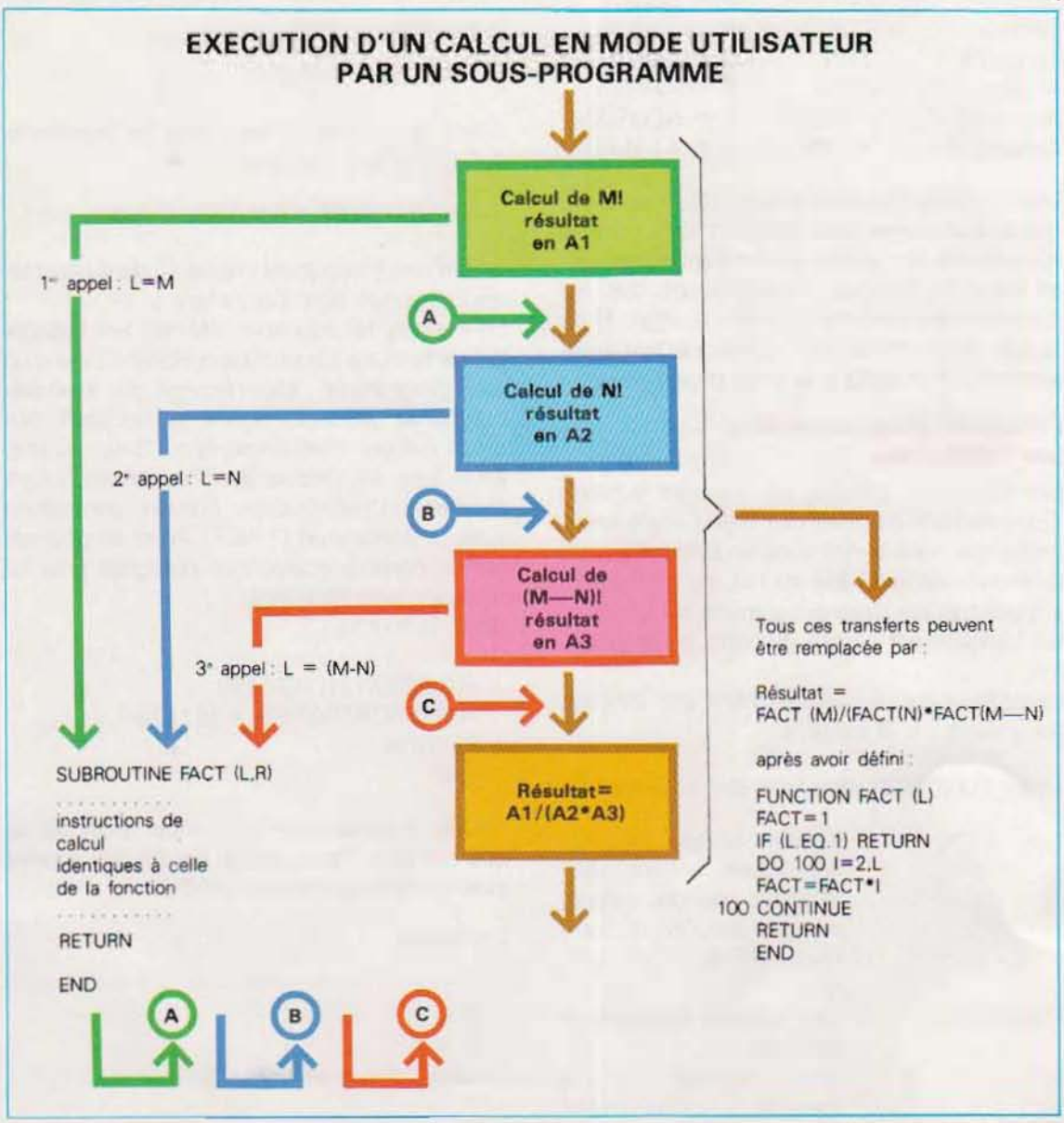
Les lignes :

```
. . . .
A = 3
B = 4
HYPOTEN = HYPO (A,B)
. . . .
```

appellent la fonction précédemment définie et transfèrent le résultat (hypoténuse) dans la variable HYPOTEN.
On aurait pu obtenir le même résultat en définissant HYPO comme sous-programme (dans ce cas il faudrait mentionner 3 paramètres : les côtés, A,B comme paramètres d'entrée et HYPOTEN comme paramètre de sortie).
Le graphique ci-dessous compare une fonction programmée par l'utilisateur à un sous-programme réalisant le même calcul.
Le programme en question (présenté sans le début et sans la fin) calcule les combinaisons de M éléments de classe N (c'est-à-dire pris par groupes de N). La formule est :

$$\text{Combinaisons} = \frac{M!}{N! \times (M-N)!}$$

où M ! (factorielle M) désigne le produit de tous les nombres entiers depuis 1 jusqu'à M (par exemple pour M × 5, on a M ! 1X2X3X4X5).
Si, pour calculer les 3 factorielles, n recourt à un sous-programme, ce dernier devra être appelé trois fois, en mettant en mémoire le résultat de chaque calcul (voir organigramme).
Le sous-programme doit posséder deux para-



paramètres : L (nombre dont on veut calculer la factorielle) et R (résultat du calcul). Au premier appel, on calcul M ! (rangé en mémoire, par exemple en A1), au deuxième, N ! (rangé en A2) et au troisième (M-N) ! (rangé en A3). La dernière étape est le calcul des combinaisons : [A1 / (A2*A3)].
En remplaçant le sous-programme par une fonction, on réduit le tout à une seule expression. Le déroulement de la fonction est identique à celui du sous-programme, mais on élimine les appels et les stockages en mémoire.
Analysons le mode d'appel.
Dans la première ligne de la fonction [FUNCTION FACT(L)] l'argument L apparaît. Comme pour les sous-programmes, son but est de signaler l'existence d'un paramètre. En remplaçant, lors de l'appel, le symbole L par le nom d'une variable quelconque ou par une valeur numérique, on lance le calcul correspondant.
Ainsi, pour calculer M !, il suffit d'appeler la fonction en appliquant M [FACT (M)] à N !.
Outre sa présentation plus complète, le Fortran fournit aussi des fonctions sur une seule ligne. Elles s'apparentent davantage à celles du Basic, mais ont pour limite de n'être utilisables que dans le contexte de leur définition.
Syntaxe de ce type de fonction :

nom (argument) = expression

où :

nom Symbolique de la fonction
arguments Paramètres utilisés dans la définition
expression Description du calcul à effectuer.

Par exemple, le calcul de l'hypoténuse d'un triangle rectangle peut s'effectuer avec :

```
HYPO (A,B) SQRT((A**2) + (B**2))
```

Il suffit, alors, de l'appeler comme pour les autres fonctions :

```
HYPOTEN = HYPO (A,B)
```

Les instructions et les formats d'E/S

En Fortran, la gestion des fonctions d'E/S est très différente de celle du Basic. Dans le Basic standard, il existe un seul périphérique d'entrée : le clavier (nous ne parlerons pas ici des fonctions E/S sur disque), et deux unités de sortie : l'écran et l'imprimante. A chaque unité correspond une instruction et il n'est donc pas nécessaire de préciser l'unité d'E/S concernée. S'il s'agit d'une instruction de sortie, le code PRINT l'adresse à l'écran, le code LPRINT à l'imprimante.

En Fortran, en revanche, il faut préciser l'unité concernée pour chaque opération d'E/S.
Cette différence provient des origines du langage. Le Fortran est né sur de gros ordinateurs qui commandent, en général, plusieurs imprimantes et plusieurs systèmes d'entrée (lecteurs de cartes et de bandes, claviers). Il a conservé ses caractéristiques au cours de son évolution.
Une deuxième différence importante concerne la possibilité, pour le Fortran, de définir un **format** pour les données, tant en entrée qu'en sortie.
Dans les fonctions d'impression du Basic, il n'existe pas une grande marge de manœuvre, sauf à recourir à l'option PRINT USING... qui, dans certaines limites, permet d'employer des formats d'impression définis par l'utilisateur.
Pour les instructions d'entrée, les limites sont encore plus grandes. Le langage n'accepte que des valeurs homogènes avec les variables, sans que l'on puisse définir de format. En fait, dans les micro-ordinateurs, il n'existe aucune raison de disposer de formats d'entrée. La saisie est réalisée uniquement au clavier et l'utilisateur ne tirerait aucun bénéfice de l'existence de formats particuliers. Au contraire, le Fortran prévoit une entrée à partir de divers périphériques, comme par exemple le lecteur de cartes perforées. Il est utile, dans ce cas, de pouvoir disposer d'instructions particulières de lecture qui permettent de reconnaître un champ, en fonction de sa position. Cette souplesse permet de perforer des cartes laissant des blancs entre une valeur et une autre, avec une lisibilité certainement supérieure.
En Fortran, chaque instruction d'E/S doit être complétée par deux paramètres : le code correspondant à l'unité physique d'E/S et un numéro de ligne qui définit le format utilisé en saisie ou en présentation de données.

L'unité doit être précisée dans la mesure où le Fortran ne dispose pas d'instruction différentes pour l'écran et pour l'imprimante. Le format de présentation ou de lecture des données peut être omis et on dira, dans ce cas, que l'instruction est **non formatée.** Les instructions d'E/S, prévues par le Fortran, sont :

READ	pour la lecture
PRINT ET WRITE	pour l'écriture

Les mots READ et WRITE sont employés aussi pour les fonctions d'E/S sur disque, mais cet aspect sera étudié ultérieurement dans la partie traitant de la gestion de fichiers. Le format général des instructions READ et WRITE est :

READ (N,M) liste de variables
WRITE (K,L) liste de variables

N et K indiquant les périphériques, respectivement d'entrée et de sortie : M et L sont les numéros de ligne où se trouve le format de lecture ou d'écriture.
L'instruction :

READ (5,270) A,B,C

signifie : « lire sur le périphérique 5 selon le format défini à la ligne 270 et transférer les valeurs dans les variables A,B,C ».
De même :

WRITE (6,300)A,B,C

signifie : « écrire les valeurs des variables A,B,C, sur le périphérique 6 selon le format défini à la ligne 300 ».

L'instruction PRINT constitue une extension de la version Fortran ANSI 77 qui ne rentre pas dans le cadre du format général du langage. Alors que les formats précédents (READ, WRITE) sont communs à toutes les versions, PRINT n'est pas prévu, par exemple, sur des systèmes plus grands utilisant des compilateurs anciens.
Observons que le n° attribué aux divers périphériques dépend de chaque machine. Normalement, le périphérique d'entrée est numéroté 5 (clavier ou lecteur de cartes) alors que celui de sortie (imprimante) est le 6 ; mais rien n'interdit d'avoir d'autres assignations.
Nous exposerons par la suite les fonctions d'E/S par rapport à leur développement sur les micro-ordinateurs. On laissera donc de côté les options pour le lecteur de cartes qui constitue une unité caractéristique des gros systèmes ; l'idée est cependant simple : il suffit de modifier le code numérique de l'unité.

PRINT

C'est l'instruction de sortie pour le périphérique standard (écran ou imprimante selon la configuration du système). Syntaxe :

PRINT N, données

N	N° de ligne de format de sortie correspondant
données	Valeurs à écrire. Il peut s'agir de variables de constantes, de calculs ou de chaînes.

L'indication du n° de ligne du format peut être omise et les valeurs seront alors écrites selon les indications contenues dans l'instruction elle-même. Examinons quelques exemples :

PRINT 10,A,B,A+B

écrit, selon le format défini à la ligne 10. Les variables A,B et leur somme :

PRINT *, 'somme=', A+B

n'ont pas de format (symbole *). La sortie est alors une chaîne « Somme= » suivie de la valeur A+B.

ASSIGN 10 TO F
PRINT F,A,B,A+B

sont une forme différente du premier exemple. A la ligne 10, où le format est défini, on a attribué le nom symbolique F(ASSIGN 10 TO F). Grâce à ce nom, la fonction peut être appelée à n'importe quel point du programme.

WRITE

L'instruction WRITE est utilisée pour transférer des données soit sur un périphérique de sortie soit sur un disque.
Format général :
WRITE (unité, format, statut, étiquette, enregistrement) données...

où

unité	Unité de sortie
format	Numéro de ligne décrivant le format. Parfois, il peut s'agir d'une définition de format entre guillemets
statut	Message d'erreur ; différent de zéro, le code renvoie à une erreur de programmation en phase d'écriture (les valeurs ne sont pas standard, chaque système disposant de son propre tableau)
étiquette	Numéro de ligne où le contrôle est transféré si erreur
enregistrement	Numéro d'enregistrement pour l'écriture sur des fichiers à accès direct
données	Liste de variables, de constantes,... à écrire

Syntaxiquement, chaque paramètre est précédé d'un mot-clé qui l'identifie :

UNIT	Unité de sortie. Exemple : UNIT=6 sélectionne l'unité 6
FMT	Format. FMT=100 indique le format décrit à la ligne 100
IOSTAT	Assigne la variable de statut. Si IOSTAT=K, au retour le code d'une erreur sera contenu dans la variable K
ERR	Ligne de contrôle en cas d'erreur, ERR=500 transfère le contrôle à la ligne 500
REC	Numéro d'enregistrement. Si REC=5, l'instruction est écrite sur l'enregistrement 5 du fichier (UNIT) n° 6

Si les deux premiers mots-clé (UNIT et FMT) peuvent être omis, les autres sont obligatoires quand les paramètres correspondants existent. Voyons les exemples suivants :

WRITE (3,10) A,B	Ecrit sur l'unité 3 selon le format 10 ; il n'y a pas d'autres paramètres et donc pas de mots-clé.
WRITE (3,10,ERR=100)	Ecrit sur la 3 dans le format 10 ; en cas

Perforatrice de ruban Facit modèle 4070 capable d'imprimer jusqu'à 75 caractères par seconde.

Facit

d'erreur, le contrôle est transféré à la ligne 100.

WRITE (3,10,IOSTAT=I,ERR=100,REC=5)A
Ecrit sur la 3 dans le format 10. La variable I contient l'éventuel code d'erreur et, dans ce cas, le contrôle passe à la ligne 100. Les données (contenu de la variable A) sont écrites dans l'enregistrement 5.

READ

L'instruction READ s'écrit de deux façons selon qu'elle adresse l'unité d'entrée standard ou un fichier. Dans le premier cas, la syntaxe est :

READ M, liste

où M est le numéro de ligne contenant les spécifications de format ; « liste » est une liste de variables où seront mémorisées les valeurs lues. Par exemple :

READ 10,A,B	lit les variables A et B selon le format défini à la ligne 10
READ*,(V(I),I=1,5)	lit, comme un DO implicite, la variable dimensionnée V(I), sans format
READ 20,(V(I),I=1,5)	même définition (DO implicite) avec format de lecture défini à la ligne 20

L'accès aux fichiers avec l'instruction READ fait appel aux mêmes paramètres que WRITE. Le format général se résume ainsi :

READ (unité, format, statut, étiquette, enregistrement) données

L'instruction :

READ (2,1,IOSTAT=KR,ERR=2,REC=6)A,B
lit les variables A et B sur l'enregistrement 6 du fichier 2 selon le format défini à la ligne 1. Le code d'erreur est dans KR, et le cas échéant, le contrôle est transféré à la ligne 2.
Cette instruction comporte en outre un nouveau paramètre qui provoque un saut en cas de fin de fichier EOF (End Of File) sur les fichiers séquentiels. Le paramètre est END=n° ligne.
Par exemple, la ligne :

READ (3,150,END=1000,REC=J)V

lit l'enregistrement J du fichier séquentiel n° 3 selon le format 150 ; en cas d'EOF, le contrôle passe à l'instruction : 1000 (END=1000).

En Fortran, ces opérations sont gérées par l'instruction READ avec l'option END=...
En Basic, par contre, il faut tester EOF pour savoir si le fichier est fini et on peut donc décider, avec un IF, de poursuivre ou non la lecture.

Formats d'E/S

Les fonctions d'E/S du Fortran peuvent faire référence à un numéro de ligne où seront données les spécifications de lecture ou d'écriture des données. Cette ligne particulière (FORMAT codes) contient le format de saisie ou de restitution. Normalement, ce type d'instruction n'est pas vérifié par le compilateur ; les erreurs de format n'apparaissent donc qu'en phase d'exécution du programme.
Voici les spécifications de traitement des variables :

Entiers	I	(INTEGER)
Réels	F	en virgule (fixed-point)
	E,D	en virgule flottante (floating-point)
	G	en virgule fixe et flottante
Caractères	A,R	
Logiques	L	
Octals	K,O	

Formats pour les entiers. La lettre d'identification (I) est suivie du nombre de chiffres qui composent la valeur à lire ou à écrire ; 13, par exemple, indique une valeur entière de 3 chiffres. Le code peut aussi être précédé d'une autre valeur numérique qui a le sens d'un **facteur de répétition** et qui indique le nombre de fois à appliquer le format défini.
Voyons quelques exemples :

READ (6,10)	Lecture de 2 valeurs dans le format 10
WRITE (5,20)I,K,L	Ecriture de 3 valeurs dans le format 20
10 FORMAT (12,15)	Peut être une erreur, car A et B ne sont pas entiers par défaut
20 FORMAT (312)	Dans ce cas, le format entier est valable ; les variables I,K,L sont entières, il faut seulement vérifier que chacune d'elles se compose de 2 chiffres (312=3 fois 12)

Le format peut s'intégrer dans la ligne qui exprime la fonction ; ainsi :

WRITE (5,'(312)') I,K,L

a la même signification que la ligne précédente. Dans certaines versions de Fortran, on prévoit l'extension :

In.m

où :
n indique le nombre maximum de chiffres, m le minimum (en OUTPUT seulement car en instruction d'entrée, m est ignoré). Exemple : avec le format 15.3, la variable 4 s'écrit :

bb004

soit avec 2 espaces pour remplir le champ maximum prévu (longueur 5) et 3 caractères apparents. Puisque la valeur numérique ne comporte qu'un chiffre (4), les chiffres manquants, dans la définition du champ minimum (longueur 3), seront remplacés par des zéros.

Formats pour les réels. La lettre F est obligatoirement suivie de deux valeurs numériques, la première pour la longueur totale du champ en caractères (point décimal compris) et la seconde pour le nombre de chiffres décimaux après la virgule).
Les lettres E et D sont employées pour les représentations en virgule flottante (voir bas de page).
Le code E réserve 2 positions à l'exposant, le code D en réserve 3. La longueur du champ (10 dans le second exemple) comprend 3 positions pour l'exposant (option D) et 1 pour le point décimal : un champ de longueur 7 en format D ne dispose donc que de 3 positions utiles pour la valeur.
Le format E fonctionne exactement comme le format D ; la seule différence est dans l'exposant, auquel on ne réserve que 2 chiffres. Le format G, s'adaptant à la longueur de la valeur, est utilisable pour représenter des valeurs en virgule flottante ou en virgule fixe.

SPECIFICATIONS DE FORMAT POUR LES NOMBRES REELS EN VIRGULE FIXE

Valeur en mémoire	Identification	Sortie
9.7584	F 4.2	9.76
9.7584	F 6.4	9.7584
15	F 4.1	15.0
128.2	F 3.1	Erreur : la valeur n'entre pas dans le champ

SPECIFICATIONS DE FORMAT POUR LES NOMBRES REELS EN VIRGULE FLOTTANTE

Valeur en mémoire	Identification	Sortie
21.412	D 7.4	Erreur : le champ d'une longueur 7 ne peut contenir la donnée
21.412	D 10.4	2.1412+01

Format pour les caractères. C'est la lettre A qui spécifie le format pour l'E/S des caractères prévus. Par exemple, A3 définit une chaîne de 3 caractères.
Si les caractères composant une chaîne sont plus nombreux que prévus par la description, on ne prend en compte que ceux décrits dans le format, en partant de la gauche.
Soit la chaîne ABCDEFG : en sortie, le format A4 générera l'impression de ABCD. De même, ce format employé en lecture n'en transfèrera que les 4 premiers caractères.
Le format R (extension du Fortran ANSI 77) s'apparente au précédent mais avec un cadrage à droite dans le champ d'E/S.

Formats pour les constantes logiques. Une constante logique (TRUE/FALSE) est indiquée par L, suivi d'un nombre représentant la longueur du champ à examiner. Celui-ci doit contenir, comme premier caractère (sauf l'espace), la lettre T (TRUE) pour VRAI ou F (FALSE) pour FAUX.
Par exemple, en entrée, le format L3 examine un champ de 3 caractères et assigne à la variable la valeur TRUE s'il trouve T, et FALSE s'il trouve F. En introduisant une des chaînes :

T,Tx, Txy

le résultat dans la variable est toujours T (VRAIE) ; les lettres qui suivent le code T ne sont pas prises en considération.
De la même manière, les chaînes F, FZ, FKL transfèrent dans la variable la valeur F (FAUX).
En sortie, on obtient la lettre T ou F, précédée d'autant d'espaces que nécessaire pour compléter le champ défini dans le format.
Par exemple, si la variable contient TRUE, le format L4 génère ␣ ␣ ␣ T, alors que le format L1 fournit T.

Descripteurs de mise en page

La gestion complète des fonctions d'E/S nécessite l'emploi de descripteurs spéciaux appelés descripteurs de mise en page. Ces éléments sont employés pour insérer des commentaires, des espaces ou des zéros dans les champs de données et ils se subdivisent en trois catégories :

- descripteurs par fonctions d'entrée
- descripteurs par fonctions de sortie
- descripteurs par fonctions d'entrée et de sortie.

Mise en page en entrée. Il s'agit de BN et de BZ. Avec les descripteurs numériques déjà décrits, les éventuels espaces (blancs) entre les valeurs numériques introduites étaient ignorés ; avec BZ, ils sont convertis en zéros.
Exemple : en lisant la valeur 3␣ 7␣ 5 avec le format 15 on obtient le nombre 375 et les espaces entre les chiffres (␣) seront ignorés. Si on utilise le format BZ, on a 30705 : les espaces sont transformés en zéros.

Mise en page en sortie. Ces descripteurs ont essentiellement deux types de fonctions :

- impression de chaînes de caractères, commentaires, descriptions.
- présentation du signe algébrique.

L'impression de chaînes s'obtient de deux manières : la première consiste à mettre entre guillemets les caractères à écrire, la seconde à les décrire comme des constantes de Hollerith (nH). Par exemple, les formats :

14, 'Total =', 13
14,9 HTotal = , 13

génèrent l'impression d'une valeur numérique (14), suivie du texte Total = et d'une autre valeur numérique (13).
Normalement, le signe + n'est pas explicité, contrairement à ce qui se produit avec les valeurs négatives (signe - toujours indiqué). SP provoque l'impression du signe +.

Descripteurs communs. Les principaux descripteurs employés, aussi bien en entrée qu'en sortie, sont :

nX : saute n positions
Tn : tabule en colonne n
/ : indique la fin d'un enregistrement

Facteur d'échelle. Il est représenté par nP, où n représente un facteur multiplicateur, P étant l'indicateur positif ou négatif. Le facteur d'échelle ne s'applique qu'aux valeurs numériques. Par exemple, avec le format F10.4 la

valeur 135.79817 est écrite comme 135.7981 (4 décimales). En appliquant un facteur d'échelle, par exemple - 2, le format devient : - 2PF10.4 et l'impression 1.357981.

Répétitions de format. N'importe quel format peut être répété autant de fois qu'on le désire et le nombre de répétitions doit précéder le descripteur. Exemple : la spécification 3F5.2 répète 3 fois le format F5.2. Le facteur de répétition s'applique aussi à des formats qui contiennent plusieurs descripteurs ; on utilise alors des parenthèses.
Par exemple, la spécification :

(312,2(14,X,F8.2))

décrit la sortie suivante :

312 Trois valeurs entières de 2 chiffres
2( ) Répétition d'un groupe composé de :
- 14 Une valeur entière de 4 chiffres
- X Un espace
- F8.2 Une valeur réelle avec un champ de 8 caractères et de 2 décimales.

Remarquons que les trois valeurs entières (312) sont écrites de manière contiguë ce qui ne permet pas de les distinguer. De même, les groupes (14,X,F8.2) seront écrits à la suite. Pour séparer les divers champs, on utilise le descripteur X. Le format précédent écrit :

(3(X,12),2(X,14,3X,F8.2))

insère un espace entre les trois premières variables entières (X,12) et un espace entre les groupes 14, etc.

Instructions d'E/S non formatées

Les instructions d'E/S sans formatage ont une syntaxe et une logique semblables aux commandes Basic correspondantes, pour l'écran, l'imprimante et le clavier alors qu'elles sont très différentes pour la gestion du disque.

Instructions d'entrée. En lecture, elles sont analogues à celles décrites pour les entrées avec format, avec en plus le symbole * :

READ *, variables

ou bien :

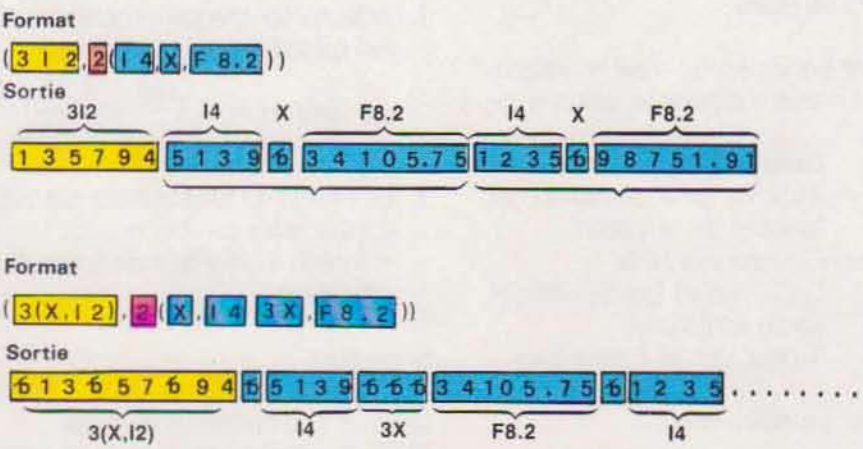
READ (U, *) variable

Dans la première, l'unité adressée (clavier) est implicite. Dans la seconde, c'est une unité d'entrée quelconque codée numérique par (U). En lecture, les valeurs, rangées en mémoire dans les variables, doivent être séparées par un ou plusieurs espaces, par le symbole / (slash) ou par une virgule. Exemple, avec l'instruction :

READ *,A,B,C

les valeurs A=3, B=5, C=11 seront entrées

EXEMPLES DE FORMATS ET SORTIES CORRESPONDANTES



par l'un des 3 formats :

```
3  5  11
3 / 5 / 11
3,5,11
```

Pour attribuer la même valeur numérique à toutes les variables on utilise un facteur de répétition. Soit la valeur 7 assignée aux variables A,B,C, on retrouve :

```
7  7  7
7 / 7 / 7
7,7,7
3*7 (signifie 3 fois la valeur 7)
```

Si besoin est, on peut spécifier l'exposant (avec un des symboles admis : E, D, etc). Par exemple, la donnée 586 se lit aussi : 5.86E2 (E2 signifie × 10^2).
Les espaces blancs ne sont pas pris en considération et la variable conserve sa valeur en fonction de l'instruction READ. Avec :

```
READ *, A, B, C
```

et les valeurs d'entrée :

```
b, 20, b
```

on transfère la valeur 20 dans la variable, B ; alors que A et C gardent leur valeur d'origine.

Instructions de sortie. L'instruction PRINT non formatée s'écrit :

```
PRINT *, variables
PRINT (U, *) variables
```

avec les mêmes significations, mais les valeurs sont écrites de manière différente, selon le type de variable :

Entiers	Comme des nombres entiers
Réels	Avec ou sans exposants, en fonction de la valeur
Double précision	Comme des réels
Complexes	Deux valeurs (partie réelle et partie imaginaire)
Logiques	T pour vrai et F pour faux

Soit les couples variables-valeur :

I=256	entier
R=25.72	réel
D=0.1173D2	double précision
C=(12,5)	complexe
L=.TRUE.	logique
CH='TEXTE'	caractères

Les sorties sont :

Instruction	**Sortie**
PRINT *,I,CH	256 TEXTE
PRINT *,C,L	(12.,5.)T
PRINT *,R,D	25.72 1.173D1

On peut aussi appeler l'instruction WRITE en précisant l'unité de sortie. Si on prend le code 6 pour le périphérique standard (6 est l'assignation la plus commune) les instructions précédentes deviennent :

```
WRITE (6,*)I,CH
WRITE (6,*)C,L
WRITE (6,*)R,D
```

avec le même résultat.

Exemple de programmation en Fortran

L'organigramme ci-contre exécute quelques instructions Fortran.
Le problème à résoudre est le suivant : étant donné un montant initial, investi avec une capitalisation mensuelle, effectuer le calcul mensuel de l'intérêt. Le taux varie selon les mois ; il faut donc prévoir 12 valeurs (une par mois).
Déroulement du calcul :

1 / A la fin de chaque période (mois) l'intérêt est calculé par :

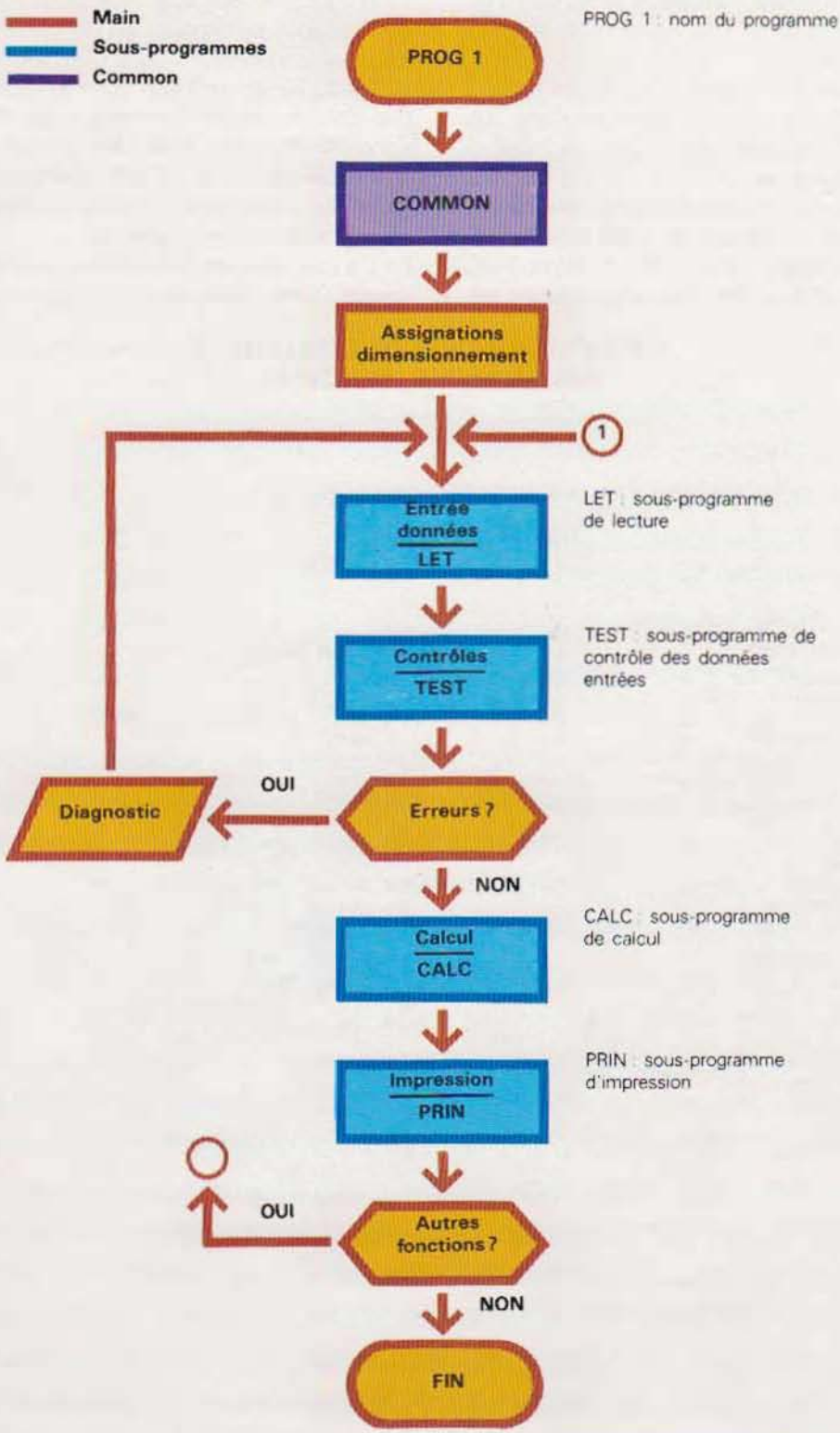
$$\text{Intérêt} = \frac{\text{Capital} \times \text{Taux (mensuel)}}{100}$$

2 / Le capital du mois suivant est calculé par :
Capital mois suivant =
= Intérêt à la fin du mois (point 1) + Capital précédent.

Si on pose :

C(12) = Capital début du mois
S(12) = Intérêt cumulé dans le mois
T(12) = Taux à appliquer pour chaque mois

ORGANIGRAMME POUR LE CALCUL DES INTERETS



pour un mois I, quelconque, on aura les expressions :

S(I)=(C(I)*T(I))/100 (Intérêt à la fin du mois)
C(I+1)=S(I)+C(I) (Capital mois suivant)

Pour effectuer le calcul sur les douze mois, il suffit d'appliquer les deux formules dans une boucle avec I variable entre 1 (premier mois) et 11 ; la dernière valeur (mois 12) est calculée séparément, car elle représente la valeur finale (total annuel) et n'a pas de mois suivant. On trouvera en pages 1185, 1187, 1188 et 1190 les organigrammes des sous-programmes appelés. Les programmes correspondants en Basic et en Fortran sont donnés page 1184, 1186, 1189 et 1191.

Le programme principal ci-dessous met en relief certaines différences fondamentales entre les deux langages. En particulier, les lignes 10 et 20 du Basic sont des commentaires, alors que dans la version Fortran il s'agit d'instructions P indispensables. La première ligne (correspondant à la 10) définit le nom du programme, la deuxième certaines variables communes aux sous-programmes.

En Fortran, ne sont numérotées que les lignes devant être adressées, en particulier les

EXEMPLE DU CALCUL D'INTERETS
PROGRAMME PRINCIPAL

Version Fortran

```
      PROGRAM PROG1
C     ************************
C     ** PROGRAMME PRINCIPAL **
C     ************************
      COMMON C(12), S(12), T(12)
30    CALL INIT
      CALL TEST (KE)
      IF (KE .EQ. 0) GOTO 80
      WRITE (*, 900)
      GOTO 30
80    CALL CALC (TA)
      CALL IMPR (TA)
      WRITE (*, 910)
      READ (*, 920) N
      IF (N .EQ. 1) GOTO 30
C     ** FORMAT DE LECTURE-ECRITURE **
900   FORMAT (3X, 6HERREUR)
910   FORMAT (3X, 'CONTINUE = 1')
920   FORMAT (1X, I1)
      STOP
      END
```

Version Basic

```
 1 REM : PROG1
 5 REM
10 REM : ************************
15 REM : ** PROGRAMME PRINCIPAL **
20 REM : ************************
25 DIM C(12), S(12), T(12)
30 GOSUB 1000
40 GOSUB 2000
50 IF PI - 0 GOTO 80
60 PRINT "ERREUR"
70 GOTO 30
80 GOSUB 3000
90 GOSUB 4000
100 PRINT "CONTINUE = 1"
110 INPUT N
120 IF N = 1 GOTO 30
170 END
```

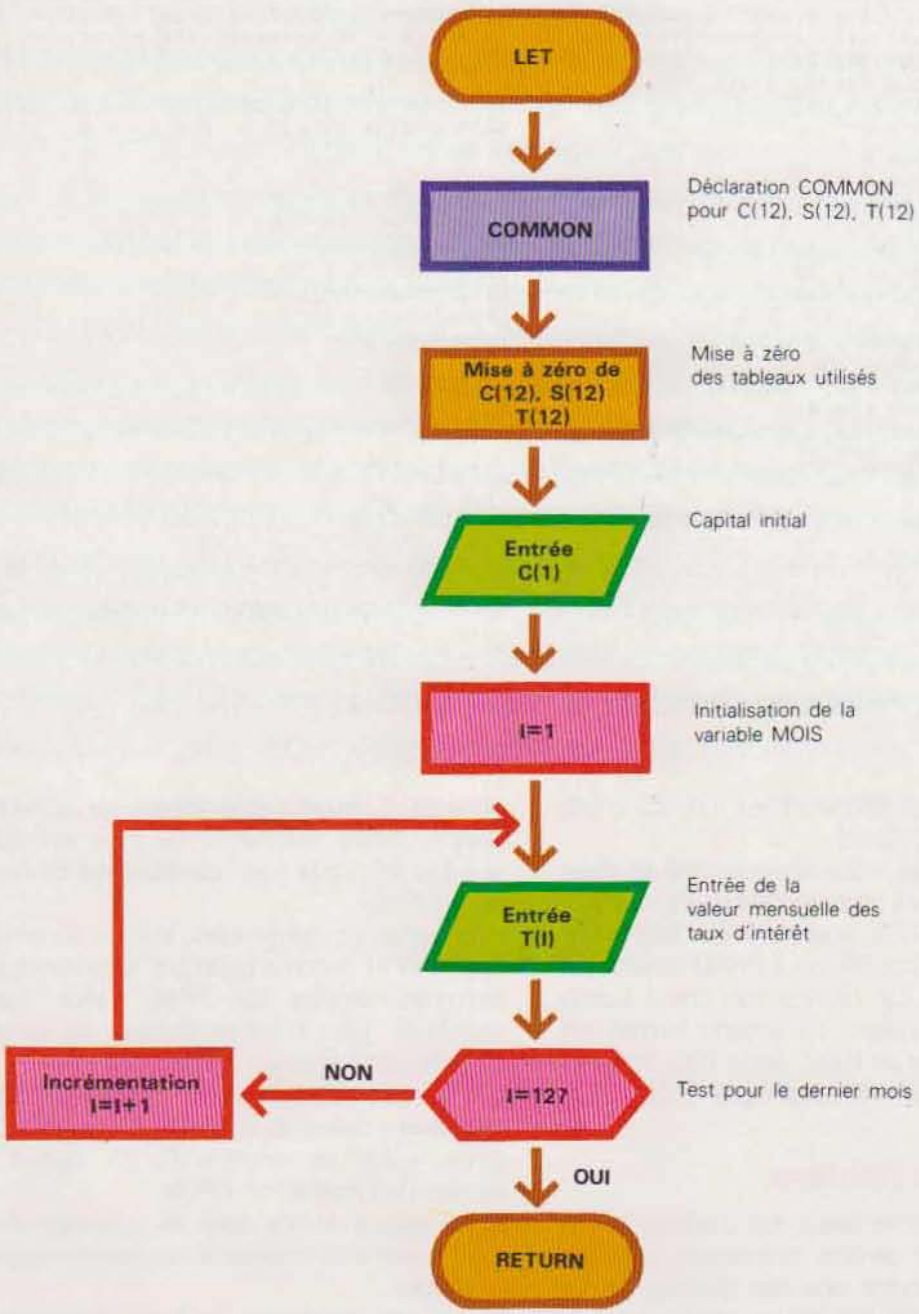
SOUS-PROGRAMME D'ENTREE DES DONNEES

Common
Fonction d'E/S
Boucle

Entrées Aucune

Fonction Mise à zéro des variables
Lecture du capital initial
Lecture du taux d'intérêt
du 12e mois

Sorties Capital initial [C1(1), début-mois 1]
Taux d'intérêt [T(12), 12 tableaux]



SOUS-PROGRAMME D'ENTREE DES DONNEES

Version Fortran

```
C       ******************************
C       ** ROUTINE D'INITIALISATION **
C       ******************************
        SUBROUTINE INIT
        COMMON C(12), S(12), T(12)
        DO 10 I=1, 12
        C(I) = 0
        S(I) = 0
        T(I) = 0
10      CONTINUE
        WRITE (*, 90)
        READ (*, 92) C(1)
        DO 20 I=1, 12
        WRITE (*, 94) I
        READ (*, 96)  T(I)
20      CONTINUE
C       ** FORMAT DE LECTURE/ECRITURE **
90      FORMAT (1X, 'CAPITAL = ')
92      FORMAT (F8.0)
94      FORMAT (1X, 'MOIS = ', 1X, I2)
96      FORMAT (F5.2)
        RETURN
        END
```

Version Basic

```
1000 REM : ************************
1001 REM : ** ROUTINE D'INITIALIS **
1010 REM : ************************

1020 FOR I = 1 TO 12
1030 C(I) = 0:S(I) =  0:T(I) = 0

1040 NEXT I
1050 INPUT "CAPITAL = "; C(1)

1060 FOR I = 1 TO 12
1070 PRINT "MOIS = "; I
1080 INPUT T(I)
1090 NEXT I

1100 RETURN
```

900,910,920 qui décrivent les formats d'E/S (sans équivalent Basic).

Dans cet exemple, nous avons adopté les deux formes possibles d'instructions de sortie : WRITE et PRINT. Si, pour WRITE, il faut préciser l'unité de sortie (6), pour PRINT, cela n'est pas nécessaire car l'instruction prend l'unité standard du système. Le second format est dont très proche du Basic (ligne 100), mais en Fortran, on y fait référence (ligne 910).

Gestion de fichiers

Le Fortran, comme beaucoup d'autres langages conçus pour de gros ordinateurs, considère le disque comme une des diverses unités d'entrée/sortie. En général, toutes les instructions d'E/S peuvent être dirigées sur des unités dont le disque fait partie. Le choix est opéré une fois précisé le code identificateur de l'unité ou du fichier.

Pour certaines d'entre elles, le code numérique est défini et connu a priori par le système (par exemple, emploi de PRINT vers l'unité standard) ; pour d'autres (fichiers sur disque) c'est le programmeur qui doit l'affecter.

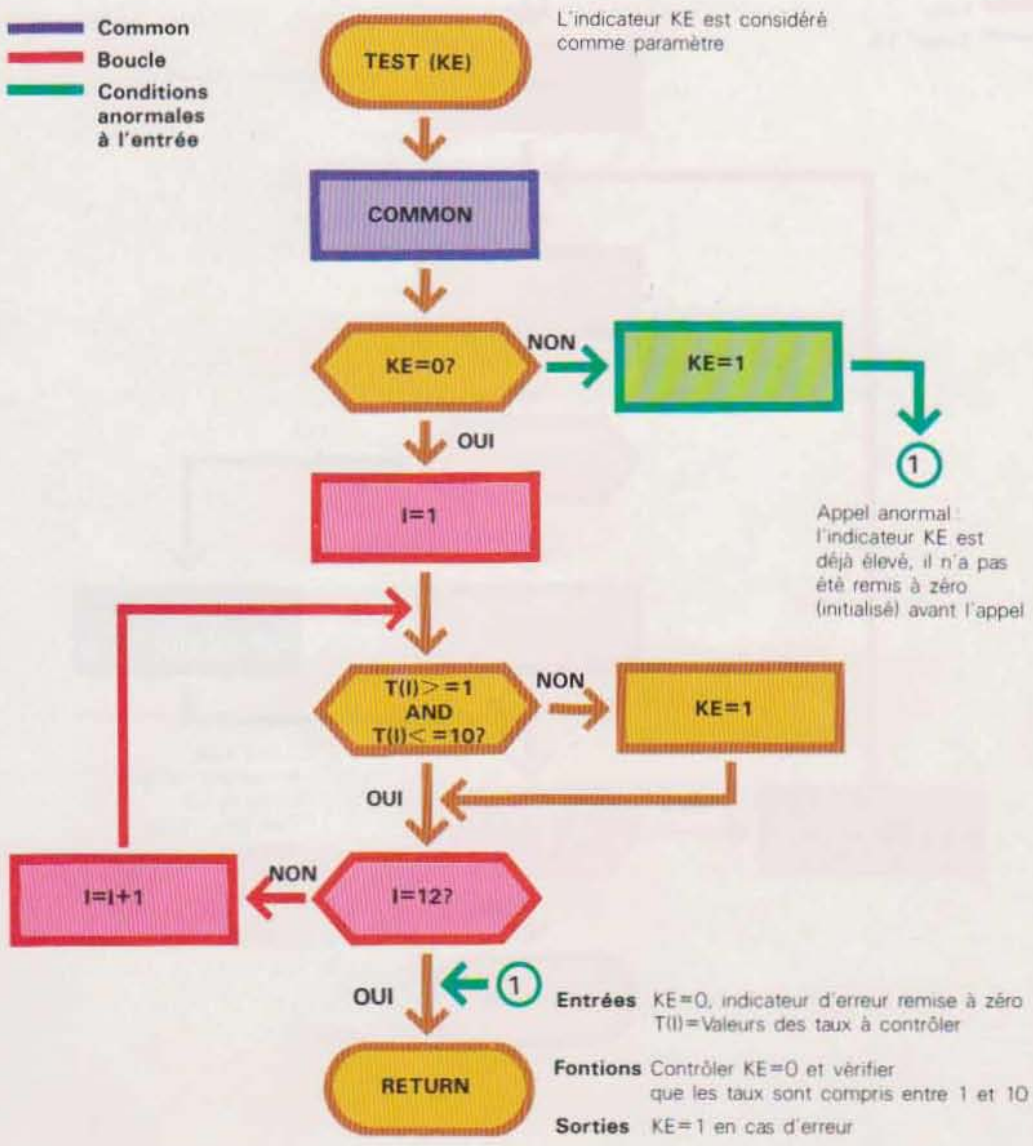
Les unités du premier type s'appellent **préliées :** celles du second type doivent être « liées » par un numéro d'unité logique au moyen de l'instruction OPEN.

Nous nous trouvons donc en présence d'une conception différente dans l'utilisation des périphériques.

En dehors du disque, le Basic standard ne

connaît pas d'autres dispositifs de gestion de fichiers. En Fortran, (comme en Cobol d'ailleurs) ces unités sont aussi considérées comme des fichiers. On peut ainsi effectuer un fichier à l'écran, à l'imprimante ou au clavier et, dans les grands systèmes, à l'unité bande. Chacune d'elles doit être activée avec des ordres et des instructions explicites, à moins d'être effectuée directement par le système.
De plus, le Fortran prévoit un type de fichier particulier, appelé **fichier interne,** composé d'une zone mémoire à laquelle on fait référence comme s'il s'agissait d'un quelconque fichier « externe », sur disque, par exemple. Chaque variable de cette zone de mémoire, ou chaque élément d'un tableau défini comme partie d'un fichier interne) est considéré comme un enregistrement.
Pour effectuer les opérations d'E/S il faut remplacer, dans les instructions respectives, le nom symbolique de la variable ou du tableau, par le numéro d'unité logique (fichier).

SOUS-PROGRAMME DE CONTROLE

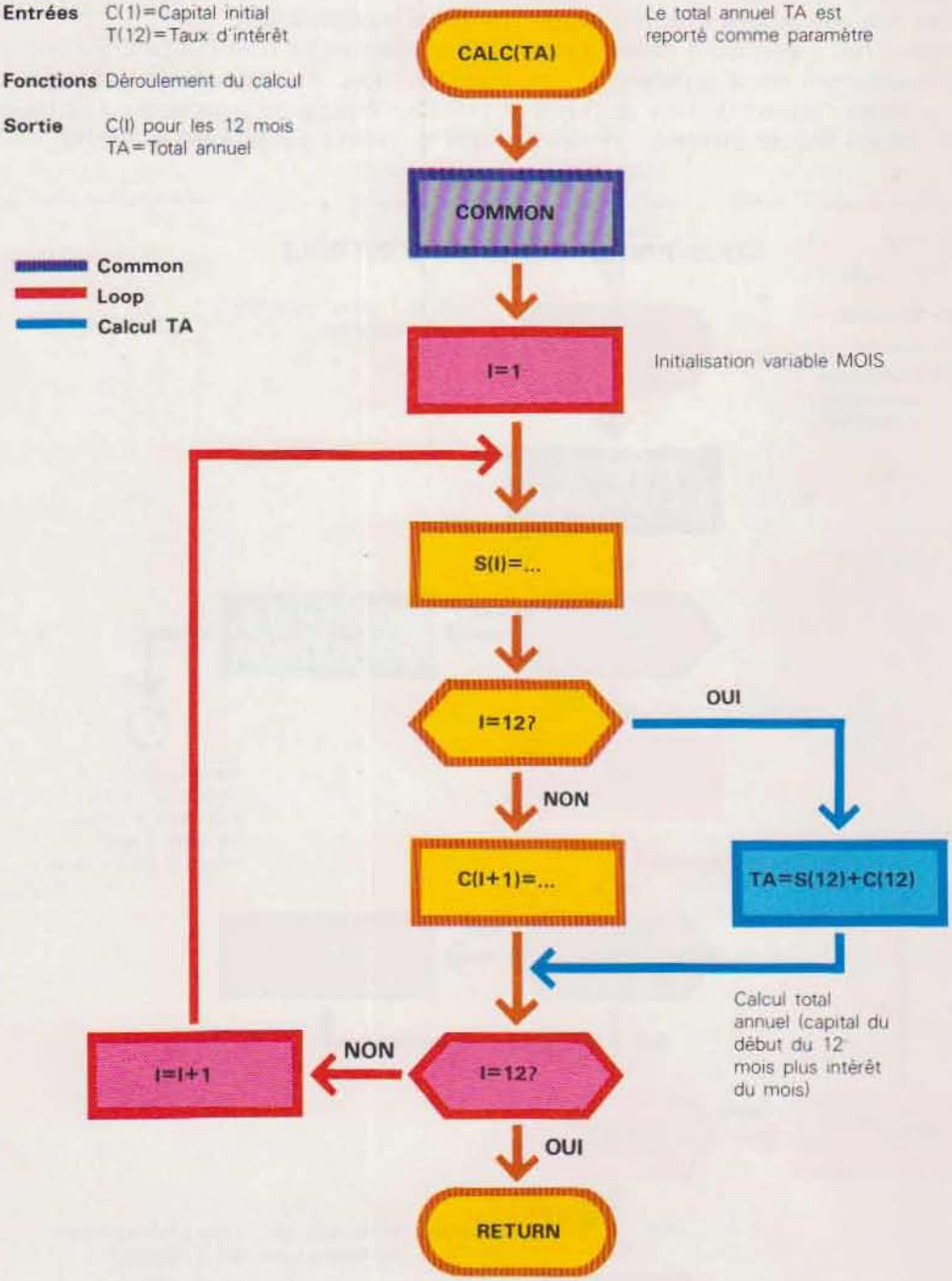


SOUS-PROGRAMME DE CALCUL

Entrées C(1)=Capital initial
T(12)=Taux d'intérêt

Fonctions Déroulement du calcul

Sortie C(I) pour les 12 mois
TA=Total annuel



SOUS-PROGRAMME DE CONTROLE ET DE CALCUL

Version Fortran

```
C     ********************************
C     ** ROUTINE DE VERIFICATION **
C     ********************************
      SUBROUTINE TEST (KE)
      COMMON C(12), S(12), T(12)
      KE = 0
      DO 10 I=1, 12
      IF (T(I) .LT. 1) KE = 1
      IF (T(I) .GT. 10) KE = 1
10    CONTINUE
      RETURN
      END

C     ********************************
C     ** ROUTINE DE CALCUL **
C     ********************************
      SUBROUTINE CALC (TA)
      COMMON C(12), S(12), T(12)
      DO 10 I=1, 12
      S(I) = (C(I) + T(I)) / 100.
      IF (I .LO. 12) GOTO 10
      C(I+1) = S(I) + C(I)
10    CONTINUE
      TA = S(12) + C(12)
      RETURN
      END
```

Version Basic

```
2000 REM : ********************************
2005 REM : ** ROUTINE VERIFICAT **
2010 REM : ********************************
2020 KE = 0
2030 FOR I = 1 TO 12
2040 IF T(I) < 1 THEN KE = 1
2050 IF T(I) > 10 THEN KE = 1
2060 NEXT I
2070 RETURN

3000 REM : ********************************
3005 REM : ** ROUTINE DE CALCUL **
3010 REM : ********************************
3020 FOR I = 1 TO 12
3030 S(I) = (C(I) + T(I)) / 100
3040 IF C(I) = 12 GOTO 3060
3050 C(I+1) = S(I) + C(I)
3060 NEXT I
3070 TA = S(12) + C(12)
3080 RETURN
```

Par exemple, l'instruction :

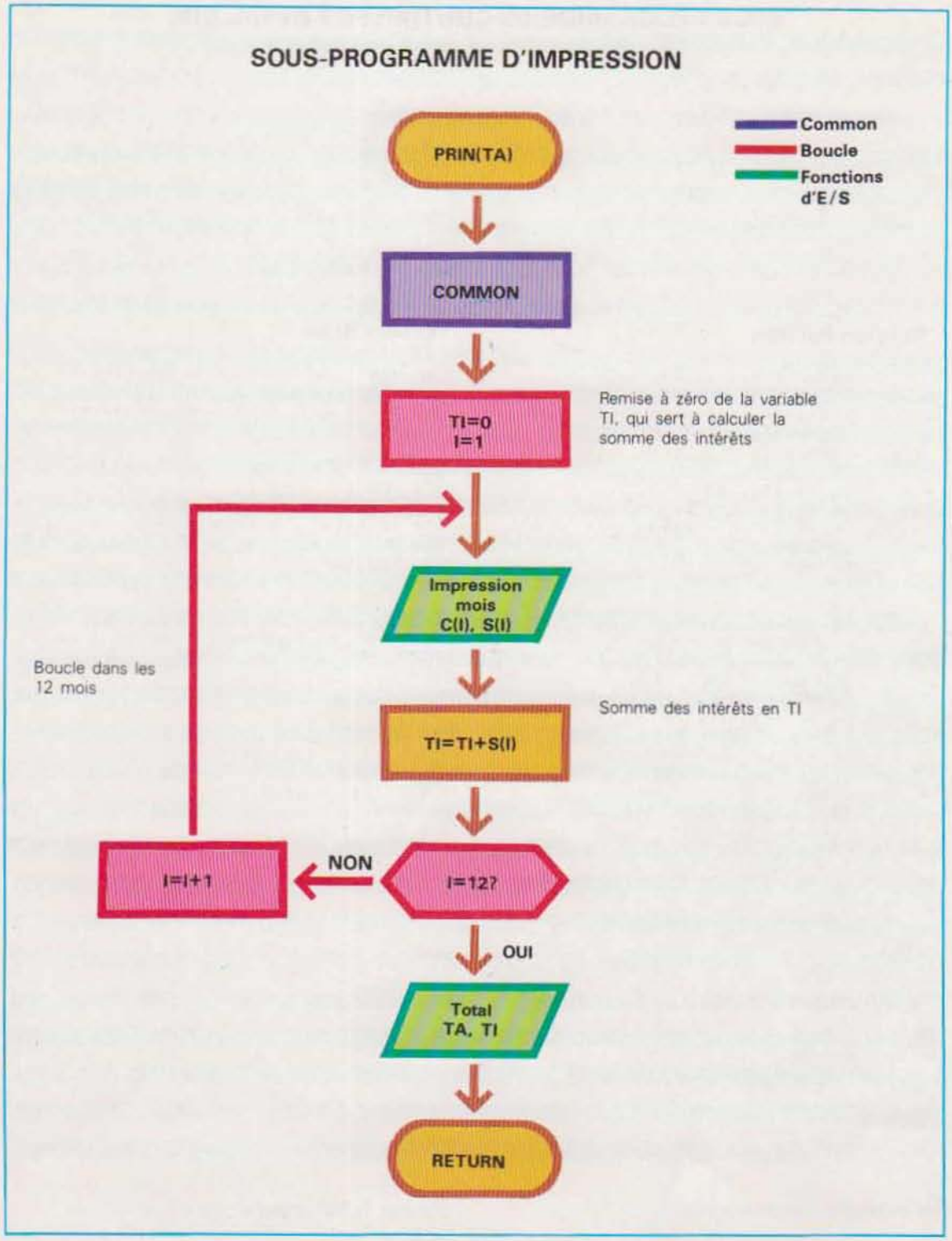
WRITE (6,30)...

se réfère, dans son usage normal, à l'unité 6 (avec le format 30). Pour l'adresser comme un fichier interne, il faut remplacer la valeur 6 par le nom de la variable (ou du tableau) considérée comme fichier interne.

La ligne :

WRITE (ESSAI,30)V

transfère le contenu de la variable V selon le format défini à la ligne 30, dans la variable ESSAI.



Gestion des fichiers sur disque

De nombreuses instructions prévues pour le disque restent également valables pour tous les autres fichiers. Par la suite, nous ferons référence implicitement à l'unité disque, car il s'agit de l'unité la plus importante quand on parle de fichiers. Le Fortran gère deux types de fichiers : ceux qui sont à accès séquentiel et

SOUS-PROGRAMME D'IMPRESSION

Version Fortran

```
C       ****************************
C       ** ROUTINE D'IMPRESSION **
C       ****************************
        SUBROUTINE IMPR (TA)
        COMMON C(12), S(12), T(12)
        TI = 0.
        DO 20 I=1, 12
        WRITE (*, 30) I, C(I), S(I)

        TI = TI + S(I)
20      CONTINUE
        WRITE (*, *) TA, TI
C       ** FORMAT D'IMPRESSION **
30      FORMAT (1X,'MOIS =', 1X,I2,
                8X,'CAPITAL =' F9.1,
                3X,'INTERET =' F8.1)
        RETURN
        END
```

Version Basic

```
' 4000 REM : ***************************
' 4005 REM : ** ROUTINE IMPRESSION **
' 4010 REM : ***************************
'
'
' 4020 TI = 0
' 4030 FOR I = 1 TO 12
' 4040 PRINT "MOIS ="; I,
'            "CAPITAL ="; C(I),
'            "INTERET ="; S(I)
' 4050 TI = TI + S(I)
' 4060 NEXT I
' 4070 PRINT TA, TI
'
'
'
'
' 4080 RETURN
'
```

ceux qui sont à accès direct, au sens donné à ces expressions en Basic.
Trois types d'enregistrements sont prévus :

- Enregistrements formatés — Ils sont écrits et lus selon des formats précis
- Enregistrements non formatés — Ils contiennent une série de données telle qu'elle se présente en mémoire
- Enregistrement Fin-de-fichier (EOF) — Il ne contient pas de données et indique la fin d'un fichier séquentiel. Il est noté par l'instruction ENDFILE.

Principales instructions liées aux fichiers :

OPEN	Relier à un fichier (ouverture)
READ	Lire données
WRITE	Ecrire données
BACKPACE	Déplacer le pointeur d'un fichier séquentiel d'un enregistrement vers l'arrière
REWIND	Placer le pointeur d'un fichier séquentiel sur le premier enregistrement
ENDFILE	Ecrire l'enregistrement End-Of-File, qui marque la fin d'un fichier séquentiel
INQUIRE	Fournir des informations relatives à un fichier
CLOSE	Fermer un fichier

OPEN. Cette instruction « relie » le fichier et le système en lui affectant un numéro d'unité logique.
Elle contient quelques définitions des caractéristiques du fichier, semblables à celles employées en Basic.

Les paramètres à fournir sont :

UNIT	numéro d'unité logique associé. Si on pose UNIT=3 le fichier est reconnu comme unité 3

IOSTAT Spécifie la variable (entière) où il doit transférer des codes d'erreurs.

ERR Numéro de ligne où est réalisé le transfert de contrôle en cas d'erreur : ERR= 130 signifie « GO TO 130 si erreur »

FILE Spécifie le nom du fichier. S'il n'existe pas, il est créé par le système lorsqu'il exécute OPEN.

STATUS Vérifie l'existence du fichier. Valeurs admises :

STATUS='OLD' Le fichier doit exister ; sinon erreur

STATUS='NEW' : le fichier ne doit pas exister ; sinon erreur

STATUS='SCRATCH' : On ne doit fournir aucun nom. Un fichier de travail vient d'être créé

STATUS='UNKNOWN' lance la recherche ; si le fichier n'existe pas, il est créé

Les deux premières options (OLD, NEW) servent à éviter les erreurs suivantes : ouvrir un fichier déjà existant ou qui ne doit pas être présent ; la dernière (UNKNOWN) gère automatiquement la sélection et relie le fichier défini, s'il existe, ou le crée (puis le relie) s'il n'existe pas. UNKNOWN est affectée par défaut.

ACCESS Moyen d'accès ; il peut être de divers types :

DIRECT Accès direct ; dans les E/S suivantes il faut spécifier le numéro d'enregistrement (REC=...)

SEQUENTIAL Fichier séquentiel ; c'est le type choisi par défaut

FORM Précise si les données doivent ou non être formatées ; elle prend la valeur FORMATTED ou UNFORMATTED

RECL Longueur des enregistrements en octets par accès direct

BLANK Peut prendre les valeurs NULL ou ZERO. Dans le premier cas, d'éventuels espaces « blanc » entre les valeurs numériques sont ignorés, dans le deuxième, ils sont convertis en zéros. Par exemple 15b79 devient avec l'option NULL, 1579 ; et 15079 avec l'option ZERO. Par défaut le système opte pour NULL.

Tableau de contrôle de la Fairchild.

Marka

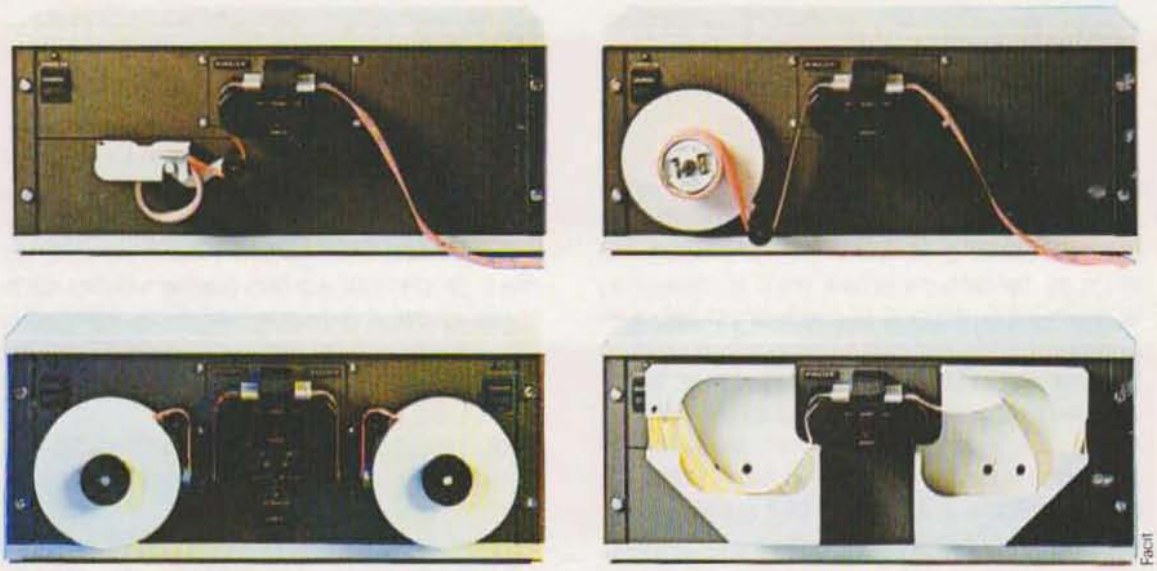
Un exemple d'emploi des spécifications associées à OPEN pourrait être le suivant :

```
OPEN (2,IOSTAT=K,ERR=1,FILE='F1')
```

définit le fichier F1 comme un fichier séquentiel (par défaut) relié à l'unité logique 2. K est la variable d'erreur et 1 le numéro de ligne pour la gestion des conditions d'erreur.
La syntaxe complète pour définir l'unité logique est UNIT=2, mais l'option UNIT peut être omise. La ligne :

```
OPEN (2,FILE='F1',ACCESS='DIRECT',
RECL=1)
```

ouvre le fichier F1 à accès direct, avec des enregistrements d'un octet de longueur, sur l'unité logique 2. La ligne :



Modèles de lecteurs de bandes perforées Facit 4030. De gauche à droite et de haut en bas : modèle pour la lecture d'une bande de 20 m de long (l'équivalent de 8 000 caractères, de 50 m (20 000 caractères), 100 m (40 000 caractères). En bas à droite, lecteur de bandes pliées.

```
OPEN (2,FILE='TEST',STATUS='OLD'
ACCESS='DIRECT,RECL=100)
```

assure les mêmes fonctions que l'instruction précédente, mais elle vérifie que le fichier existe déjà (STATUS='OLD'). Si tel n'est pas le cas, elle génère un message d'erreur. L'autre forme aurait, en revanche, créé le fichier en cas d'erreur ; en effet, par défaut, on a STATUS = 'UNKNOWN'.

READ ET WRITE. Les instructions READ et WRITE, appliquées à des fichiers sur disque, ont des formats identiques à ceux des autres périphériques.
Après l'ouverture du fichier et l'affectation d'un numéro d'unité logique, les opérations d'E/S sont identiques à celles des autres périphériques.
Naturellement, pour les fichiers directs, il faut préciser le numéro d'enregistrement à employer. Par exemple, la ligne :

```
READ  (2,130,REC=12)variable1,variable2,...
```

lit, selon le format 130, l'enregistrement numéro 12 du fichier numéro 2 et transfère les valeurs dans les variables spécifiées. De la même manière, en écriture, avec la ligne :

```
WRITE (2,130,REC=7)variables
```

Les valeurs des variables sont transférées dans l'enregistrement n° 7.

BACKSPACE. Dans les fichiers séquentiels, à la suite de chaque opération d'E/S, un pointeur (géré par le système) sera incrémenté ; il indique le numéro du premier enregistrement disponible. L'opération suivante traite l'enregistrement suivant, et ainsi de suite. L'instruction BACKSPACE décrémente ce pointeur (-1), en le positionnant à nouveau sur le dernier enregistrement traité. La syntaxe de l'instruction est la suivante :

```
BACKSPACE n
```

avec n numéro du fichier.

REWIND. Positionne le pointeur au début du fichier. Les paramètres sont UNIT, IOSTAT ERR, chacun avec la signification décrite dans OPEN. Par exemple, la ligne :

```
REWIND(2,IOSTAT=K,ERR=100)
```

positionne le pointeur au début du fichier 2 ; la variable d'erreur est K, et en cas d'erreur, le contrôle passe à l'instruction 100.

ENDFILE. Ecrit un enregistrement à la fin du

fichier ; le numéro d'unité est le seul paramètre. La ligne :

ENDFILE 2

ferme le fichier précédemment ouvert avec le numéro 2. Les instructions REWIND et ENDFILE, exécutées dans cet ordre, provoquent la perte du contenu du fichier, dans la mesure où la première postionne le pointeur au début et que la seconde marque cette position comme Fin-de-Fichier (EOF). Le résultat est un fichier avec un seul enregistrement contenant la fin du fichier. Les données qui suivent ne sont plus accessibles, car le système traite le fichier de manière séquentielle, trouve EOF sur le premier enregistrement et s'arrête.

INQUIRE. Fournit les caractéristiques d'un fichier. Les paramètres sont ceux employés pour OPEN, avec des valeurs dépendant de l'état du fichier. Par exemple, l'instruction :

```
INQUIRE (FILE='ESSAI',OPENED=K,
NUMBER=N,ACCESS=A)
```

restituera :

K = TRUE si le fichier est ouvert (sinon FALSE)
N = numéro d'unité logique associée
A = Moyen d'accès

CLOSE. Ferme le fichier spécifié et le déconnecte du numéro d'unité logique. Les paramètres prévus sont :

UNIT	Numéro de l'unité
IOSTAT	Etat d'erreur
ERR	Numéro de ligne pour les erreurs
STATUS	Prend deux valeurs : KEEP (fermeture fichier), ou DELETE (fichier écrasé). KEEP est assumé par défaut.

Dans l'ordinogramme ci-contre, le programme lit un fichier séquentiel et calcule la moyenne des valeurs (numériques) qu'il contient.
Chaque valeur (variable V) occupe un enregistrement composé d'un réel (trois chiffres entiers) et d'un chiffre décimal (format F5.1).
Dans l'ordinogramme, apparaît le contrôle de fin de fichier (qui n'existe pas sur le listing de la page 1194, car il est implicite dans l'instruction READ, ligne 100) avec l'option END = 200.
Le fichier à ouvrir s'appelle ESSAI et il est associé à l'unité logique 1. Observons la ligne 100. Pour le calcul de la moyenne des valeurs on ne peut utiliser le nom MOYENNE car il commence par une des lettres qui définissent les variables entières (I,J,K,L,M,N) ; dans ce cas, en effet, on perdrait les éventuelles valeurs décimales. Comme on le constate sur le listing, des formats peuvent se trouver à un endroit quelconque du programme, à condition que ce soit avant END.
Cette instruction doit être la dernière, car elle agit comme un signal de fin pour le compilateur.
De plus, en Basic, le contrôle d'EOF ne peut s'effectuer dans l'instruction de lecture même ; il faut donc le prévoir séparément.

Instructions particulières au Fortran

A ce groupe, appartiennent des instructions propres aux formes de programmation les plus évoluées. Elles ne seront mentionnées qu'à titre indicatif car leur existence est liée au type de machine et de compilateur employés.
Les principales sont :

EXTERNAL
ENTRY
SAVE
BLOCK DATA

EXTERNAL

Définit un nom de sous-programme et l'utilise comme argument dans un appel.

La syntaxe est la suivante :

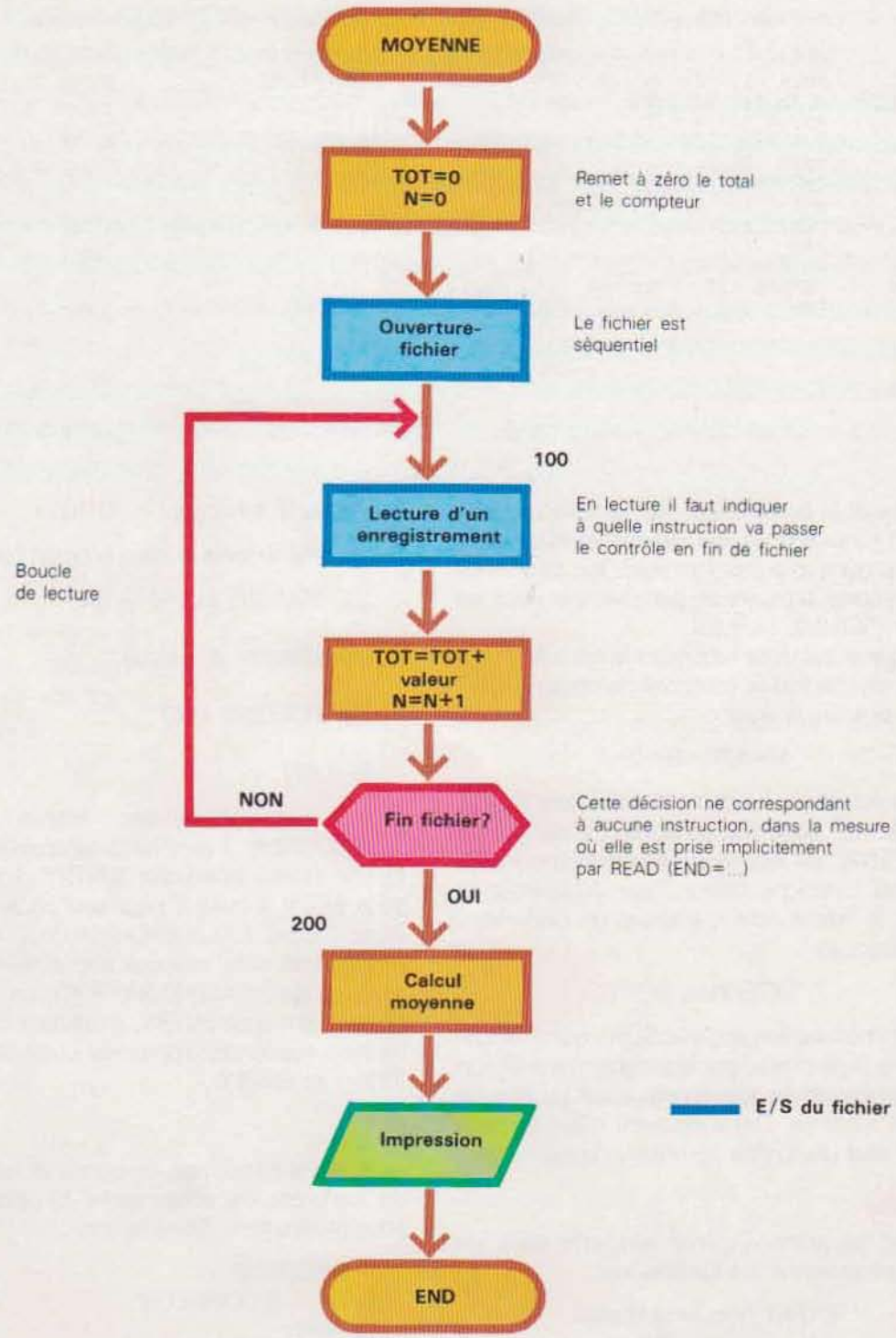
EXTERNAL nom 1, nom 2, etc.

Où nom 1, nom 2... sont les noms des sous-programmes à transférer.

Par exemple, les instructions :

```
EXTERNAL A
CALL SOMME (A,B,C)
SUBROUTINE SOMME (A,B,C)
X = A (N) + B + C
RETURN
```

CALCUL DE LA VALEUR MOYENNE DES DONNEES CONTENUES DANS UN FICHIER



LECTURE D'UN FICHIER ET CALCUL DE LA MOYENNE DES VALEURS

```
      PROGRAM MEDIA
      TOT = 0
      N = 0
      OPEN (1, FILE = 'ENTREE', ACCESS = 'SEQUENTIAL')
100   READ (1, 110, END = 200) V
      TOT = TOT + V
      N = N + 1
      GOTO 100
200   RMOYENNE = TOT / N
      WRITE (6, 210) RMOYENNE
      CLOSE (1)
      STOP
110   FORMAT (F5.1)
210   FORMAT (1X, 'MOYENNE = ', F9.1)
      END
```

informent le compilateur que le nom symbolique A n'indique pas une variable, mais un autre sous-programme (ou fonction), qui peut avoir ses propres arguments, par exemple dans un appel [SOMME (A,B,C)].
Lorsque le système rencontre le nom A, il utilise la routine (ou la fonction) correspondante.
Par exemple, la ligne :

X=A(N)+B+C

utilise A comme fonction du paramètre N alors que B et C sont des variables. La déclaration EXTERNAL est également employée pour définir des fonctions définies par l'utilisateur et ayant le même nom que celles du système.
Par exemple :

EXTERNAL SQRT

décrit une fonction appelée SQRT qui effectuera les calculs prévus par le programmeur et non une racine carrée telle qu'elle aurait pu être lue par le système. Cette dernière n'est évidemment plus disponible au niveau du système.

ENTRY

Définit un point d'entrée alternatif dans un sous-programme. La syntaxe est :

ENTRY nom (arguments)

où nom est le nom symbolique de cette nouvelle entrée (entry-point) et arguments les grandeurs à échanger.

Par exemple, dans le sous-programme :

```
SUBROUTINE ESSAI (A,B,C)
. . . . . .
(déroulement du calcul)
. . . . . .
ENTRY ESSAI 1 (C)
. . . . . .
RETURN
```

on a bien prévu une entrée normale (SUBROUTINE...) avec les 3 paramètres A,B,C et une entrée alternative (ENTRY...) qui s'appelle ESSAI 1 avec C pour seul paramètre.
Avec l'appel CALL ESSAI (A,B,C), le sous-programme sera exécuté depuis son début, alors qu'avec CALL ESSAI 1 (C), on passe le contrôle à la ligne ENTRY, en sautant toutes les lignes précédentes, comprises entre SUBROUTINE... et ENTRY...

SAVE

Avec cette instruction, on conserve les valeurs de certaines variables après le retour d'un sous-programme. Dans la liste :

```
SUBROUTINE xy
SAVE A,B/DONNEES/
RETURN
SUBROUTINE Z
SAVE
```

La première instruction (SAVE A,B...) préserve les valeurs des variables A et B dans le bloc DONNEES, la seconde (SAVE) conserve les valeurs de toutes les variables utilisées dans le sous-programme Z.

BLOCKDATA

En Fortran, on peut appeler des sous-programmes spéciaux qui ont pour seule fonction de définir et d'initialiser des variables intégrées à un COMMON étiqueté (labelled common).
Outre les instructions de déclaration, ces sous-programmes ne peuvent contenir que des instructions d'initialisation. Par exemple :

```
BLOCKDATA UN
COMMON/B/V (5), C,X
DATA V (5*3.1)
END
```

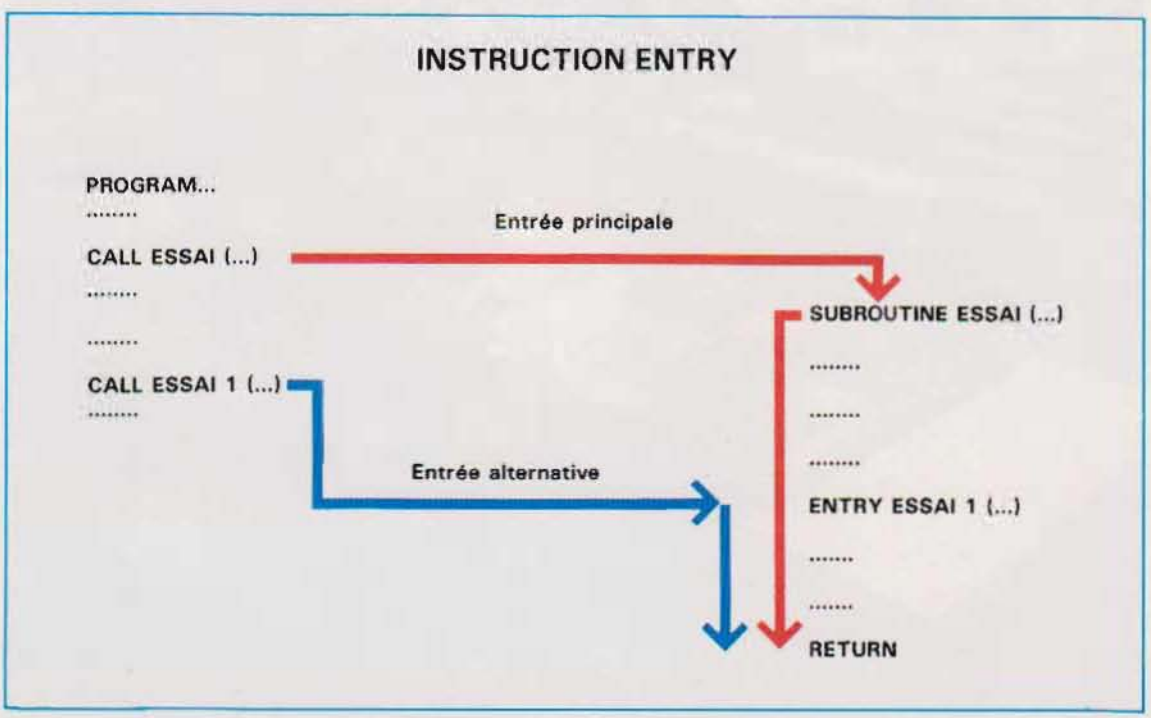
La routine, qui s'appelle UN, définit un zone common de nom B, à laquelle appartiennent les variables V(5), C et X ; de plus, cette routine initialise le tableau V(5), avec des valeurs égales, soit 3.1.

Améliorations spéciales

Par rapport au Fortran standard, certains compilateurs comportent des améliorations obéissant aux normes MIL-STD-1753 (Military Standard) et qui permettent d'agir directement sur les bits de chaque variable.

SHIFT	Déplace les bits d'un nombre déterminé de positions
EXTRACTION	Prélève un certain nombre de bits d'une position de mémoire, (extraction partielle)
TEST	Effectue un contrôle d'état de chaque bit (ON/OFF)
SET	Place n'importe quel bit sur ON
CLEAR	Met en OFF (zéro) le bit indiqué
MOVE	Déplace un ou plusieurs bits
PRODUIT LOGIQUE	fonction AND sur les bits d'un mot (ET)
SOMME LOGIQUE	fonction OR (OU)
OU EXCLUSIF	fonction XOR (OUX)
COMPLEMENT	fonction NOT (NON)

Ces fonctions, qui sont caractéristiques des compilateurs les plus avancés, sont très



utilisées en gestion de périphériques, en particulier des servomécanismes et des appareils d'acquisition de données. Elle confèrent, au Fortran, à la fois une souplesse et une variété d'emploi équivalente sinon supérieure au Basic, la seule différence restant dans la nécessité de compiler, ce qui rend plus difficile le développement et la mise au point de programmes.

Format normal des instructions

La syntaxe des instructions de n'importe quel langage peut se représenter graphiquement, sous forme de description synthétique de ses principales caractéristiques.
Cette représentation, appelée **Railroad Normal Form** (RNF) n'est pas très pédagogique, surtout pour un néophyte. En revanche, elle est très utile comme aide-mémoire pour ceux qui ont une connaissance même superficielle des instructions. En principe, la représentation en format normal s'applique à tout langage symbolique. Elle consiste à illustrer graphiquement les alternatives pour atteindre un certain but ou pour définir une fonction.
Le seul symbole particulier est représenté par un cercle associé à un chiffre. Il signifie que l'instruction correspondante doit figurer un nombre de fois au moins égal au chiffre du cercle.
Les instructions indiquant un programme ou une de ses parties sont :

— Main ou programme principal (PROGRAM) ; l'instruction apparaît au moins une fois (elle est signalée par le symbole I)
— Function (fonction)
— Subroutine (sous-programme)
— Block (bloc de données)

Elles ont toutes des significations analogues et seront donc représentées de manière « parallèle » dans la RNF.
Un programme principal contient des instructions (program statement) et les fonctions function statement... Des étiquettes (n° de ligne) leur sont affectées et l'ensemble doit s'achever avec le mot END.

Terminal d'un Hewlett-Packard.

D. Bonecchi