manuel de référence

COBOL

Tout ce que vous n'auriez jamais su si vous ne l'aviez pas lu

daniel tischer

laurent crivello

année 1992-1993

Nous remercions Malika Saulnier ainsi que M. Rosenthal pour les corrections apportées à cet ouvrage.
Merci également à nos parents qui nous ont beaucoup soutenus et à qui nous dédions ce livre.

Table des matières

Table des matières i

Introduction v

Comment utiliser ce manuel ... v
Organisation ... v
Conventions typographiques ... v

Présentation du langage 1

Eléments du langage ... 1
Jeu de caractères utilisable ... 1
Mots réservés ... 1
mots-clés (keywords) ... 1
mots facultatifs (optional words) ... 1
constantes figuratives ... 1
noms du constructeur ... 1
Noms de variables et de procédures (mots utilisateur) ... 2
Littéraux ... 2
Littéraux numériques (constantes numériques) ... 2
Littéraux alphanumériques (chaîne de caractères) ... 3
Conventions de codage ... 3
Règles ... 3
Suggestions ... 4
Indentation ... 4
Procedure Division ... 4
Comment rédiger ... 4
Modèles d'écriture ... 4
Structure du COBOL ... 6
Structure générale ... 6
SENTENCE ... 6
PARAGRAPH ... 6
SECTION ... 6
DIVISION ... 6
Les divisions ... 6
Identification (IDENTIFICATION DIVISION) ... 6
Description des périphériques (ENVIRONMENT DIVISION) ... 7
Description des données (DATA DIVISION) ... 7
Partie des traitements (PROCEDURE DIVISION) ... 8
Description de données en WORKING-STORAGE SECTION ... 8
Définition et description des champs de données ... 8
Les différentes classes de données ... 10
1. données numériques ... 10
2. données alphabétiques ... 10
3. données alphanumériques ... 11
4. champs numériques édités ... 11
5. champs alphanumériques édités ... 11
Affectations ... 11
Niveau 66 : Renommer et redéfinir - Ambiguïtés ... 12
RENAMES ... 12
REDEFINES ... 13

Niveau 88 : Noms-condition ... 13
Elargissement de la description de données ... 15
FILLER ... 15
VALUE ... 16
BLANK WHEN ZERO ... 16
JUSTIFIED-RIGHT ... 17
SIGN ... 17
USAGE ... 18

Programmation COBOL 19

Fonctions d'entrées-sorties ... 19
ACCEPT ... 19
DISPLAY ... 19
Fonctions d'affectation ... 20
MOVE ... 20
MOVE CORRESPONDING ... 21
INITIALIZE ... 22
Instructions de contrôle ... 22
COPY ... 22
STOP RUN ... 23
Instructions de saut ... 23
GO TO ... 23
GO TO DEPENDING ON ... 23
Opérations arithmétiques ... 24
Règles générales ... 24
ADD ... 25
SUBTRACT ... 25
MULTIPLY ... 26
DIVIDE ... 26
COMPUTE ... 27
Opérations sur les chaînes de caractères ... 28
EXAMINE ... 28
EXAMINE TALLYING ... 28
EXAMINE REPLACING ... 28
EXAMINE TALLYING REPLACING ... 29
STRING ... 29
UNSTRING ... 30
TRANSFORM ... 31
La condition ... 32
IF ... 32
Opérateurs de comparaison : `IS [NOT] = < >` ... 32
Conditions de signe : `IS [NOT] NEGATIVE POSITIVE ZERO` ... 33
Conditions de classe : `IS [NOT] NUMERIC ALPHABETIC` ... 33
Les noms-condition : Niveau 88 ... 33
Appel de procédures ... 34
Définition ... 34
PERFORM ... 34
PERFORM TIMES ... 34
PERFORM UNTIL ... 35
PERFORM VARYING ... 36
EXIT ... 37
Récapitulation ... 37
Modules ... 42
CALL ... 42

EXIT PROGRAM 42

Le traitement des erreurs 45

Aide à la mise au point 45
WITH DEBUGGING MODE 45
DECLARATIVES 45
READY / RESET TRACE 46
163 : Illegal character in numeric field 46
Signification 46
Solutions 46

Les tableaux 47

Tableaux et dimensions 47
Introduction 47
Définition de tableaux 47
OCCURS 47
Les indices 48
Indexation de tableaux 50
Définition des index 50
Traitement des index (SET) 51
Index relatifs 51
Recherche dans un tableau (SEARCH) 52
Différences entre indices et index 53

Les fichiers 55

Organisation 55
Les différentes formes d'organisation 55
Séquentielle 55
Séquentielle indexée 55
Directe, relative ou sélective 55
Chaînée 55
Accès et modifications 55
Accès 55
Accès séquentiel 55
Accès dichotomique 56
Accès direct 56
Modifications 57
Mise à jour de valeurs 57
Ajout 57
Suppression 57
Dégénérescence de l'organisation par modifications 59
Définition de fichiers 59
SELECT 59
Description de fichiers 60
FILE DESCRIPTION (FD) 60
Instructions pour la gestion des fichiers 62
OPEN 62
CLOSE 62
WRITE 62
READ 64
AT END 64
INVALID-KEY 64
REWRITE 66
DELETE 68

START 69
Tableau de synthèse 69

Programmation avancée **71**

Plus de convivialité 72
Programme de menu 72
Génération de pages écran 74
Création des pages écran 74
Affichage des pages écran 75
Nombres aléatoires / opérations sur les chaînes 76
Test de validité de la date 78
Algorithmes de tri 79
Recopie dans un fichier indexé 79
Bubble Sort 81
Arbre binaire 82
Explications 82
Utilisation des indices 86
Utilisation des index 88

Annexes **91**

Mots réservés 91
Tableau récapitulatif des formats 94
Codes d'erreurs des entrées-sorties 96

Index **97**

Introduction

Comment utiliser ce manuel

Organisation

Dans ce manuel, nous avons adopté une organisation thématique qui diffère de l'organisation habituellement rencontrée dans la plupart des manuels de référence. Non seulement cette organisation est mieux adaptée pour servir de support de cours, mais elle est aussi plus accessible au programmeur, c'est-à-dire que, si les difficultés apparaissent dans un certain contexte, on doit pouvoir s'y référer. Par conséquent, la recherche ne s'effectuera pas seulement sur un seul mot, mais sur un thème, et l'on trouvcra avec certitude le terme précis dont on ne se souvenait plus.

Le contenu de cette brochure s'articule autour des trois axes suivants :

1. Les connaissances de base
2. Les tableaux
3. Les fichiers

La partie *programmation avancée* est abordable si l'on a acquis un minimum de connaissances. Cette partie présente des techniques de programmation originales qui profiteront certainement à tous ceux qui veulent aller un peu plus loin.

Si nous avons omis les appels système, le passage de paramètres à des modules écrits en d'autres langages, le tri-fusion, la communication inter-programmes, l'édition de rapports et les fonctionnalités vidéo étendues, c'est parce que deux années d'études suffiraient à peine pour entrer dans le détail de ces domaines très passionnants.

Nous vous souhaitons donc bon courage !

Conventions typographiques

- Les mots COBOL sont toujours écrits en `MAJUSCULES`.
- Les mots COBOL en **`caractères gras`** doivent être utilisés (mots réservés).
- Les mots COBOL en `caractères normaux` sont facultatifs.
- Parmi les instructions entre accolades une seule doit être utilisée.
- Les instructions entre crochets sont facultatives.
- En cas d'ambiguïté l'espace est signalé par le caractère : □
- Les noms de données sont représentés par `nd1`, `nd2`, ...
  les littéraux par `lit1`, `lit2`, ...
- Les termes *champ de données* et *zone de données* sont synonymes.

Présentation du langage

Eléments du langage

Jeu de caractères utilisable

Les chiffres	`0 à 9`
Les lettres	`A à Z` et `a à z`
Les opérateurs	`+ - * /` et `**`
Les signes de ponctuation	`, ; .`
Les caractères spéciaux	`" $ = < > ( )` BLANK

Le COBOL ne fait généralement pas la différence entre les majuscules et les minuscules, sauf pour les textes entre guillemets. Les commentaires, quant à eux, peuvent contenir n'importe quels caractères affichables sur le terminal.

Mots réservés

Les mots réservés constituent le vocabulaire de base du langage de programmation COBOL. Ils se répartissent en plusieurs sous-groupes :

mots-clés (keywords)

Ils sont essentiels car ils forment les instructions. Les mots clés sont affichés en gras dans la description du langage.

mots facultatifs (optional words)

Ils permettent de rendre plus lisibles certaines instructions, par exemple `IS` et `THEN`.

constantes figuratives

Elles sont remplacées par le compilateur :

`ZERO ZEROE ZEROES`	un ou plusieurs zéros
`SPACE SPACES`	un ou plusieurs espaces
`HIGH-VALUE(S)`	valeur la plus haute, FF pour le code ASCII.
`LOW-VALUE(S)`	valeur la plus basse, 00 pour le code ASCII.
`QUOTE QUOTES`	un ou plusieurs guillemets
`ALL` *Littéral*	une ou plusieurs répétitions du Littéral

noms du constructeur

Ces noms sont donnés par le constructeur, en général pour les périphériques d'entrées-sorties.

`CONSOLE`	Console de l'opérateur
`TERMINAL`	Terminal utilisateur
`SYSIN`	Périphérique d'entrée système
`SYSOUT`	Périphérique de sortie système

Noms de variables et de procédures (mots utilisateur)

Ils peuvent être choisis par l'utilisateur avec les quelques restrictions suivantes :

- Les caractères autorisés sont les lettres de l'alphabet, les chiffres et le tiret (signe moins).
- Le tiret de doit pas figurer au début ou à la fin du mot utilisateur.
- La longueur maximale du mot est de 30 caractères.
- Les noms de données doivent comporter au moins une lettre.

Littéraux

Les littéraux sont des constantes qui ne sont pas caractérisées par un nom symbolique. On répartit les littéraux en deux catégories :

Littéraux numériques (constantes numériques)

On différencie les littéraux fixes et flottants. Les caractères autorisés sont les chiffres de 0 à 9, le point décimal, les signes + et – ainsi que le signe exponentiel E pour les littéraux flottants.

Règles pour la formation de littéraux numériques fixes

- au maximum 18 chiffres
- au plus un signe à la première position
- au plus un point décimal, et non à la fin du littéral

Règles pour la formation de littéraux numériques flottants

- Le format d'un littéral flottant est : $\left[\begin{smallmatrix}+\\-\end{smallmatrix}\right]$ `mantisse` **`E`** $\left[\begin{smallmatrix}+\\-\end{smallmatrix}\right]$ `Exposant`
- La mantisse contient au plus 16 chiffres et le point décimal ; elle peut être précédée d'un signe.
- L'exposant est placé derrière le signe **E**. Il comporte un ou deux chiffres et peut être précédé d'un signe.

Exemple 1-1

littéraux autorisés	littéraux interdits
123456789.12345678	12345678901234567
6789	89
3.1415	6 789
+0.7	5.
-6354.0	0.7+
0	5-
001	1,234.56
+1234567890.123456E33	1,2
1.3E01	1234567890.1234567E9
1.3E1	1.3-E01
0.13E02	1.3E001
3.3E00	E11
	33E00

Littéraux alphanumériques (chaîne de caractères)

- Les littéraux alphanumériques sont une suite d'au plus 160 caractères placés entre guillemets, la valeur du littéral étant la chaîne sans les guillemets.

- Caractères autorisés : Tous, sauf le guillemet. Pour pouvoir afficher aussi les guillemets utilisez la constante figurative `QUOTE`.

Exemple 1-2

```
"Constante de texte"
"12345"                  Les calculs sont interdits avec cette constante !
"THAT'S ALL, FOLKS !"
QUOTE                    QUOTE et SPACE sont des constantes figuratives, une forme
SPACE                    particulière de littéraux alphanumériques.
```

Conventions de codage

Règles

Chaque ligne d'un programme COBOL contient normalement 80 caractères. L'espace d'édition est partagé en plusieurs zones dont voici la description.

Colonne 1-6	Cette zone contenait autrefois les numéros de lignes ; elle n'est pas évaluée par le compilateur et devrait donc rester vide. Comme les programmes étaient codés sur des cartes perforées, les numéros de lignes permettaient de remettre ces cartes en ordre lorsque celles-ci tombaient par terre.
Colonne 7	Sert à spécifier des lignes de commentaires, des lignes de débogage ou la suite d'un littéral alphanumérique. **vide** ligne normale. **-** poursuite de ligne (voir plus loin). ***** ligne de commentaires, n'est pas évaluée. **D** ligne de débogage, elle n'est évaluée que si l'on a spécifié la clause `WITH DEBUGGING MODE`.
Colonne 8-11	**Zone A** ; ici commencent tous les titres de paragraphes et de sections.
Colonne 12-72	**Zone B** ; ici commencent les phrases COBOL (les instructions).
Colonne 73-80	Zone de commentaires, elle n'est pas évaluée et contenait autrefois le nom du programme.

- Durant l'écriture d'un programme COBOL, on prendra garde à laisser vide les six premières et les huit dernières colonnes. Un caractère dans les six premières colonnes peut provoquer une erreur ; attention en particulier au caractère de tabulation qui n'apparaît pas dans de nombreux éditeurs !

- Poursuite de ligne : Le littéral doit être écrit jusqu'en colonne 72 et séparé exactement à cet endroit, la ligne suivante doit contenir le signe - (moins) en septième colonne. L'écriture du littéral est poursuivie en Zone B, elle commence et finit par des guillemets.

Exemple 1-3

Colonne	0 1 2 3 4 5 6 7 12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
Zone	A B
Ligne 1	... "Ceci-est-un-littéral-jusqu'à-la-colonne-72-coup
Ligne 2	- "ure-deuxième-moitié-du-littéral-alphanumérique"

Remarque :

Du fait que la seconde moitié du littéral est placée entre guillemets, il n'est pas nécessaire de la faire débuter en colonne 12. Ceci permet alors d'aligner le texte par rapport à la chaîne du dessus.

Suggestions

Indentation

a) Laissez deux espaces entre le numéro de niveau et le nom de donnée.
b) Indentez chaque niveau successif de quatre espaces par rapport au niveau précédent.
c) Alignez les clauses `PIC`, `VALUE`, `SYNC` et `USAGE`.

Procedure Division

a) Un programme devrait n'avoir qu'une seule instruction de fin `STOP RUN` ou `GOBACK`.
b) Essayez d'éviter les sections, utilisez-les pour `SORT` et pour les `DECLARATIVES`.
c) Employez `GO TO` avec parcimonie (mais dans certains cas il est préférable de l'utiliser).
d) Si le niveau d'imbrication de `IF` dépasse 5, revoyez votre algorithme, il y a sûrement moyen de faire plus simple.

Comment rédiger

a) Les commentaires alourdissent le programme et ne doivent pas expliquer ce que font des instructions déjà claires par elles-mêmes[1].
b) Donnez des noms significatifs à vos variables, un nom COBOL peut avoir une longueur de 30 caractères, alors ne soyez pas mesquins ; utilisez autant de caractères qu'il vous faut pour rendre votre programme plus lisible.
c) Ecrivez en toutes lettres les opérateurs relationnels, même si aujourd'hui les caractères < et > sont disponibles sur presque tous les terminaux.

Modèles d'écriture[2]

Exemple 1-4

```
OPEN INPUT  TRANSACTION-FILE
            OLD-MASTER-FILE
     OUTPUT NEW-MASTER-FILE
            REPORT-FILE
            ERROR-FILE.
```

[1] encore que dans un but pédagogique cela puisse être nécessaire...

[2] Ces exemples sont extraits du *Structured Programming Cookbook* de Paul Noll.

Exemple 1-5

```
CLOSE TRANSACTION-FILE
      OLD-MASTER-FILE
      NEW-MASTER-FILE
      REPORT-FILE
      ERROR-FILE.
```

Exemple 1-6

```
MOVE SPACE        TO PR-RECORD.
MOVE TP-ITEM-NO   TO PR-ITEM-NO.
MOVE TP-ITEM-DESC TO PR-ITEM-DESC.
```

Exemple 1-7

```
READ TRANSACTION-FILE
    INTO TP-INTPUT-AREA
    AT END
        MOVE 'Y'        TO TRAN-EOF-SW
        MOVE HIGH-VALUE TO TR-ITEM-NO.
```

Exemple 1-8

```
WRITE PR-OUTPUT-AREA
    INVALID KEY
        PERFORM 420-PRINT-ERROR-MESSAGE.
```

Exemple 1-9

```
MULTIPLY TR-QUANTITY BY TB-UNIT-PRICE
    GIVING IL-SALES-AMOUNT
    ON SIZE ERROR
        PERFORM 360-PROCESS-INVALID-TRAN.
```

Exemple 1-10

```
SEARCH ITEM-CODE-TABLE-ENTRY
    AT END
        PERFORM 320-PRINT-ERROR-MESSAGE
    WHEN IT-ITEM-NO (IT-INDEX) EQUAL TO TR-ITEM-NO
        MOVE IT-UNIT-PRICE (IT-INDEX) TO TP-UNIT-PRICE.
```

Exemple 1-11

```
IF CR-CUST-CODE EQUAL TO 1
    MOVE 0.020 TO DISCOUNT-PERCENT
ELSE
    IF CR-CUST-CODE EQUAL TO 2
        MOVE 0.050 TO DISCOUNT-PERCENT
    ELSE
        MOVE ZERO TO DISCOUNT-PERCENT.
```

Structure du COBOL

Structure générale

SENTENCE

Elément de base d'un programme COBOL, constitué d'instructions qui se terminent par un point. En français, on parle de *phrase* COBOL.

PARAGRAPH

Regroupe plusieurs phrases COBOL reliées entre elles par un lien logique. Chaque paragraphe est introduit par un titre commençant en colonne 8 et s'arrêtant au prochain titre (sinon à la fin du programme).

SECTION

La section est constituée d'unité fonctionnelles formées de paragraphes ayant un lien logique entre eux. Chaque section est introduite par un titre suivi de la clause `SECTION` et se termine à la prochaîne section (ou en fin de programme).

DIVISION

Tout programme COBOL comporte quatre divisions ayant chacune un rôle spécifique. Les divisions sont décrites par la suite.

Les divisions

Identification (IDENTIFICATION DIVISION)

Effet : Identification du programme et documentation du programme.

Format :

```
IDENTIFICATION DIVISION.

PROGRAM-ID. Nom du programme.
[AUTHOR. Nom du programmeur (commentaire).]
[INSTALLATION. Commentaire.]
[DATE-WRITTEN. Commentaire.]
[DATE-COMPILED. Commentaire.]
```

Règles :

- L'en-tête et le nom du programme après `PROGRAM-ID` sont obligatoires. Le champ `DATE-COMPILED` est mise à jour dans le fichier **.lst** au moment de la compilation.
- Le nom du programme fait entre 1 et 30 caractères, mais seulement les 8 premiers sont évalués. Les caractères autorisés sont les chiffres et les lettres, le premier caractère devant être une lettre.

Description des périphériques (ENVIRONMENT DIVISION)

Effet : Description des périphériques, entrées-sorties.

Format :

```
ENVIRONMENT DIVISION.

[CONFIGURATION SECTION.
 [SOURCE-COMPUTER. Nom de l'ordinateur source
                  [WITH DEBUGGING MODE ].
 [OBJECT-COMPUTER. Nom de l'ordinateur objet.]
 [SPECIAL-NAMES.
  Nom du constructeur IS mnémonique.]

[INPUT-OUTPUT SECTION.
 [FILE-CONTROL. Gestion des fichiers.]
 [I-O-CONTROL. Entrées-sorties.]
```

Règles :

- La CONFIGURATION SECTION et la INPUT-OUTPUT SECTION sont optionnelles, c'est à dire qu'il n'est pas toujours nécessaire de décrire le matériel sur lequel on travaille. On peut donner le nom de l'ordinateur après SOURCE-COMPUTER (ou OBJECT-COMPUTER) et la clause WITH DEBUGGING MODE apporte des facilités de débogage.
- A définir éventuellement dans SPECIAL NAMES :

```
CONSOLE IS CRT
CURRENCY SIGN IS F
DECIMAL POINT IS COMMA
```

Description des données (DATA DIVISION)

Effet : Description des données et des fichiers traités par le programme.

Format :

```
DATA DIVISION.

[FILE SECTION.]
[WORKING-STORAGE SECTION.]
[LINKAGE SECTION.]
```

Règles :

- La FILE SECTION contient la description des enregistrements des fichiers, ainsi que la structure de ces enregistrements.
- La WORKING-STORAGE SECTION contient la description de toutes les variables utilisées par le programme.
- Si l'on veut partager des données avec un module maître, celles-ci doivent être décrites dans la LINKAGE SECTION.

Partie des traitements (PROCEDURE DIVISION)

Effet : Partie principale, contient toutes les instructions. La `PROCEDURE DIVISION` est généralement subdivisée en sections et en paragraphes.

Format :

```
PROCEDURE DIVISION.

[Nom de la section SECTION.
 [Nom de paragraphe.
     Phrase COBOL.      ]
                        ]
```

Règles :

- La `PROCEDURE DIVISION` comporte plusieurs paragraphes qui sont réunis en unités fonctionnelles appelées sections (`SECTION`).
- Chaque paragraphe comporte un titre et est composé de phrases COBOL. Celles-ci doivent commencer en colonne 12 (au moins).

Description de données en WORKING-STORAGE SECTION

Définition et description des champs de données

Les champs de données sont les éléments qui vont être traités par le programme. Il ne s'agit pas seulement des données numériques sur lesquelles seront effectués les calculs, mais aussi des chaînes de caractères (qui peuvent faire l'objet d'un tri). Toutes les données doivent être définies et décrites dans la `WORKING-STORAGE SECTION`, de préférence dans l'ordre suivant :

1. Champs de données indépendants : Ces données ne sont pas liées logiquement, elle ne s'intègrent donc pas dans une structure.

2. Champs de données dépendants : Ces données sont caractérisées par le lien logique qui les unit et par leur structure hiérarchique.

Chaque définition d'un champ de données comporte un numéro de niveau suivi du nom de la variable. S'il n'y a pas d'autres champs subordonnés à cette zone, cette dernière doit être décrite à l'aide de la clause `PICTURE`. On parle alors d'une **zone de données élémentaire**.

L'ensemble des champs subordonnés est aussi appelé **groupe de données**.

Format :

```
Niveau nom de champ [{PICTURE} IS masque de formatage.]
                    {PIC    }
```

Règles : a) Niveaux

A l'aide des niveaux, il est possible de relier entre eux des champs de données et de leur imposer une structure hiérarchique. Les numéros de niveaux sont tous à deux chiffres et à choisir parmi les suivants : 01, 02, 03, ... , 49, 66, 77 et 88. Chaque numéro de 01 à 49 correspond à un niveau dans la hiérarchie, sachant que les niveaux ayant les numéros les plus petits sont hiérarchiquement supérieurs aux autres niveaux qui ont des numéros plus grands. Les niveaux 66 et 88 ont une signification spéciale et seront décrits plus loin. Le niveau 77 est réservé aux zones de données indépendantes.

Les numéros de niveau 01 et 77 doivent commencer dans la zone A (colonne 8 à 11), tous les autres commencent soit en zone A, soit en zone B (colonne 12 à 72). Pour plus de clarté les mêmes numéros de niveaux devront être alignés sur une même colonne.

b) Noms de variables

Les noms de variables sont laissés au choix du programmeur. Les restrictions qui s'appliquent aux noms de variables sont décrites au paragraphe Noms de variables et de procédures, **page 2**.

c) Clause `PICTURE`

Dans la clause `PICTURE` sont décrits la structure et le format des champs de données. Le masque se compose de caractères de formatage qui correspondent chacun à un emplacement du champ de données. La chaîne de formatage peut comporter au maximum 30 caractères (il est cependant possible de décrire des champs plus longs en ne répétant pas les caractères mais en indiquant leur nombre entre parenthèses).

d) Formats

9	pour un chiffre
V	pour un point décimal (représentation interne)
S	pour le signe (représentation interne)
E	permet l'entrée d'un exposant
A	pour un signe alphabétique (BLANK, A-Z, a-z)
X	pour n'importe quel caractère
.	affichage d'un point
+	affichage du signe
-	affichage du signe, si valeur négative
,	affichage d'une virgule
B	affichage d'un BLANK
0	affichage d'un zéro
/	affichage d'une barre oblique
Z	affichage d'un chiffre, les zéros à gauche sont affichés comme des blancs
*****	affichage d'un chiffre, les zéros à gauche sont affichés comme des étoiles
CR	(crédit) affiche CR si négatif
DB	(débit) affiche DB si négatif
cs	(currency sign) $ par défaut.

Règles de formatage :

- Les format d'affichage (formats édités) ne permettent pas les calculs.
- Les symboles de formatage **S** et **V** sont représentés en interne, ils ne sont donc normalement pas affichés par la commande DISPLAY. Ceci permet une représentation plus compacte, car ces signes ne comptent pas dans la longueur du champ de données.
- La taille d'un champ de données est définie par le nombre de caractères de formatage, à l'exception de **S** et **V**. On peut raccourcir l'écriture en précisant le nombre de répétitions du caractère entre parenthèses, par exemple : `PIC X(9)`, est identique à `PIC XXXXXXXXX`.
- Les symboles de formatage **V**, **S**, **E** et **.** (point décimal) ne peuvent être utilisés qu'une seule fois dans un masque de formatage. Les signes **V** et **.** s'excluent mutuellement. Le point décimal termine la suppression des zéros en tête, cette règle comporte néanmoins une exception : si tous les chiffres de la zone de données sont prévus pour la suppression des zéros en tête et si la valeur du champ est nul, alors ne seront affichés que des blancs.
- Les symboles de formatage **S,** + et - s'excluent mutuellement. **S** doit se trouver en première position, + et - peuvent être en première ou en dernière position. Ils peuvent se répéter en première position, le signe est alors flottant.
- Pour les affichages flottants de valeurs numériques, plusieurs signes sont à la disposition du programmeur (**Z**, *****, +, ou -). Ces signes doivent être placés au début du masque de formatage.
- **CR** et **DB** doivent se trouver à l'extrême droite du masque, *cs* à l'extrême gauche.

Les différentes classes de données

La construction du masque de formatage différencie implicitement cinq classes de données.

1. données numériques

Symboles de formatage permis : `9 V S E`

Exemple 1-12

`77 N PICTURE IS S99999.`
 	
`77 P PICTURE 999V99.`
	Définit une variable `N` à cinq positions, signée.
	Définit une variable `P` à trois positions et deux postions décimales, non signée.
		Si `N` contient 123, l'affichage sera : 00123
		Si `P` contient 123.45 l'affichage sera : 13245

2. données alphabétiques

Symboles de formatage permis : `A B`

Exemple 1-13

`77 NOM PICTURE IS AAAAAAA.`
 	
`77 NOM PIC A(7).`
	Définit un champ alphabétique NOM à 7 positions.
	Même effet, mais écriture plus courte.
		Le contenu du champ aligné à gauche.
		Même affichage.

3. données alphanumériques[3]

Symboles de formatage permis : `X A 9`

Exemple 1-14

Définition d'une zone de données formée d'un nombre à cinq chiffres (code postal) et de 10 caractères alphabétiques au maximum (ville).

```
77 CPVILLE PIC 9(5)A(10).
77 CPVILLE PIC X(15).
```

4. champs numériques édités

Symboles de formatage permis : `9 . + - , B 0 / Z *`

Exemple 1-15

Définition d'un champ numérique dans un format édité. Attention ! les calculs sont interdits sur ces formats.

`77 NOMBRE PIC +9999.99` | Si NOMBRE contient 123.10 l'affichage sera : +0123.10

5. champs alphanumériques édités

Symboles de formatage permis : `9 . + - , B 0 / Z *`

Exemple 1-16

Définition d' un champ alphanumérique édité de 5 caractères séparés par des blancs.

`77 TITRE PIC XBXBXBXBX` | Si TITRE contient RAMBO : R□A□M□B□O

Affectations

Format	Taille	Valeur	Affichage	Remarques
99999	5	123	00123	
9(5)	5	-123	00123	Erreur de signe
999V99	5	1.2	00120	Point décimal non affiché.
S9V9	2	-1.23	12	Signe non affiché.
S9V9	2	-11	10	Erreur ! valeur = -1
A(10)	10	"MERGUEZ"	MERGUEZ□□□	
AAA	3	"BATMAN"	BAT	
X9X	3	"H2O"	H2O	
9X9	3	"H2O"	H2O	Pas d'erreur.
9(5)	5	123	00123	
9(5)	5	1234567	34567	Erreur de format.
Z(5)	5	123	□□123	
Z(5).ZZ	8	0.1	□□□□.10	
ZZ9,B999.99	11	12345.678	□12,□345.67	
ZZ9,B999.99	11	1.1	□□0,□001.10	
99/99/99	8	010193	01/01/93	
9B9B9B9	7	1993	1□9□9□3	
XBXBXBXBX	9	"TITRE"	T□I□T□R□E	
X0X0X	5	"12345"	10203	

[3] Les masques de formatage contenant exclusivement les signes A ou 9 ne sont pas alphanumériques.

Niveau 66 : Renommer et redéfinir - Ambiguïtés

Les niveaux permettent de créer une structure de données hiérarchique et de regrouper des données liées par un lien logique en groupe de données.
En règle générale, des champs de données différents doivent avoir des noms différents. Pour éviter les ambiguïtés, il faut qualifier par `OF` ou `IN` d'où viennent et où vont les données.

Format : `Nom de donnée-1` { **OF** / **IN** } `nom de donnée-2`

Exemple 1-17

```
01  ADRESSE-1.

    02  NOM
        03  PRENOM      PIC X(20).
        03  NOMFAM      PIC X(20).
    02  LIEU
        03  RUE         PIC 999X(20).
        03  CPOSTAL     PIC 99999.
        03  VILLE       PIC X(30).

01  ADRESSE-2.

    02  PRENOM          PIC X(20).
    02  NOMFAM          PIC X(20).
    02  RUE             PIC 999X(20).
    02  CPOSTAL         PIC 99999.
    02  VILLE           PIC X(30).
```

Pour accéder au nom de famille de la première adresse utilisez `NOMFAM OF NOM` ou `NOMFAM OF NOM OF ADRESSE-1`, et pour la deuxième adresse `NOMFAM OF ADRESSE-2`.

RENAMES

Effet : Tous les champs de données compris entre les noms précisés avant et après `THROUGH` sont rassemblés sous le nom de donnée-1.

Format : **66** nd1 **RENAMES** nd2 [{ **THROUGH** / **THRU** } nd3].

Règles :
- La clause `RENAMES` se rapporte au champ de données qui la précède immédiatement.
- La clause `RENAMES` est interdite avec les niveaux 01, 66, 77 et 88.

Exemple 1-18

```
01  ADRESSE.
    02  PRENOM          PIC X(20).
    02  NOMFAM          PIC X(20).
    02  RUE             PIC X(20).
    02  NUMERO          PIC 999.
    02  CPOSTAL         PIC 99999.
    02  VILLE           PIC X(30).
        66  NOM  RENAMES PRENOM THROUGH NOMFAM.
        66  LIEU RENAMES RUE THROUGH VILLE.
```

REDEFINES

Effet : Permet de donner un deuxième nom à une même zone de données. Attention : L'emplacement en mémoire est le même !

Format :
```
Niveau nom de donnée-1
      REDEFINES nom de donnée-2 PICTURE formatage.
```

Règles :
- Les définitions de nom de donnée-1 et de nom de donnée-2 doivent être consécutives et les données doivent avoir le même niveau.
- La taille des champs redéfinis doit être la même pour les deux noms de données.
- Les clauses `REDEFINES` et `VALUE` s'excluent mutuellement.

Exemple 1-19

```
77 A PICTURE 9(4).
77 B REDEFINES A PICTURE 9(4).
...
MOVE 123 TO A.
DISPLAY A "/" B.
```

Affichage : 0123/0123

Exemple 1-20

```
01  UNE-DATE      PICTURE 9(6).

01  DATE-R REDEFINES UNE-DATE.

    02  JOUR      PICTURE 99.
    02  MOIS      PICTURE 99.
    02  ANNEE     PICTURE 99.
```

Niveau 88 : Noms-condition

Effet : La définition de noms-condition permet de donner des noms à des valeurs fixes de champs. On peut directement tester ces noms avec la clause IF. La condition est vraie si le champ de niveau directement supérieur contient la valeur du nom-condition.

Format :
```
88 nom-condition { VALUE IS  } Lit1 [ { THROUGH } Lit2 ].
                { VALUES ARE }       { THRU    }
```

Règles :
- Le nom-condition est défini directement après le champ dont il dépend.
- Ce champ ne doit pas définir un groupe de données, il contient donc la clause PICTURE.
- On peut définir plusieurs noms-condition pour un champ de données.
- On peut définir une fourchette de valeurs pour un nom-condition, celle-ci allant du littéral-1 au littéral-2.

Exemple 1-21

```
IDENTIFICATION DIVISION.
PROGRAM-ID. GEST-ADR.

ENVIRONMENT DIVISION.
DATA DIVISION.

WORKING-STORAGE SECTION.
77  ENTREE PICTURE X.

    88 nouvelle-adresse      VALUE "1".
    88 modifier-adresse      VALUE "2".
    88 supprimer-adresse     VALUE "3".
    88 fin-du-programme      VALUE "0".
    88 non-valable           VALUE "4" THROUGH "9".
...
PROCEDURE DIVISION.
MENU.
      DISPLAY "PROGRAMME DE GESTION D'ADRESSES".
      DISPLAY "==============================".
      DISPLAY "VOTRE CHOIX :".
      DISPLAY "Nouvelle adresse .......... (1)".
      DISPLAY "Modifier une adresse ...... (2)".
      DISPLAY "Supprimer une adresse ..... (3)".
      DISPLAY "Fin du programme .......... (0)".

PARAGRAPHE-ENTREE.
      ACCEPT ENTREE.
      IF ENTREE NOT NUMERIC OR non-valable
          DISPLAY "Mauvaise entrée !"
          GO TO PARAGRAPHE-ENTREE.

TRAITEMENT.
      IF nouvelle-adresse  PERFORM NOUVEAU.
      IF modifier-adresse  PERFORM MODIFIER.
      IF supprimer-adresse PERFORM SUPPRIMER.
      IF fin-du-programme  STOP RUN.
      GO TO MENU.

NOUVEAU.
...
MODIFIER.
...
SUPPRIMER.
...
```

Elargissement de la description de données

FILLER

Effet : Remplit un champ de nom `FILLER` avec des blancs ou du texte destinés à améliorer la présentation d'une page d'écran. `FILLER` est aussi d'une grande utilité dans la gestion des fichiers. Les enregistrements étant lus dans un certain ordre, on peut déclarer en `FILLER` les parties qui ne seront pas traitées. Ceci apporte un gain de temps considérable pour le traitement des fichiers ayant une structure complexe.

Format : `Niveau` **`FILLER`** `Format.`

Règles :

- Le nom de champ `FILLER` ne doit être utilisé que dans les zones de données élémentaires.
- `FILLER` représente un champ de type alphanumérique.
- On peut utiliser `FILLER` autant de fois que l'on veut dans la `WORKING-STORAGE SECTION`.

Exemple 1-22

```
01 AFFICHAGE.
      02 FILLER PIC X(18) VALUE "La factorielle de ".
      02 N      PIC 9.
      02 FILLER PIC X(5) VALUE " est : ".
      02 fact   PIC 9(8).
```

Par exemple, si N a la valeur 5, l'affichage sera :
La factorielle de 5 est : 00000120

Exemple 1-23

On veut extraire d'un fichier de paye les noms des employés qui gagnent plus de 15000 francs par mois. Pour la lecture et le traitement du fichier il faut d'abord définir sa structure. A l'aide de la clause `FILLER` on ne définira que les champs qui nous intéressent. La longueur totale de l'enregistrement est de 130 caractères. Le nom et le prénom comportent chacun 15 caractères et commencent à la position 13. Le salaire commence en position 69 et est de format numérique, six chiffres pour la partie entière et deux chiffres pour la partie décimale, sans point décimal.

```
01  TRAITEMENT-SALAIRE.

    02  FILLER          PIC X(12).

    02  NOM.
        03  PRENOM      PIC A(15).
        03  NOMFAM      PIC A(15).

    02  FILLER          PIC X(26).
    02  SALAIRE         PIC 9(6)V99.
    02  FILLER          PIC X(53).
```

VALUE

Effet : En COBOL toutes les variables devraient être initialisées avant de subir un traitement. L'avantage de la clause VALUE est qu'elle permet d'initialiser les variables dans la WORKING-STORAGE SECTION avant qu'aucun calcul ne soit effectué dans la PROCEDURE DIVISION.

Format : Niv. nd1 **PICTURE** format **VALUE** IS { Lit. numérique / Lit. alphanumérique }.

Règles :
- La clause VALUE est interdite pour les formats édités numériques.
- La clause VALUE ne doit pas être utilisée avec JUSTIFIED, SYNCHRONIZED, ou USAGE, dans une description de données contenant REDEFINES ou un champ subordonné à cette définition.
- Pour les formats édités alphanumériques la clause VALUE est acceptée, mais l'initialisation ne se fait pas au format édité.

Exemple 1-24

```
77  NOTE          PIC 999     VALUE ZERO.
77  DEPART        PIC 99999   VALUE 1.
77  PI            PIC 9V9999  VALUE 3.1415.
77  RESULTAT      PIC X(18)   VALUE "Le résultat est : ".
77  VIDE          PIC X(80)   VALUE SPACES.
77  TIRETS        PIC X(80)   VALUE ALL "-".
77  alpha-n-d     PIC XB9BX   VALUE "H 2 O".
```

BLANK WHEN ZERO

Effet : Il est parfois souhaitable que l'affichage d'une valeur nulle se fasse par l'affichage d'un blanc. Ceci peut être réalisé grâce à la clause BLANK WHEN ZERO.

Format : Définition de données **BLANK** WHEN **ZERO**.

Règles :
- La clause BLANK WHEN ZERO est placée après la description de donnée dans la WORKING-STORAGE SECTION.
- Un champ numérique comportant la clause BLANK WHEN ZERO est un format édité.
- La clause VALUE étant interdite pour les champ numériques édités, on ne pourra pas l'utiliser conjointement à BLANK WHEN ZERO.

Exemple 1-25

```
77  SORTIE-1      PIC ZZ9.99  BLANK WHEN ZERO.
77  SORTIE-2      PIC ZZZ.ZZ.
```

Ces deux définitions sont équivalentes : Le programme n'affiche rien pour une valeur nulle. Dans la deuxième ligne, on est plus limité dans la définition du format (uniquement **Z**).

JUSTIFIED-RIGHT

Effet : Lors de l'affichage, les données alphabétiques et alphanumériques non éditées sont justifiées à gauche. La clause `JUSTIFIED-RIGHT` permet de les aligner à droite dans le champ.

Format : `Définition de donnée` { **JUSTIFIED** / **JUST** } **RIGHT.**

Règles :

- La clause `JUSTIFIED-RIGHT` est placée après la description de donnée dans la `WORKING-STORAGE SECTION`.
- La clause n'est autorisée que pour les champs alphabétiques et alphanumériques.

SIGN

Effet : Le symbole de formatage **S** est représenté en interne, on ne réserve donc aucune place pour le signe dans le masque de formatage. Il est donc impossible de recueillir des données entrées au clavier dans un littéral numérique non édité. Il doit néanmoins être possible de faire des entrées au clavier de nombres signés.
Le langage COBOL a fait son apparition au temps où l'on codait encore les programmes sur des cartes perforées. Les données étaient codées de la même façon, et selon la convention du COBOL on marquait les nombres négatifs en perforant le dernier chiffre du nombre dans la ligne du signe. Cet emplacement donnait automatiquement un nouveau caractère : 1 devenait J, 2 devenait K, 3 devenait L, etc. Le 9 se transformait finalement en R. Par exemple, pour entrer la valeur numérique -123 dans un champ de données avec le format S999, il fallait coder (perforer) 12L.
De façon analogue on pouvait indiquer explicitement le signe positif en remplaçant dans le dernier emplacement 1 par A, 2 par B, ou 9 par I.
Cette façon de procéder, bien qu'encore pratiquée aujourd'hui sur des machines modernes, a deux grands défauts : d'abord l'utilisateur doit avoir les connaissances pour effectuer ce codage, et ensuite il est impossible de perforer le 0. Ainsi, les nombres négatifs ne pouvaient pas être entrés au clavier.
On a donc créé avec la clause SIGN une possibilité de définir pour S une position propre dans le masque de formatage.

Format : [**SIGN** IS] { **LEADING** / **TRAILING** } [**SEPARATE** CHARACTER]**.**

Règles :

- La clause `SIGN` est utilisée exclusivement avec les champs de données contenant le symbole de formatage S dans leur masque de formatage.
- Avec `LEADING`, le signe est fixé au premier emplacement du champ numérique.
- Avec `TRAILING`, le signe est fixé au dernier emplacement du champ numérique.
- Si le signe doit avoir sa propre position dans le champ numérique, il faut le préciser avec la clause `SEPARATE CHARACTER`.

Remarque :

La saisie d'un nombre négatif doit correspondre exactement au masque de formatage. Par exemple, si ce dernier est de la forme `S999 LEADING SEPARATE`, pour entrer la valeur -12 il faudra taper -012.

USAGE

Effet : Spécifie la représentation interne des données.

Format :

```
USAGE IS { BINARY
         | COMPUTATIONAL
         | COMP
         | DISPLAY
         | PACKED-DECIMAL }
```

Règles :

- `BINARY`, `COMPUTATIONAL` et `COMP` sont synonymes. `PACKED-DECIMAL` est aussi synonyme de `COMPUTATIONAL-3` (nombres réels).
- La clause `USAGE` peut être écrite à n'importe quel niveau. Si elle est écrite derrière un champ de données *tableau* de niveau 01, elle s'applique à tout le groupe de données ; attention : *tableau* lui-même ne pourra pas être utilisé dans les calculs.
- La clause `USAGE` d'un champ de données ne doit pas contredire celle du groupe auquel il appartient.
- La clause `USAGE` n'est pas autorisée pour les niveaux 66 et 88.
- Par défaut, on est en `USAGE DISPLAY`.

Remarque :

La clause `USAGE` agissant sur la représentation interne des données, elle n'affecte pas l'utilisation normale des données dans le programme. Cependant, si les champs numériques entiers sont déclarés en `USAGE COMPUTATIONAL`, on obtient un gain de performances d'environ 25 % sur les opérations arithmétiques. De plus, lorsque l'on définit des tables de taille importante, la taille du fichier exécutable s'en trouve réduite.

Programmation COBOL

Fonctions d'entrées-sorties

ACCEPT

Effet : Permet de lire des données entrées au clavier, l'heure ou la date système.

Format : **ACCEPT** nd1 [**FROM** { **DATE** / **DAY** / **TIME** }] [**AT** LINE y COL x] [**WITH** attribut].

Règles :

- DATE est la date système (format : AAMMJJ).
- DAY est le jour système (format : AANNN). Les deux premières positions représentent l'année, les trois suivantes le numéro du jour dans l'année. Ainsi le 1er juillet 1968 sera codé : 68183.
- TIME est l'heure système (format : HHMMSSNN). TIME se base sur le temps écoulé depuis minuit, par exemple 14 h 41 sera codé : 14410000. La valeur minimale de TIME est 00000000, la valeur maximale 23595999.
- Les attributs peuvent varier suivant l'implémentation.

Exemple d'attributs :

AUTO-SKIP	poursuit le programme dès que le champ est rempli
NO-ECHO	n'affiche pas les caractères entrés
SECURE	idem
PROMPT ">"	affiche une invite

DISPLAY

Effet : Affichage de données à l'écran.

Format : **DISPLAY** { nom de donnée / littéral } [**AT** LINE y COL x] [**WITH** attribut].

Règles :

- On peut omettre LINE et COL, mais dans ce cas il conviendra d'accoler x à y, par exemple : `DISPLAY "kestananapété" AT LINE 5 COL 4.`
  est équivalent à : `DISPLAY "kestananafout" AT 0504.`

Attributs :

BEEP	Signal sonore
BLANK LINE	Effacement de la ligne
BLANK SCREEN	Effacement de l'écran
BLINK	Clignotement
HIGHLIGHT	Surbrillance
NO ADVANCING	Ne saute pas à la ligne après affichage
REVERSE-VIDEO	Vidéo inverse
UNDERLINE	Souligné

Fonctions d'affectation

MOVE

Effet : Transfert de données ou de groupes de données vers un autre champ : *nom-de-donnée-2* reçoit *nom-de-donnée-1*.

Format : `MOVE` { `nom-de-donnée-1` / `littéral` } `TO nom-de-donnée-2.`

Règles :

- La zone émettrice reste toujours inchangée.
- Les zones émettrice et réceptrice peuvent être des données ou des groupes de données.
- Les groupes de données sont traités comme des champs alphanumériques.
- Les **zones réceptrices numériques** sont transmises conformément à la position du point décimal (sinon justifiées à droite).
  Si la zone de réception est :
  - **trop grande**, on remplit avec des zéros,
  - **trop petite**, le nombre est tronqué de part et d'autre.

  L'erreur qui en résulte peut être détectée grâce à la clause `ON SIZE-ERROR`.

- Les **zones réceptrices alphabétiques** et alphanumériques sont justifiées à droite si la clause `JUSTIFIED-RIGHT` est précisée, sinon à gauche.
  Si la zone de réception est :
  - **trop grande**, on remplit avec des blancs,
  - **trop petite**, le littéral est tronqué.

- Si l'on affecte une valeur numérique signée vers un champ numérique non signé, la valeur reçue est sa valeur absolue.

Exemple 2-1

zone d'émission		zone de réception	
Masque de données	**Contenu**	**Masque de données**	**Contenu**
99999	12345	9999999	0012345
99999	12345	999	345 (ERREUR !)
99999	12345	X(7)	"12345□□"
99999	12345	X(3)	"123"
99V99	9876	999.9	098.7
99V99	9876	9.999	8.760 (ERREUR !)
99V99	1234	X(7)	- interdit -
99.99	12.34	A(7)	- interdit -
99.99	12.34	X(3)	"12."
A(4)	"BOUM"	X(3)	"BOU"
A(4)	"BOUM"	X(3) JUST	"OUM"
A(4)	"BOUM"	XBX	"B□O"
A(4)	"BOUM"	XBX JUST	"U□M"
X(6)	"3.1415"	9.99	3.14
X(5)	"BOUM"	9.99	ERREUR EXECUTION

	zone de réception					
zone d'émission	num-entier	num-frac	num-édité	alpha	alphanum	alphanum-édité
num-entier	+	+	+	-	+	+
num-frac	+	+	+	-	-	-
num-edité	-	-	-	-	+	+
alphabétique	-	-	-	+	+	+
alphanum	+(!)	+(!)	+(!)	+(!)	+	+
alphanum-édité	-	-	-	+(!)	+	+

Les transferts marqués d'un point d'exclamation (!) ne fonctionnent que si tous les caractères de la zone émettrice sont autorisés dans la zone réceptrice.

MOVE CORRESPONDING

Effet : Contrairement à `MOVE` qui traite les groupes de données comme un seul champ, `MOVE CORRESPONDING` est un transfert de données *intelligent*.

Format : **MOVE** { **CORRESPONDING** / **CORR** } nd1 **TO** nd2

Règles :
- Les champs d'émission et de réception ne sont pas des zones de données élémentaires.
- Seules les zones ayant le même nom dans les groupes de données émettrices et réceptrices sont transférées, les autres champs restent inchangés.

Exemple 2-2

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RECUP-DATE.
ENVIRONMENT DIVISION.
DATA DIVISION.

WORKING-STORAGE SECTION.
01  DATE-SYS.
    02 ANNEE      PIC 99.
    02 MOIS       PIC 99.
    02 JOUR       PIC 99.

01  DATE-AFF.
    02 JOUR       PIC 99.
    02 FILLER     PIC X       VALUE "-".
    02 MOIS       PIC 99.
    02 FILLER     PIC X       VALUE "-".
    02 ANNEE      PIC 99.

PROCEDURE DIVISION.
    ACCEPT DATE-SYS FROM DATE.
    MOVE CORRESPONDING DATE-SYS TO DATE-AFF.

AFFICHAGE.
    DISPLAY DATE-AFF.
```

INITIALIZE

Effet : Initialisation de certains champs d'un type déterminé avec des valeurs données.

Format :

```
INITIALIZE nd1
    [ REPLACING { ALPHABETIC              } DATA BY { nd2 }
                { ALPHANUMERIC            }         { lit }
                { NUMERIC                 }
                { ALPHANUMERIC-EDITED     }
                { NUMERIC-EDITED          } ]
.
```

Règles :

- *nd1* est la zone de réception, *nd1* la zone d'émission.
- Un index ne peut pas être un opérande de `INITIALIZE`.
- *nd1* et les champs subordonnés ne doivent pas contenir les clauses `DEPENDING`, `OCCURS` ou `RENAMES` dans leur description.
- `INITIALIZE` sans la clause `REPLACING` initialise par défaut les champs alphabétiques, alphanumériques et aphanumériques-édités avec des espaces et les champs numériques et numériques-édités avec des zéros.
- L'initialisation se fait de la même façon qu'avec l'instruction `MOVE`, elle doit donc respecter la classe des données.

Exemple 2-3

```
01  action.
    02  titre     PIC X(20).
    02  date1     PIC X(8).
    02  valeur    PIC S9(6)V99.
...
INITIALIZE action NUMERIC DATA BY ZEROES.
INITIALIZE action ALPHANUMERIC DATA BY SPACES.
```

ou plus simplement :

```
INITIALIZE action.
```

Instructions de contrôle

COPY

Effet : `COPY` est une directive de compilation permettant l'incorporation de texte dans le programme source. On l'utilise en particulier pour insérer des en-têtes standards dans plusieurs fichiers, ce qui évite de les réécrire à chaque fois. Ceci est aussi valable pour des fonctions que l'on utilise souvent.

Format :

```
COPY nom-de-fichier [ { OF } nom-de-bibliothèque ]
                    { IN }

    [ REPLACING { pseudo-texte-1 } BY { pseudo-texte-2 } ]
                { nom-donnée-1   }    { nom-donnée-2   }
                { littéral-1     }    { littéral-2     }
                { mot-1          }    { mot-2          }
.
```

Règles :

- Le nom de fichier ne doit pas contenir d'extension. Le compilateur COBOL cherche ce fichier dans le répertoire courant. Ensuite, s'il ne le trouve pas, il lui ajoute l'extension par défaut **.cbl**
- Le nom de librairie est en fait une lettre et doit être un sous-répertoire du répertoire courant. Par exemple **`COPY prog OF A`** sera converti en : **`A/prog`**.
- mot-1 et mot-2 sont des mots COBOL.
- pseudo-texte-1 ne doit pas être vide, en revanche pseudo-texte-2 peut l'être.
- Le texte à inclure ne doit pas contenir la clause `COPY`.
- Si la clause `REPLACING` est spécifiée, chaque occurrence de pseudo-texte-1, nom-donnée-1, littéral-1 ou mot-1 sera remplacée par pseudo-texte-2, nom-donnée-2, littéral-2 ou mot-2. Les lignes de commentaires restent inchangées.
- On ne peut pas préciser de lignes de débogage dans pseudo-texte-1. Les lignes de débogage sont traitées comme des lignes normales.

STOP RUN

Format : **`STOP RUN.`**

Règles :

- La clause `STOP RUN` doit figurer au moins une fois dans le programme.
- Après l'exécution de `STOP RUN`, le programme rend la main au système d'exploitation.

Instructions de saut

GO TO

Effet : Interrompt le déroulement du programme et saute à un endroit déterminé.

Format : **`GO TO`** { nom de paragraphe / nom de SECTION } **.**

Règles :

- Le programme poursuit son exécution à l'endroit précisé après `GO TO`.
- Utilisez `GO TO` pour sortir rapidement d'une boucle imbriquée, dans les autre cas utilisez `PERFORM`. L'utilisation simultanée de `GO TO` et de `PERFORM` est fortement déconseillée, du moins pour les débutants.

GO TO DEPENDING ON

Effet : Extension de la commande précédente, elle peut être considérée comme un aiguillage. La valeur du nom de donnée donne le rang dans la liste du paragraphe qui va être exécuté. Si cette valeur est inférieure à 1 ou supérieure au nombre de paragraphes présents dans la liste, l'instruction de saut est ignorée.

Format : **`GO TO`** paragraphe-1 paragraphe-2 **`DEPENDING`** ON nd1.

Règles :

- Le nom de donnée doit être défini en tant qu'entier numérique.
- Les paragraphes doivent tous exister dans le programme.

Exemple 2-4

```
IDENTIFICATION DIVISION.
PROGRAM-ID. AIGUILLAGE.

ENVIRONMENT DIVISION.
DATA DIVISION.

WORKING-STORAGE SECTION.

77  CHOIX PICTURE 9 VALUE ZERO.

PROCEDURE DIVISION.
MENU.
    DISPLAY "Programme de gestion de stocks".
    DISPLAY "Entrées en stock ..........(1)".
    DISPLAY "Sortie de stock ...........(2)".
    DISPLAY "Choix de l'entrepôt .......(3)".
    DISPLAY "Statistiques ..............(4)".
    DISPLAY "Inventaire ................(5)".
    DISPLAY "Fin du programme ..........(0)".

    ACCEPT CHOIX.

    GO TO ENTREE SORTIE ENTREPOT STATISTIQUES INVENTAIRE
          DEPENDING ON CHOIX.

    STOP RUN.

    ENTREE.
    ...
    SORTIE.
    ...
    ENTREPOT.
    ...
    STATISTIQUES.
    ...
    INVENTAIRE.
    ...
```

Opérations arithmétiques

Règles générales

- Tous les opérandes doivent être de format numérique non édité.
- Un opérande a au maximum 18 positions.
- Les résultats sont arrondis sur la dernière position si l'on précise la clause `ROUNDED`.
- Si l'on précise la clause `ON SIZE ERROR`, la phrase impérative suivant la clause est exécutée dans le cas d'un dépassement de taille. C'est le cas lorsque le résultat ne tient plus dans le champ de données décrit dans la `WORKING-STORAGE SECTION`, ou lors d'une division par zéro.
- Si la clause `ON SIZE ERROR` n'est pas précisée, la division par zéro ne provoque pas d'erreur et le programme poursuit les calculs avec une valeur erronée !

ADD

Format : **ADD** { nd1 / lit1 } **TO** nd2 [**ROUNDED**] [**ON SIZE ERROR** phrase-imp].

ou

ADD { nd1 / lit1 } **... GIVING** nd2 [**ROUNDED**]
[**ON SIZE ERROR** phrase-imp].

Règles :

- Tous les champs de données ou littéraux précédant la clause GIVING sont additionnés et affectés à chaque variable suivant GIVING.

Exemple 2-5

```
77 A PIC 9V99 VALUE 1.19.
77 B PIC 99V9 VALUE 5.7.
77 C PIC 9V99 VALUE 9.99.
77 D PIC 9.9.

ADD 1 C TO A B ROUNDED.                    | A = 12.18   (1+C+A=12.18)
                                           | B = 16.7    (1+C+B=16.69)
ADD A B C GIVING D.                        | D = 6.8     (A+B+C=16.88)
ADD A B C GIVING D ON SIZE ERROR           | D = 0
                        MOVE ZERO TO D.
```

SUBTRACT

Format : **SUBTRACT** { nd1 / lit } **FROM** nd2 [**ROUNDED**] [**ON SIZE ERROR** phrase-imp].

ou

SUBTRACT { nd1 / lit } **... FROM** { nd2 / lit }
GIVING nd3 [**ROUNDED**] [**ON SIZE ERROR** phrase-imp].

Règles :

- Les champs résultats peuvent être de format édité.

Exemple 2-6

```
77 A PIC  9V99 VALUE 1.11.
77 B PIC 99V9  VALUE 5.7.
77 C PIC  9V99 VALUE 9.99.
77 D PIC  9V9  VALUE 8.3.

SUBTRACT A A FROM B ROUNDED C.             | B = 3.5     (B-A-A=3.48)
                                           | C = 7.77    (C-A-A=7.77)
SUBTRACT A A FROM C GIVING D ROUNDED.      | D = 7.8     (C-A-A=7.77)
SUBTRACT 1 A B FROM C D.                   | C = 2.18    (C-(1+A+B)=2.18)
                                           | D = 0.4     (D-(1+A+B)=0.49)
```

MULTIPLY

Format : **MULTIPLY** {nd1 / lit} **BY** nd2 [**ROUNDED**] [**ON SIZE ERROR** phrase-imp].

ou

MULTIPLY {nd1 / lit1} **BY** {nd2 / lit2}
GIVING nd3 [**ROUNDED**] [**ON SIZE ERROR** phrase-imp].

Règles :
- Les champs résultats peuvent être de format édité.

Exemple 2-7

```
77 A PIC 9V99 VALUE 1.11.
77 B PIC 99V9 VALUE 5.7.
77 C PIC 9V99 VALUE 9.99.
77 D PIC 9.9.

MULTIPLY A BY B C.                          | B = 6.3     (A*B=6.327)
                                            | C = 1.08    (A*C=11.0889)
MULTIPLY A BY C GIVING B D                  | B = 11.0    (A*C=11.0889)
           ON SIZE ERROR GO TO ERREUR.      | D inchangé, saut vers
                                            | paragraphe ERREUR
```

DIVIDE

Effet : Division.

Format : **DIVIDE** {nd1 / lit} **INTO** nd2 [**ROUNDED**] [**ON SIZE ERROR** phrase-imp].

ou

DIVIDE {nd1 / lit1} {**BY** / **INTO**} {nd2 / lit2}
GIVING nd3 [**ROUNDED**] [**ON SIZE ERROR** phrase-imp].

ou

DIVIDE {nd1 / lit1} {**BY** / **INTO**} {nd2 / lit2}
GIVING nd3 [**ROUNDED**] **REMAINDER** nd4
[**ON SIZE ERROR** phrase-imp].

Règles :
- Les champs résultats peuvent être de format édité.
- Le reste de la division se calcule de la façon suivante :

  Reste = dividende – (diviseur × résultat).

- Si l'on utilise la clause REMAINDER avec la clause ROUNDED, le reste est d'abord formé et ensuite seulement le résultat est arrondi.

Exemple 2-8

```
77 A PIC 99V9 VALUE 20.
77 B PIC 99V9 VALUE 1.5.
77 C PIC 99V9 VALUE 9.
77 D PIC 9.9.

DIVIDE A INTO B C ROUNDED.          | B = 0       (B/A=0.075)
                                    | C = 0.5     (C/A=0.45)
DIVIDE A BY B GIVING C              | B = 13.3    (A/B=13.3333333)
    REMAINDER D.                    | D = 0 (!)   (A-B*C=20-19.95=0.05)
DIVIDE C BY A                       | D = 0.4     (C/A=0.45)
    GIVING D ROUNDED                | C = 1       C-A*D=9-20*0.4=9-8=1)
    REMAINDER C.                    | D = 0.5     (arrondi après le calcul du
                                    |             résultat)
```

COMPUTE

Effet : Traite des formules entières. Les opérateurs sont :

l'addition	+	la division	/
la soustraction	-	la puissance	**
la multiplication	*	les parenthèses	(et)

Format :
```
COMPUTE nd1 [ROUNDED] = expression arithmétique
                        [ON SIZE ERROR phrase impérative].
```

Règles :

- Le signe de multiplication ne peut pas être omis comme cela est courant en algèbre.
- Tous les opérateurs doivent être délimités par des espaces.
- Avant la parenthèse ouvrante et après la parenthèse fermante il doit y avoir un espace, sinon un point s'il s'agit de la fin de la phrase COBOL.
- Tous les noms de données et littéraux de l'expression arithmétique doivent être de format numérique non édité.
- Le champ de données précisé immédiatement après `COMPUTE` peut être de format édité.
- `COMPUTE` devrait être réservé exclusivement aux formules complexes, car les calculs se font plus rapidement avec les instructions arithmétiques de base.

Exemple 2-9

Exemple d'instructions	Formules
`COMPUTE A ROUNDED B C = (A + X) ** 2 + 2 / 4.`	$(A+X)^2+\frac{2}{4}$
`COMPUTE F = A ** 2 + 2 * A * B + B ** 2` `    ON SIZE ERROR DISPLAY "TROP GRAND !".`	$A^2+2\times A\times B+B^2$
`COMPUTE X = 1 / X ON SIZE ERROR` `    DISPLAY "DIVISION PAR ZERO !".`	$X=\frac{1}{X}$

Opérations sur les chaînes de caractères

EXAMINE

EXAMINE TALLYING

Effet : Compte les occurrences d'un caractère dans une chaîne.

Format :
```
EXAMINE nd1 TALLYING { ALL          } littéral.
                     { LEADING      }
                     { UNTIL FIRST  }
```

Règles :
- Le nombre d'occurrences du littéral dans nom-de-donnée est placé dans **TALLY**, une **variable définie automatiquement** de format 99999.
- `ALL` compte le nombre de fois où le littéral apparaît dans la chaîne.
- `LEADING` compte le nombre de fois où le littéral est en début de donnée.
- `UNTIL FIRST` nombre de caractères jusqu'à la première apparition du littéral.

Exemple :

```
77 nombre   PIC 9(6).
77 nom      PIC A(9).
77 X        PIC X(12).

MOVE 132456 TO nombre.
EXAMINE nombre TALLYING UNTIL FIRST "3".     | TALLY = 00002
MOVE "Catherine" TO nom.
EXAMINE nom TALLYING ALL "e".                | TALLY = 00002
MOVE "A1B2C3D4" TO X.
EXAMINE X TALLYING LEADING "D".              | TALLY = 00000
MOVE "DDDA1B2C3 D4" TO X.
EXAMINE X TALLYING LEADING "D".              | TALLY = 00003
```

EXAMINE REPLACING

Effet : Remplace des caractères dans une chaîne.

Format :
```
EXAMINE nd1 REPLACING { ALL          } lit1 BY lit2.
                      { LEADING      }
                      { [UNTIL] FIRST }
```

Règles :
- `ALL`, tous les caractères correspondant au littéral-1 seront remplacés par le littéral-2.
- `LEADING`, tous les premiers caractères correspondant au littéral-1 seront remplacés par le littéral-2.
- `UNTIL-FIRST`, tous les caractères jusqu'à la première occurrence du littéral-1 seront remplacés par le littéral-2.
- `FIRST`, la première occurrence du littéral-1 sera remplacée par le littéral-2.

Exemple 2-10

```
77 valeur PIC Z(10).99.
77 sortie PIC X(13).
77 nom    PIC A(9).
77 mot    PIC X(4).

MOVE "Laurent" TO nom.                                  | nom = "Laurent--"
EXAMINE nom REPLACING ALL BLANK BY "-"                  |
MOVE 1234567.89 TO valeur.                              |
MOVE valeur TO sortie.                                  | sortie
EXAMINE sortie REPLACING LEADING BLANK BY "*".          | ="***1234567.89"
MOVE "4711" TO mot.                                     | mot = "1111"
EXAMINE mot REPLACING UNTIL FIRST "1" BY "1".           |
```

EXAMINE TALLYING REPLACING

Effet : Compte et remplace des caractères dans une chaîne. Dans le nom de donnée, le littéral-1 est d'abord compté puis remplacé par le littéral-2. Le résultat du comptage se trouve dans **TALLY**, une variable créée automatiquement à cet effet et **qui n'est pas à définir** dans la WORKING-STORAGE SECTION.

Format :

```
EXAMINE nd1 TALLYING { ALL          } lit1 REPLACING BY lit2.
                     { LEADING      }
                     { UNTIL FIRST  }
```

Règles :

- Le comptage et le remplacement se font de la même façon que dans EXAMINE TALLYING et EXAMINE REPLACING.

STRING

Effet : Permet de concaténer plusieurs zones de données dans une seule zone.

Format :

```
STRING { nd1          } [ nd12          ] DELIMITED BY { nd13          }
       { littéral-1   } [ littéral-12   ]             { littéral-13  }
                                                      { SIZE         }

[ { nom-donnée-2 } [ nd22          ] DELIMITED BY { nd23          } ]
[ { littéral-2   } [ littéral-22   ]             { littéral-23  } ]
                                                 { SIZE         }

  INTO nd3 [WITH POINTER nd4] [ON OVERFLOW phrase-imp].
```

Règles :

- Les zones de données émettrices et réceptrices doivent être alphanumériques et les transferts de données suivent les règles habituelles. On remarquera toutefois que, lorsque la zone réceptrice est plus grande que la somme des zones émettrices, il n'y a pas de remplissage à droite par des espaces.
- La zone réceptrice nom-donnée-3 doit être une zone élémentaire déclarée sans symbole d'édition.
- L'option DELIMITED BY permet de spécifier une limite de transfert de données. Cette limite peut être désignée par le contenu de nom-donnée-13 ou littéral-13 (les caractères délimitant nom-donnée-13 ou littéral-13 ne sont pas transmis) ou par la taille (option SIZE) de la zone réceptrice.

Règles :

- L'option POINTER permet de préciser la position de gauche du transfert en zone réceptrice. Nom-donnée-4, zone élémentaire numérique, doit alors avoir été chargée avec la valeur correspondante. Au fur et à mesure, le pointeur nom-donnée-4 est incrémenté de 1 pour chaque caractère transféré.
- L'option OVERFLOW n'agit que conjointement à l'option POINTER dans le cas où le contenu de nom-donnée-4 est inférieur à 1 ou supérieur au nombre de caractères de la zone réceptrice.

Exemple 2-11

```
77 zone PIC X(12) VALUE SPACE.

STRING "ANNEE" SPACE "1974"                 | zone = ANNEE□1974□□ [4]
      DELIMITED BY SIZE INTO ZONE.
```

UNSTRING

Effet : Eclate une chaîne de caractères en plusieurs sous-chaînes.

Format :

```
UNSTRING nd-1 [ DELIMITED BY [ALL] { nd-12  }                    ]
              [                    { lit-12 }                    ]
              [                                                  ]
              [          [ OR [ALL] { nd-13  } ]                 ]
              [                     { lit-13 } ]                 ]

[WITH POINTER nd-2] [TALLYING IN nd-3]

[ON OVERFLOW ordre-imp.] INTO nd4 [DELIMITER IN nd41]

[COUNT IN nd42] [ndi ] [DELIMITER IN ndi1] [COUNT IN ndi2].
```

Règles :

- Les règles de transfert de données sont identiques à celles du verbe STRING. Nom-donnée-1 est la zone émettrice à éclater. La règle d'éclatement est donnée par l'option DELIMITED BY. Les caractères qui précèdent le contenu de nom-donnée-12, littéral-12, nom-donnée-13, littéral-13, etc. sont transférés en zone réceptrice.
- L'option ALL est destinée à éliminer les occurrences multiples d'un caractère délimiteur. Par exemple, ALL "AB" signifie que AB ou même ABABAB seront considérés comme délimiteurs.
- Nom-donnée-2, zone numérique élémentaire, permet de compter le nombre de caractères examinés dans la zone émettrice (ne pas oublier de l'initialiser à zéro).
- Nom-donnée-3, zone numérique élémentaire, permet de compter le nombre de zones émettrices créées (ne pas oublier de l'initialiser à zéro).
- Lorsque l'option n'a pas été spécifiée et qu'une condition OVERFLOW est déclenchée, l'instruction UNSTRING est arrêtée et le traitement poursuivi à l'instruction suivante.

[4] On trouvera un exemple plus sophistiqué dans le chapitre *Programmation Avancée*.

Règles :

- L'option `OVERFLOW` arrête le déroulement de l'instruction `UNSTRING` essentiellement dans deux cas :

  a) Lorsque le contenu du pointeur nom-donnée-2 est négatif ou supérieur à la taille de la zone émettrice nom-donnée-1

  b) Lorsque toute la zone émettrice n'a pas été examinée et qu'il n'y a plus de zone réceptrice disponible.

- Nom-donnée-4, nom-donnée-i, sont les zones réceptrices de l'éclatement. Lorsque des délimiteurs ont été spécifiés, on peut les recueillir dans les zones nom-donnée-41, nom-donnée-i1 et compter le nombre de caractères transférés dans nom-donnée-42 et nom-donnée-i2.

Exemple 2-12

```
77 zone     PIC X(12).
77 nom      PIC X(5).
77 an       PIC X(4).

UNSTRING zone                                  | zone = ANNEE□1974□□
      DELIMITED BY SPACE INTO nom an.          | nom  = ANNEE
                                               | an   = 1974
```

TRANSFORM

Effet : Convertit certains caractères selon une table définie.

Format : **`TRANSFORM`** `champ-de-donnée`

```
CHARACTERS FROM { nom-donnée-1     } TO { nom-donnée-2     }.
               { littéral-1        }    { littéral-2        }
               { constante-fig-1   }    { constante-fig-2   }
```

Règles :

- `TRANSFORM` ne traite que des champs **non numériques**.
- Nom-donnée-1 ou 2 désigne une zone contenant les caractères de transformation. Il s'agit, en fait, de bâtir une table de transformation des caractères un à un.
- En principe, la suite de caractères-1 doit être de même longueur que la suite de caractères-2 pour qu'il n'y ait pas d'ambiguïté sur les transformations de caractères à réaliser. Toutefois, caractère-2 peut se réduire à un seul caractère et ce sont alors tous les caractères de l'ensemble-1 qui sont transformés en caractère-2.

Exemple 2-13

```
77 ZONE PIC X(8) VALUE "18301840".
77 ZONA PIC X(3) VALUE "834".
77 ZONB PIC X(3) VALUE "967".

TRANSFORM ZONE FROM ZONA TO ZONB.              | ZONE = "19601970"
MOVE "84142" TO ZONE.
TRANSFORM ZONE FROM '1248' TO 'BLOC'.          | ZONE = "COBOL□□□"
MOVE "□□□□1248" TO ZONE.
TRANSFORM ZONE FROM SPACE TO ZERO.             | ZONE = "00001248"
TRANSFORM ZONE FROM "123456789" TO ZERO.       | ZONE = "00000000"
```

La condition

IF

Format : **IF** condition [**THEN**] { phrase-imp-1 / **NEXT SENTENCE** }

[**ELSE** { phrase-imp-2 / **NEXT SENTENCE** }] [**END-IF**].

Règles :

- Si la condition est remplie, ce sera la phrase-impérative-1 qui sera exécutée, sinon la phrase-impérative-2.
- NEXT SENTENCE implique l'exécution de l'instruction consécutive à la phrase conditionnelle.
- La portée d'une instruction IF se limite soit à la locution END-IF de même niveau d'imbrication, soit au point. En cas d'imbrication sa portée se limite à la phrase ELSE associée à l'instruction IF du niveau supérieur.
- Dans le cas de conditions imbriquées, chaque ELSE est rapporté au IF qui le précède immédiatement. Les ambiguïtés peuvent être levées si l'on précise toujours ELSE NEXT SENTENCE dans les conditionnelles incomplètes.
- Il est souhaitable de commencer une nouvelle ligne pour chaque phrase impérative et, pour les conditions imbriquées, d'aligner les phrases de même niveau sur une même colonne .
- On peut relier plusieurs conditions avec les opérateurs booléens suivants (classés par ordre de priorité) : NOT, AND, OR.

Opérateurs de comparaison : IS [NOT] = < >

Effet :

égal	**=**	ou	**EQUAL**
supérieur	**>**	ou	**GREATER**
inférieur	**<**	ou	**LESS**
différent	**NOT =**	ou	**NOT EQUAL**
supérieur ou égal	**NOT <**	ou	**NOT LESS**
inférieur ou égal	**NOT >**	ou	**NOT GREATER**

Règles :

- La comparaison d'opérandes non numériques se fait de gauche à droite. La comparaison se fait sur le premier caractère différent et selon l'ordre lexicographique (BLANK < 0 < 1 < ... < 9 < A < B < Z < ...).
- Dans les conditions composées l'expression la plus à gauche peut être omise si elle ne change pas.

A	B	A **AND** B	A **OR** B	**NOT (**A **AND** B)	(A **OR** B) **AND NOT** (A **AND** B)[5]
1	1	1	1	0	0
1	0	0	1	1	1
0	1	0	1	1	1
0	0	0	0	1	0

[5] Une manière de réaliser l'opération **XOR**.

Exemple 2-14

L'expression :	Correspond à :
`A > B AND NOT < C OR D`	`((A > B) AND (A NOT < C)) OR (A NOT < D)`
`A NOT EQUAL B OR C`	`(A NOT EQUAL B) OR (A NOT EQUAL C)`
`NOT A = B OR C`	`(NOT(A = B)) OR (A = C)`
`NOT (A GREATER B OR < C)`	`NOT ((A GREATER B) OR (A < C))`
`NOT(A NOT > B AND C AND NOT D)`	`NOT( ((A NOT > B) AND (A NOT > C)) AND ( NOT (A NOT > D)) )`
`X > A OR Y AND Z`	`X > A OR ( X > Y AND X > Z )`

Conditions de signe : `IS [NOT] NEGATIVE POSITIVE ZERO`

Effet : Permet de déterminer si un champ de données contient une valeur positive, négative ou nulle.

Format : `IS [NOT] { POSITIVE | NEGATIVE | ZERO }`

Règles :
- Ce format est à utiliser à la place de la comparaison à zéro, car plus lisible.

Exemple 2-15

```
IF B IS NOT ZERO THEN DIVIDE B INTO A.
IF A NOT NEGATIVE THEN COMPUTE RACINE-CARREE = A ** 0.5.
```

Conditions de classe : `IS [NOT] NUMERIC ALPHABETIC`

Effet : Permet de savoir si un champ alphanumérique contient exclusivement des données numériques ou alphabétiques.

Format : `IS [NOT] { NUMERIC | ALPHABETIC }`

Règles :
- Les champs numériques ne doivent pas être testés sur `ALPHABETIC` et les champs alphabétiques ne doivent pas être testés sur `NUMERIC`.
- Ce test devrait être réalisé pour toutes les entrées au clavier, car cela résout en général certaines erreurs d'exécution assez difficiles à détecter.

Exemple 2-16

```
IF ENTREE NOT NUMERIC
    DISPLAY "Illegal character in numeric field !!!"
```

Les noms-condition : Niveau 88

Avec les noms-condition on peut faire des branchements très élégants. Un nom-condition a la valeur booléenne "VRAI" si le champ de niveau supérieur contient une des valeurs définies dans la clause `VALUE` associée au nom-condition. Chaque nom-condition représente une condition en soi et peut être utilisé à la place d'une autre condition dans une instruction `IF` (voir Niveau 88 : Noms-condition, **page 13**).

Appel de procédures

Définition

En COBOL, une procédure est composée d'un ou plusieurs paragraphes, ou d'une ou plusieurs sections qui peuvent se trouver à n'importe quel endroit de la PROCEDURE DIVISION. Elles sont identifiées par le paragraphe (respectivement la section) de début et de fin, le compilateur indiquant l'adresse de retour dans la dernière instruction du paragraphe. Une fois l'exécution de la procédure terminée, le programme, contrairement à GO TO qui ne tient pas compte de cette adresse de retour, poursuit son déroulement immédiatement après l'instruction PERFORM.

PERFORM

Effet : Appel de procédures.

Format : **PERFORM** { nom de paragraphe / nom de section } { **THROUGH** / **THRU** } { nom de paragraphe / nom de section }.

Règles :

- Il ne faut utiliser que des paragraphes ou que des sections, et ne pas mélanger paragraphes et sections dans l'appel de procédures.
- Un paragraphe se termine là où commence le paragraphe suivant, de même une section se termine à la section suivante.
- Un paragraphe contient une ou plusieurs instructions COBOL, une section un ou plusieurs paragraphes.
- On peut appeler une procédure autant de fois que l'on veut ; on peut appeler une procédure dans une autre procédure. Dans ce dernier cas, la nouvelle procédure doit se situer soit entièrement à l'intérieur de la première, soit entièrement à l'extérieur, mais elle ne doit surtout pas chevaucher les limites définies pour la première procédure.
- Les appels de procédures se placent généralement au début de la PROCEDURE DIVISION et les procédures appelées en fin de programme. Ceci permet une meilleure lisibilité du programme.

PERFORM TIMES

Effet : La procédure appelée est exécutée autant de fois que le précise la valeur du littéral ou de nom-donnée avant TIMES.

Format : **PERFORM** procédure { nom-donnée / littéral } **TIMES.**

Règles :

- La zone de données indiquée doit être de format numérique entier et ne pas dépasser 32768.
- La modification de la zone de données dans la procédure n'aura aucun effet sur le nombre d'itérations, celui-ci étant fixé une fois pour toutes lors de l'appel.

Exemple 2-17

```
...
PROCEDURE DIVISION.
PRINCIPAL.
    ACCEPT A.
    ACCEPT N.
    PERFORM A-PUISSANCE-N.
    DISPLAY A "puissance " N " est egal a " resulat.
    STOP RUN.

A-PUISSANCE-N.
    MOVE 1 TO resultat.
    PERFORM MULTIPLICATION N TIMES.

MULTIPLICATION.
    MULTIPLY A BY resultat.
```

PERFORM UNTIL

Effet : Répétitive. Procédure exécutée tant que la condition n'est pas vérifiée.

Format :

```
PERFORM procédure-1 [ {THROUGH} procédure-2 ]
                    {THRU   }

    [ WITH TEST {AFTER  } ] UNTIL condition.
                {BEFORE }
```

Règles :

- `WITH TEST BEFORE` (par défaut), si la condition est vérifiée dès le départ, la procédure n'est pas exécutée et l'instruction `PERFORM` est sautée. Le test est fait d'abord, contrairement à ce que l'on pourrait penser en traduisant littéralement `UNTIL` par *jusqu'à ce que...*
- `WITH TEST AFTER`, la procédure est d'abord exécutée, ensuite la condition est évaluée.
- Si la condition n'est jamais vérifiée, le programme bouclera indéfiniment et ne se terminera jamais. On pensera à initialiser les champs correspondants, et l'on veillera avant tout à ce que la condition puisse être vérifiée.

Exemple 2-18

```
 ...
 PROCEDURE DIVISION.

     PERFORM INITIALISATION.
     PERFORM DICHOTOMIE UNTIL B - A < EPSILON.
     DISPLAY "Resultat = " X.
     STOP RUN.

 INITIALISATION.
* initialisation des limites d'intervalle a et b, calcul de f(a).
 ...
 DICHOTOMIE.
     COMPUTE X = (A + B) / 2.
     COMPUTE FX = ...  .
     IF FX * FA > 0 THEN MOVE  X TO A
                         MOVE FX TO FA
     ELSE MOVE  X TO B.
```

PERFORM VARYING

Effet : Exécution multiple d'une procédure avec, à chaque itération, incrémentation ou décrémentation d'une variable compteur. On peut imbriquer plusieurs variables-compteurs. La procédure est répétée tant que la condition n'est pas vérifiée. Nom-donnée-1, 2 et 3 sont les variables compteur dont les valeurs de départ sont respectivement nom-donnée-12, 22 et 32. Ces variables sont incrémentées respectivement de nom-donnée-13, 23, 33 jusqu'à ce que les conditions 1, 2 et 3 soient vérifiées. A l'utilisation de plusieurs variables, une variable-1 est incrémentée (ou décrémentée) seulement si la variable-2 a déjà parcouru toutes ses valeurs et que la condition pour la variable-2 est vérifiée. Dans ce cas, variable-1 prend la valeur suivante, et variable-2 est réinitialisée.

Format :

```
PERFORM procédure [ WITH TEST { AFTER  } ]
                              { BEFORE }

    VARYING nd1 FROM { lit1 } BY { lit1 } UNTIL cond1
                     { nd2  }    { nd2  }

[ AFTER nd2 FROM { nd22 } BY { nd23 } UNTIL cond2 ]
                { lit22 }    { lit23 }

[ [ AFTER nd3 FROM { nd32 } BY { nd33 } UNTIL cond3 ] ]
                   { lit32 }    { lit33 }
```

Règles :

- Tous les champs de données doivent être numériques.
- Un incrément ne peut pas avoir la valeur zéro.
- Les valeurs de tous les paramètres sont fixées avant la première exécution de la procédure, c'est-à-dire qu'une modification des variables de l'instruction PERFORM n'aura aucune incidence sur le déroulement du programme.
- Après l'exécution de l'instruction PERFORM tous les champs de données contiennent leur dernière valeur d'itération.

Exemple 2-19

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FACT.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

77  N        PIC 9(6).
77  I        PIC 9(6).
77  resultat PIC 9(18) VALUE 1.

PROCEDURE DIVISION.
PRINCIPAL.
    ACCEPT N.
    PERFORM FACTORIELLE VARYING I FROM 2 BY 1 UNTIL I > N.
    DISPLAY "Factorielle de " N " égale " resultat.
    STOP RUN.

FACTORIELLE.
    MULTIPLY I BY resultat.
```

EXIT

Effet : Permet définir un paragraphe vide à un endroit donné du programme.

Format : **`EXIT`**.

Règles :
- La clause `EXIT` doit être précédée d'un nom de paragraphe.
- Dans ce paragraphe il ne doit pas y avoir d'autres instructions.
- Si aucune procédure n'a été appelée, `EXIT` est considéré comme du commentaire et le programme se poursuit linéairement.

Exemple 2-20

```
...
    IF CHOIX = ...
       PERFORM Cherche-Client THROUGH Fin-Recherche.
...
Cherche-Client.
    DISPLAY "Recherche d'un client".

Entree-no-compte.
    DISPLAY "Numéro de compte ?".
    ACCEPT no-compte.
    PERFORM Lire-Fichier.
    IF status-fichier > 0 PERFORM Affiche-Message-Erreur
                          GO TO Fin-Recherche.
Affiche-Client.
...
Fin-Recherche.
    EXIT.
```

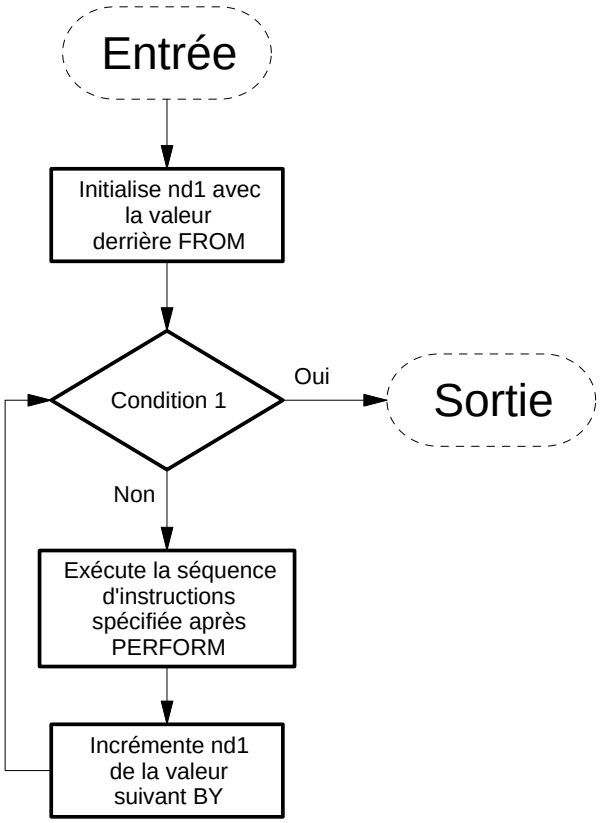
Récapitulation

Ce paragraphe a l'ambition d'apporter plus de clarté sur l'utilisation du verbe `PERFORM`. Les difficultés se situent à plusieurs niveaux et nous allons donner quelques explications et conseils qui, nous l'espérons, faciliteront la programmation des appels de procédures.

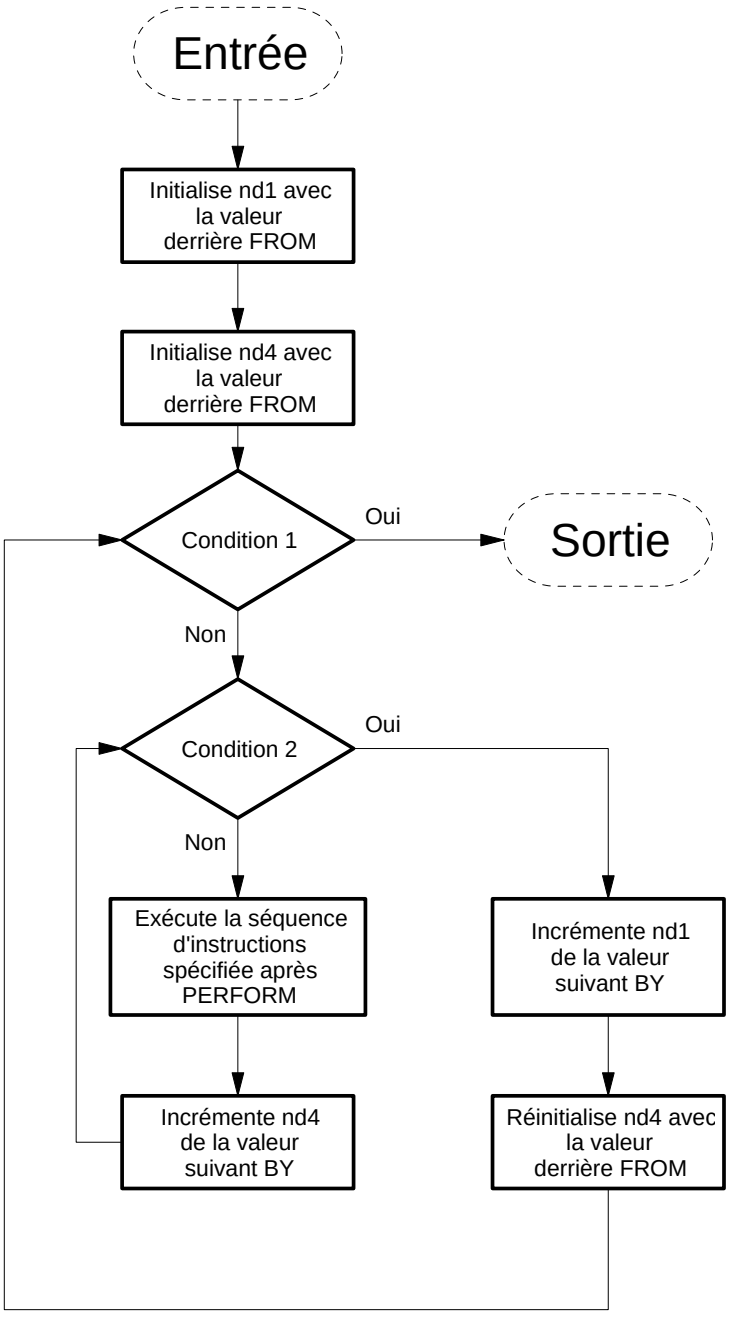
PERFORM est difficile à assimiler parce qu'on a l'habitude des boucles **for** ou **while** du langage C où les instructions sont clairement délimitées par des accolades. En COBOL l'adresse de retour est stockée sur la dernière instruction du paragraphe appelé. Comme cette adresse n'est pas visible, une technique consiste à réserver systématiquement un paragraphe de nom `FIN-PROCEDURE-BIDULE` qui contiendra l'instruction `EXIT`. Aussi, si l'on veut arrêter la procédure `BIDULE` (dans le cas d'une erreur, par exemple), on fera un `GO TO FIN-PROCEDURE-BIDULE`[6]. Si d'un autre côté la procédure n'est pas appelée par un `PERFORM`, le programme se poursuivra normalement de façon linéaire.

En ce qui concerne la phrase `PERFORM VARYING AFTER`, qui correspond à deux boucles **while** imbriquées, elle est plutôt *sophistiquée*. Si l'on tient encore compte de `TEST AFTER` et de `TEST BEFORE`, les organigrammes suivants seront sans doute une aide précieuse.

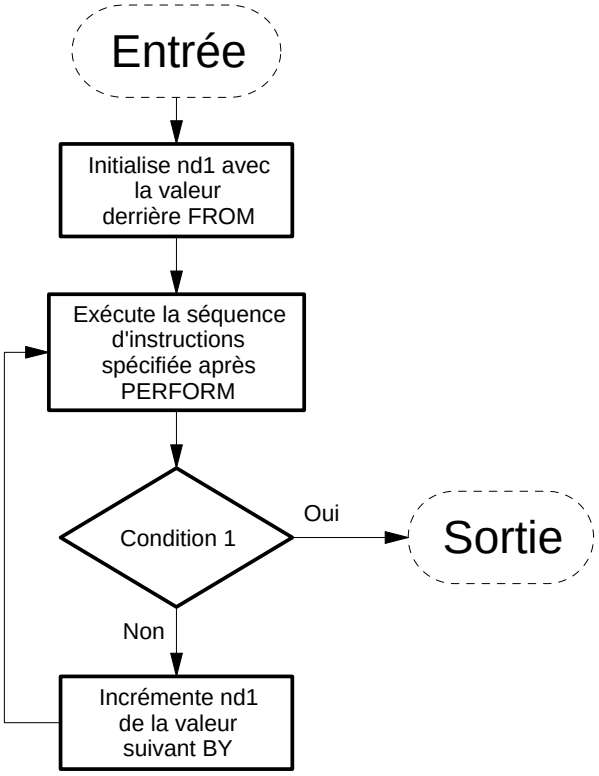
[6] On a eu beau chercher, on n'a pas trouvé d'autre nom.



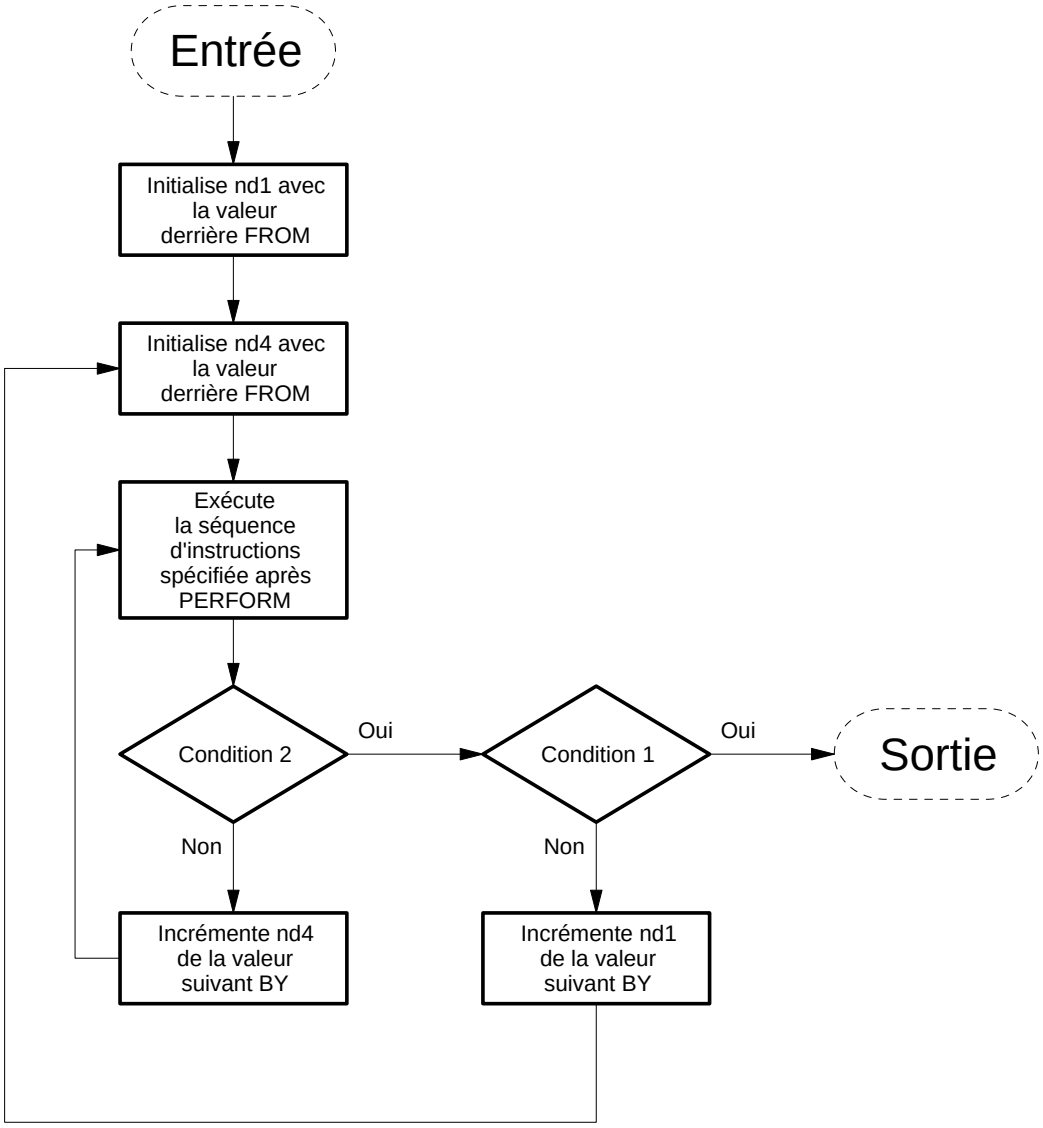
Option VARYING de la phrase PERFORM avec une condition et TEST BEFORE



Option VARYING de la phrase PERFORM avec deux conditions et TEST BEFORE



Option VARYING de la phrase PERFORM avec une condtion et TEST AFTER



Option VARYING de la phrase PERFORM avec deux conditions et TEST AFTER

Modules

CALL

Effet : Appel de module. Le module est un autre fichier programme compilé séparément (on ne traitera ici que le cas de modules écrits en COBOL).

Format :
```
CALL nom-fichier USING nom-donnée-1 nom-donnée-2 ...
```

Règles :

- Le module appelé doit définir dans la `LINKAGE SECTION` toutes les données partagées par le programme principal et le sous-programme. La `LINKAGE SECTION` est limitée à 15 descriptions de niveau 01 ou 77.
- Les données déclarées doivent correspondre exactement à leur homologues dans le module appelant, où elles sont définies dans la `WORKING-STORAGE SECTION`. Une manière élégante de procéder est d'inclure ces définitions dans les deux fichiers à l'aide d'un `COPY`. Ainsi, une modification de la structure de données n'aura que peu ou pas d'incidences sur l'appel du module.
- Après `CALL` il faut toujours envoyer toutes les données prévues à cet effet, même si l'on ne s'en sert pas pour un traitement spécifique.
- L'ordre des données lors de l'appel est déterminant et ne doit pas être changé.
- Les index de tables et les structures de fichier (`FD`) ne peuvent être pris comme paramètres, donc chaque module doit gérer ses fichiers indépendamment.
- Dans le module appelé, la `PROCEDURE DIVISION` doit être suivie de la clause :

```
USING nom-donnée-1 nom-donnée-2 etc.
```

- La `PROCEDURE DIVISION` doit comporter l'instruction `EXIT PROGRAM` au lieu de `STOP RUN` de manière à garantir le retour au programme appelant.
- Dans le module appelé, il est interdit de faire des appels récursifs (appeler le programme appelant), ceci est aussi valable avec plusieurs modules. Lorsque le premier module appelle un deuxième module, ce dernier ne doit appeler ni le programme principal, ni le programme appelant. Pour ce genre d'acrobaties il est fortement recommandé de faire un schéma d'appels.

EXIT PROGRAM

Effet : Complémentaire à CALL, retour au programme appelant.

Format :
```
nom-de-paragraphe.
    EXIT PROGRAM.
```

Règles :

- `EXIT PROGRAM` doit être la seule instruction du paragraphe.
- `EXIT PROGRAM` évolue de la même façon que `EXIT` dans un appel de procédure, c'est-à-dire que si le module est appelé par un module principal on rend la main à celui-ci, sinon `EXIT PROGRAM` est ignoré et le programme se poursuit linéairement. Si cela n'est pas souhaité, on peut utiliser `GOBACK` qui, dans ce cas, arrête l'exécution du programme.

Exemple 2-21

Programme principal

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TESTCALL.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

77  valeur       PIC X(20)  VALUE SPACE.
77  code-retour  PIC XX     VALUE SPACE.

01  type-oper    PIC 9      VALUE 1.
    88  lecture             VALUE 1.
    88  ecriture            VALUE 2.

PROCEDURE DIVISION.

Initialisation.
    MOVE "bla bla bla" TO valeur.

Appel-ss-prgm-pour-ecriture.
    MOVE 2 TO type-oper.
    CALL "sousprog" USING type-oper valeur code-retour.
    PERFORM Affiche-Operation THROUGH Affiche-Succes.

Appel-ss-prgm-pour-lecture.
    MOVE 1 TO type-oper.
    CALL "sousprog" USING type-oper valeur code-retour.
    PERFORM Affiche-Operation THROUGH Affiche-Succes.

Affichage-Valeur-recuperee.
    IF code-retour = "00"
        DISPLAY "Resultat : " valeur.

Fin-du-programme.
    STOP RUN.

Affiche-Operation.
    IF lecture
        DISPLAY "Lecture  : " WITH NO ADVANCING
    ELSE
        DISPLAY "Ecriture : " WITH NO ADVANCING.

Affiche-Succes.
    IF code-retour = "00"
        DISPLAY "OK."
    ELSE
        DISPLAY "ERREUR " code-retour.
```

Sous programme

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SOUSPROG.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

LINKAGE SECTION.

77  valeur       PIC X(20)  VALUE SPACE.
77  code-retour  PIC XX     VALUE SPACE.

01  type-oper    PIC 9.
    88  lecture             VALUE 1.
    88  ecriture            VALUE 2.

PROCEDURE DIVISION USING type-oper valeur code-retour.

Principal.
   IF  lecture  PERFORM Lit.
   IF  ecriture PERFORM Ecrit.

Retour.
   EXIT PROGRAM.

Lit.
   DISPLAY SPACE.
   DISPLAY "Entrez du texte, svp.".
   ACCEPT valeur.
   IF valeur NOT EQUAL SPACE
       MOVE ZEROES TO code-retour
   ELSE
       MOVE "-1" TO code-retour.

Ecrit.
   DISPLAY valeur.
   MOVE ZEROES TO code-retour.
```

Le traitement des erreurs

Aide à la mise au point

WITH DEBUGGING MODE

Effet : Exécute les lignes marquées d'un D en septième colonne.

Format : **SOURCE-COMPUTER.** BECANE **WITH DEBUGGING MODE.**

Règles :
- On marquera d'un D en colonne 7 les lignes contenant les instructions pour la mise au point.
- Toute instruction COBOL valide peut être utilisée. En l'absence de la clause WITH DEBUGGING MODE ces lignes sont considérées comme du commentaire.

DECLARATIVES

Effet : Paragraphe spécial dans la PROCEDURE DIVISION à utiliser pour le traitement des incidents d'entrées-sorties et pour la mise au point du programme.

Format :

```
PROCEDURE DIVISION.
DECLARATIVES.

DEC1 SECTION. USE FOR DEBUGGING ON { nom-condition
                                   | [ALL REFERENCES OF] nd1
                                   | fichier
                                   | procérure
                                   | ALL PROCEDURES }.

...
END DECLARATIVES.
Début du programme.
```

Règles :
- Avant tout, il faut positionner la variable d'environnement suivante :

  COBSW=+D

- De façon générale la section sera exécutée à chaque référence de l'objet spécifié dans la PROCEDURE DIVISION.
- Si l'objet cité dans la phrase USE est un nom de fichier, la section est exécutée après toute instruction OPEN, CLOSE, READ, DELETE ou START.
- Si l'objet est un nom de procédure, la section est exécutée avant de passer dans cette procédure.
- Si l'objet est un nom de donnée, la section est exécutée immédiatement après chaque instruction y faisant référence, sauf pour WRITE ou REWRITE.
- Si la locution ALL REFERENCES n'est pas utilisée, l'exécution de la section ne se fait qu'à chaque changement de la variable spécifiée.
- Dans une section de déclaratives, on ne peut pas faire référence à une autre procédure définie hors de la section.

READY / RESET TRACE

Effet : Traçage du programme. Dès la rencontre de `READY TRACE`, chaque nom de procédure exécutée sera affiché à l'écran. L'instruction `RESET TRACE` arrête le traçage du programme.

Format :

```
PROCEDURE DIVISION.
...
READY TRACE.
...                  } partie du programme qui sera tracée.
RESET TRACE.
...
```

Règles :

- Le traçage n'est possible que si l'on précise à la compilation :

```
COBOL nom-fichier TRACE
```


163 : Illegal character in numeric field™

Signification[7]

Par défaut la valeur que l'on entre dans un champ numérique est testée sur sa numéricité. Si le programme s'aperçoit qu'un caractère n'est pas numérique, il génère cette erreur. Celle-ci peut aussi se produire si l'on affecte une zone **non initialisée** à un champ numérique, car ce dernier est alors rempli avec des espaces et la zone initiale sera considérée comme **non numérique**.

Solutions

L'année dernière, nous recherchions une option du compilateur qui invalide la vérification de classe sur les champs numériques. Aujourd'hui, nous l'avons découverte, mais les puristes vous conseilleront de procéder dans l'ordre suivant :

1. Initialisation de toutes les variables numériques dans la `WORKING-STORAGE SECTION` (clause `VALUE`).
2. ou initialisation de toutes les variables numériques dans la `PROCEDURE DIVISION` (avant traitement).
3. Test de numéricité sur tous les `ACCEPT` de champs destinés à un traitement numérique.
4. Décomposition des instructions `READ INTO` en `READ + MOVE TO` si les champs concernés sont numériques.
5. Vérification des affectations et pointage des zones de format édité.
6. Remplacement des caractères de tabulation et de contrôle par des espaces. Ces caractères provoquent en général les erreurs les plus inexplicables. Pour repérer ces caractères, le plus simple est encore d'écrire un petit utilitaire (en C, par exemple) qui les remplace automatiquement.
7. En dernier recours, positionnez la variable d'environnement : **COBSW=-F**

[7] Demandez à un deuxième année ce qu'il en pense : 💣☠*#??!!

Les tableaux

Tableaux et dimensions

Introduction

Avec le COBOL MICRO FOCUS il est possible de créer des tableaux jusqu'à 16 dimensions, où chaque élément peut être un champ ou un groupe de données. Pour que la machine puisse implanter un tableau en mémoire, elle a besoin de savoir exactement combien de dimensions et d'éléments il comporte. Pour reconnaître un élément dans le tableau, il faut lui donner la position dans le tableau et cela pour chaque dimension. Dans un tableau à trois dimensions on aura donc trois indicateurs de position. Les tableaux sont particulièrement indiqués pour le traitement rapide d'un nombre peu important de données. Pour gérer des quantités massives de données il vaut mieux utiliser les fichiers.

Définition de tableaux

OCCURS

Effet : Définition d'un tableau. Une zone de données-1 est répétée autant de fois que le précise le littéral après `OCCURS`. La clause `KEY` indique comment le tableau est trié pour la recherche rapide (`SEARCH ALL`), la clé étant : nom-donnée-2 (`ASCENDING` : croissant, `DESCENDING` : décroissant). La clause `INDEXED BY` implique le traitement du tableau selon la méthode des index, sinon la méthode appliquée est celle des indices.

Format :

```
OCCURS [lit1 TO] lit2 TIMES [DEPENDING ON nd1]

[ {ASCENDING } KEY IS nd2 ] [INDEXED BY nom-index] ...
[ {DESCENDING}            ]
```

Règles :

- lit1 et lit2 doivent être de format numérique compris entre 0 et 32767, et lit1 < lit2.
- La clause `OCCURS` ne peut être spécifiée dans une description de données de taille variable (`DEPENDING ON`).
- La clause `OCCURS` n'est pas autorisée avec les niveaux 01, 66, 77 ou 88.
- La clause `OCCURS` et la clause `VALUE` s'excluent mutuellement.
- La clause `PICTURE` peut figurer après `OCCURS` uniquement si la zone de données est élémentaire.
- L'ordre des clauses `OCCURS` et `PICTURE` n'a pas d'importance.
- Si le tableau doit être traité selon la méthode des indices, la clause `INDEXED BY` ne doit pas figurer dans la description de données.
- Si le tableau doit être traité selon la méthode des index la clause `INDEXED BY` doit figurer dans la description de données. Les noms d'index sont automatiquement définis en binaire par le système et ne doivent pas, par conséquent, être définis dans la `WORKING-STORAGE SECTION`.
- Si la clause `KEY` est précisée, alors il est convenu que le tableau soit trié sur nom-donnée-2, mais c'est le programmeur qui est responsable du tri.

Exemple 4-1

Exemple d'instructions	Commentaires
01 VECTEUR. 02 A PICTURE 9(5) OCCURS 5.	a) Définition d'un tableau à une dimension de nom VECTEUR comportant 5 positions.
01 ADRESSES. 02 CHAMP-ADRESSES OCCURS 200. 03 nom PIC X(15). 03 prenom PIC X(15). 03 rue PIC X(20). 03 lieu-de-residence. 04 c-postal PIC 9(5). 04 ville PIC X(20).	b) Définition d'un tableau à une dimension de nom ADRESSES comportant 200 champs d'adresses.
01 EMPLOI-DU-TEMPS. 02 jour OCCURS 5. 03 heure OCCURS 8 PIC X(20).	c) Définition d'un tableau à deux dimensions, EMPLOI-DU-TEMPS, pour cinq jours et 8 heures par jour. Le nom des matières comporte 20 caractères.
01 TAB. 02 A OCCURS 5. 03 B OCCURS 20. 04 C OCCURS 10 PIC X(5).	d) Définition d'une matrice à trois dimensions qui comporte $5 \times 20 \times 10 = 1000$ éléments. Chaque élément a 5 positions alphanumériques, ce qui fait en tout 5000 caractères alphanumériques dans le tableau.

Les indices

Effet : Pour traiter un tableau avec la méthode des indices, la clause INDEXED BY ne devrait pas figurer après la clause OCCURS. L'accès à un élément du tableau se fait en indiquant le champ de données concerné suivi de la position de l'élément dans le tableau entre parenthèses ; c'est l'indice. Ce dernier peut être donné soit directement par un littéral, soit indirectement par une variable.

Format :

```
nd1 ( {lit 1} [ {lit2} [ {lit3} ] ] )
    ( {nd2  }   {nd3 }   {nd4 }     )
```

Règles :

- Le nombre d'indices correspond au nombre de clauses OCCURS dans la définition du tableau.
- Les indices sont indiqués entre parenthèses et séparés par des espaces. Avant la parenthèse ouvrante et après la parenthèse fermante il doit y avoir un espace, par contre après la parenthèse ouvrante et avant la parenthèse fermante il ne doit pas y en avoir. La parenthèse fermante peut être suivie directement d'un point de fin de paragraphe.
- Tous les littéraux et noms de données doivent être de format entier numérique.
- La valeur d'un indice doit être supérieure à zéro et ne doit pas excéder la valeur précisée après la clause OCCURS.

Exemple 4-2

Exemple d'instructions	Commentaires

`DISPLAY A (1) A (2) A (3) A (4) A (5).`
ou
`DISPLAY VECTEUR.`
	a) Affichage des cinq éléments de la table VECTEUR. Dans les deux cas les valeurs sont affichées sur une ligne de l'écran.

`MOVE 15 TO I.`
`DISPLAY CHAMP-ADRESSES (I).`
`...`
`DISPLAY c-postal (I) ville (I).`
ou
`DIPLAY lieu-residence (15).`
	b) Affichage de la quinzième adresse du tableau d'adresses à l'aide d'un indice I défini en `WORKING-STORAGE SECTION`. Affichage du code postal et de la ville du quinzième champ dans le tableau.

`MOVE "COBOL" TO heure (3 5).`	c) Ajout du cours "COBOL" dans l'emploi du temps à la cinquième heure du troisième jour.

TAB
tout le tableau
A
`A (1) A (2) A (3) A (4) A(5)`
B
`B (1 1)  B (1 2)  ...  B (1 20)`
`B (2 1)  B (2 2)  ...  B (2 20)`
`...`
`B (5 1)  B (5 2)  ...  B (5 20)`
C
`C (1  1 1)  C (1  1 2)  ...  C (1  1 10)`
`C (1  2 1)  C (1  2 2)  ...  C (1  2 10)`
`...`
`C (1 20 1)  C (1 20 2)  ...  C (1 20 10)`
`...`
`C (2  1 1)  C (2  1 2)  ...  C (2  1 10)`
`C (2  2 1)  C (2  2 2)  ...  C (2  2 10)`
`...`
`C (2 20 1)  C (2 20 2)  ...  C (2 20 10)`
`...`
`C (5  1 1)  C (5  1 2)  ...  C (5  1 10)`
`C (5  2 1)  C (5  2 2)  ...  C (5  2 10)`
`...`
`C (5 20 1)  C (5 20 2)  ...  C (5 20 10)`
	d) Il existe quatre possibilités pour accéder au tableau TAB.
	Le nom de données TAB représente tout le tableau, c'est à dire les 5000 caractères du tableau. Ici aucun indice ne doit être donné.
	
	Avec A, on peut accéder à cinq champs. Chaque champ est composé de 20 × 10 éléments de chacun 5 caractères alphanumériques, en tout donc 1000 caractères. Un indice est nécessaire.
	
	Avec B, on peut accéder à 5 × 20 = 100 champs. Chaque champ comporte 10 éléments de 5 caractères.
	
	Avec C, il faut donner trois indices. C donne 1000 champs de 5 caractères.

Indexation de tableaux

Définition des index

Effet : Pour traiter un tableau avec la méthode des index, la clause `INDEXED BY` doit figurer après la clause `OCCURS`. L'accès à un élément du tableau se fait de la même façon qu'avec un indice.

Format : `nom de donnée ( index-1 [index-2 [index-3]] )`

Règles :

- Si l'on prévoit de gérer un tableau à plusieurs dimensions avec la méthode des index, chaque dimension doit être pourvue de la clause `INDEXED BY`.
- Il y a un index par clause `INDEXED BY`, un tableau à une dimension aura donc un index, un tableau à deux dimensions deux index, etc.
- En parlant d'index, on entend notamment le nom de l'index de la clause `INDEXED BY` correspondante auquel on peut éventuellement retrancher ou ajouter un littéral au format entier numérique (**Index relatifs, page 51**).
- Les index sont séparés par des espaces ou par des virgules.
- La valeur d'un index doit être de format entier numérique supérieur à zéro et ne doit pas dépasser les limites du tableau définies par `OCCURS`.
- La valeur d'un index ne peut être donnée ou modifiée que par l'instruction `SET` ; les opérations directes sur les index ne sont pas autorisées.
- Les index ne doivent pas être redéfinis dans la `WORKING-STORAGE SECTION` car ceci se fait automatiquement par l'ajout de la clause `INDEXED BY`.

Exemple 4-3

Exemple d'instructions	Commentaires

`01 VECTEUR.`
`  02 A PICTURE 9(5) OCCURS 5`
`  INDEXED BY I.`
	a) Définition d'un tableau à une dimension de nom `VECTEUR` comportant 5 positions.

`01 ADRESSES.`
`  02 CHAMP-ADRESSES OCCURS 200`
`  INDEXED BY I.`
`  03 nom PIC X(15).`
`  03 prenom PIC X(15).`
`  03 rue PIC X(20).`
`  03 lieu-de-residence.`
`  04 c-postal PIC 9(5).`
`  04 ville PIC X(20).`
	b) Définition d'un tableau à une dimension de nom `ADRESSES` comportant 200 champs d'adresses.

`01 EMPLOI-DU-TEMPS.`
`  02 jour OCCURS 5 INDEXED BY I.`
`  03 heure OCCURS 8 INDEXED BY J`
`  PICTURE X(20).`
	c) Définition d'un tableau à deux dimensions, EMPLOI-DU-TEMPS, pour cinq jours et 8 heures par jour. Le nom des matières comporte 20 caractères.

`01 TAB.`
`  02 A OCCURS 5 INDEXED BY I.`
`  03 B OCCURS 20 INDEXED BY J.`
`  04 C OCCURS 10 INDEXED BY K`
`  PICTURE X(5).`
	d) Définition d'une matrice à trois dimensions qui comporte $5 \times 20 \times 10 = 1000$ éléments. Chaque élément a 5 positions alphanumériques, ce qui fait en tout 5000 caractères alphanumériques dans le tableau.

Traitement des index (SET)

Effet : Alors qu'avec la méthode des indices on pouvait utiliser les instructions de base du COBOL pour initialiser et modifier ces indices, les index nécessitent des instructions particulières, ceci étant dû à leur format binaire interne. Pour le traitement des index il faut utiliser l'instruction `SET`.

Format :

```
SET { nom-index  } TO { nom-index  }.
    { nom-donnée }    { nom-donnée }
                      { littéral   }
```

ou

```
SET nom-index ... { UP   } BY { nom-donnée }.
                  { DOWN }    { littéral   }
```

Règles :

- Tous les noms d'index doivent être définis dans une clause `INDEXED BY`.
- Tous les littéraux doivent être de format entier numérique.
- Rappelons que la valeur d'un index doit être supérieure à zéro et ne doit pas dépasser les limites du tableau définies par `OCCURS`.

Index relatifs

Si l'on ajoute ou si l'on soustrait à l'index un littéral numérique, alors on parle d'indexation relative.

Exemple 4-4

On veut faire la somme des carrés $1^2 + 2^2 + ... + I^2$ pour I allant de 1 à 100. Les valeurs seront stockées dans un tableau indexé. Pour le calcul nous avons besoin d'une autre variable II, car les index ne sont pas permis dans l'instruction `COMPUTE`. La méthode utilisée travaille récursivement en ajoutant à la dernière somme des carrés le terme I^2 ($=II^2$).

```
...
WORKING-STORAGE SECTION.

01  II PICTURE 999.

01  TABLEAU.
    02 SOMME OCCURS 100 INDEXED BY I PICTURE 9(8).

PROCEDURE DIVISION.
TRAITEMENT.
      MOVE 1 TO SOMME (1).
      PERFORM ADDITION VARYING I FROM 2 BY 1 UNTIL I > 100.
      STOP RUN.
ADDITION.
      SET II TO I.
      COMPUTE SOMME (I) = SOMME (I - 1) + II * II.
```

Recherche dans un tableau (SEARCH)

Effet : Le traitement indexé permet des recherches rapides dans un tableau. Pour ce faire on utilise une des deux formes de SEARCH, la première étant une recherche linéaire, la seconde une recherche binaire (ou dichotomique) qui nécessite un tableau trié. Dans la recherche linéaire, l'index est incrémenté jusqu'à ce que la condition indiquée après WHEN soit vérifiée. Si la condition n'est vérifiée pour aucune des valeurs du tableau, c'est la phrase impérative suivant AT END qui est exécutée.

Format :

```
SEARCH nd1 [ VARYING { nom-index } ] [AT END phrase-imp .]
                     { nd2       }

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

ou

```
SEARCH ALL nom-de-donnée [AT END phrase impérative ]

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

Règles :

- Le nom de donnée indiqué après SEARCH doit être défini en WORKING-STORAGE SECTION avec les clauses OCCURS et INDEXED BY.
- La recherche se fait dans tous les cas sur le premier index de la clause INDEXED BY. La clause VARYING permet d'ajouter un compteur additionnel ou un autre index qui sera incrémenté durant la recherche.
- La recherche commence avec la valeur actuelle de l'index du tableau.
- La phrase impérative figurant après la clause AT END est exécutée si la recherche a été infructueuse.
- La clause NEXT SENTENCE implique la poursuite normale du programme après une recherche infructueuse.
- Dans le cas d'une recherche binaire, le programmeur est responsable du tri du tableau.

Remarque :

Alors que pour une recherche linéaire sur N éléments il faut au plus N itérations, la recherche binaire (dichotomique) permet de trouver le résultat dans le même tableau trié après au plus $LOG_2(N)$ itérations. Ceci est particulièrement intéressant si l'on n'est pas sûr de trouver la valeur recherchée. En prenant l'exemple d'un tableau de 1000 éléments, la recherche binaire ne ferait que 10 itérations contre 1000 pour la recherche linéaire.

Exemple 4-5

Recherche linéaire

```
...
01  TABLEAU.
    02  CHAMP-ADRESSES OCCURS 1000 INDEXED BY I.
        03  nom         PIC X(10).
        03  prenom      PIC X(10).
        03  rue         PIC X(20).
        03  lieu        PIC X(20).
...
SEARCH CHAMP-ADRESSES AT END DISPLAY "Nom non trouvé !"
      WHEN nom (I) = "DUPONT" DISPLAY CHAMP-ADRESSES.
```

Recherche binaire (dichotomique)

```
...
01  TABLEAU.
    02  CHAMP-ADRESSES OCCURS 1000 INDEXED BY I
            ASCENDING KEY IS nom.
        03 nom          PIC X(10).
        03 prenom       PIC X(10).
        03 rue          PIC X(20).
        03 lieu         PIC X(20).
...
SEARCH ALL CHAMP-ADRESSES AT END DISPLAY "Non trouvé !"
      WHEN nom (I) = "DUPONT" DISPLAY CHAMP-ADRESSES.
```

Différences entre indices et index

Pour décider d'un traitement par indices ou pas index on tiendra compte des points suivants :

- Le traitement des tableaux par la méthode des index est environ 40 % plus rapide que par les indices[8].
- Les index sont définis automatiquement par la clause `INDEXED BY`, alors que les indices doivent être définis normalement en `WORKING-STORAGE SECTION`.
- Pour le traitement des index il faut utiliser une instruction spécifique : `SET`.

[8] Si toutefois ceux-ci ne sont pas déclarés en `USAGE COMPUTATIONAL`.

Les fichiers

Organisation

Les différentes formes d'organisation

Séquentielle

Les éléments sont physiquement représentés de manière consécutive et sont généralement contigus (bandes, cartes). L'accès est uniquement séquentiel.

Séquentielle indexée

Les éléments sont encore consécutifs mais il existe une table d'index permettant d'accéder directement à chaque enregistrement. Ce type de rangement permet d'accéder au fichier soit séquentiellement, soit directement par une clé.

Directe, relative ou sélective

Il existe encore un index (ou clé), à travers lequel on recherche l'élément désiré. Le rangement physique peut être partiellement consécutif mais on ne peut en tenir compte. Les éléments sont soit chaînés entre eux, soit rangés à des adresses connues que l'on sait calculer (hash code).

Chaînée

Le lien de chaînage est ici l'élément important. Il peut y en avoir plusieurs (chaînage et chaînage inverse, ou divers liens et divers niveaux). L'accès possible est évidemment obtenu en suivant les liens de chaînage selon un algorithme donné, la clé, si elle existe, servant à reconnaître l'élément cherché.

Accès et modifications

Accès

Parmi les opérations sur les fichiers il y a celles qui concernent le fichier entier (création, suppression, tri) et celles qui concernent quelques articles (consultation, modification, adjonction, suppression d'un article). Il faut donc se réserver des **accès au fichier** dans tous les cas, et des **accès aux articles** pour toutes les opérations du deuxième type.

Accès séquentiel

C'est le plus simple. On part d'une extrémité du fichier et on le balaie article par article. Pour retrouver un article sur un fichier non trié, il faut alors en moyenne N/2 consultations, où N est le nombre d'articles présents. Si le fichier est rangé en séquentiel, la conservation de l'adresse de l'article considéré à un instant donné fournit (table en mémoire) l'article suivant.
Si le fichier est rangé en désordre[9], mais avec un lien de chaînage par article vers son successeur, il faut décoder ce lien et calculer l'adresse du successeur.
Si le fichier est rangé avec accès par table d'index, il faut balayer toute la table, élément par élément. On voit donc que l'accès séquentiel sera souvent utilisable. Il est simple d'emploi en organisation séquentielle (consécutive) ou séquentielle indexée et plus complexe ailleurs.

[9] ou dérangé en ordre, comme vous préférez ...

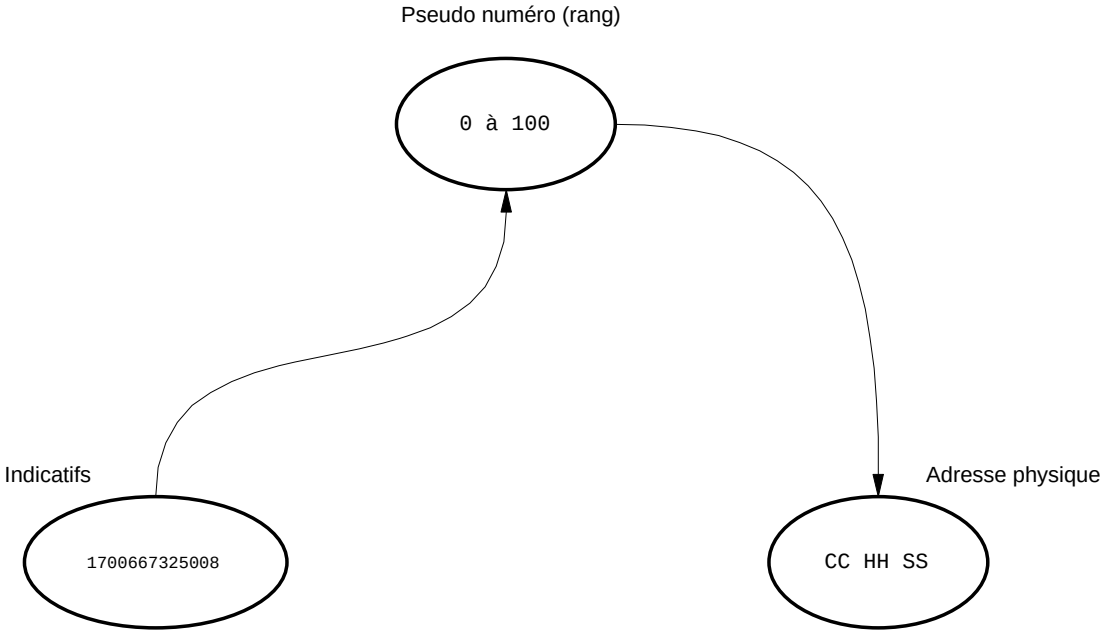
Accès dichotomique

On peut le rapprocher de l'accès séquentiel parce que, comme lui, il privilégie l'organisation consécutive. Le fichier doit être trié, et pour ne pas perdre de temps, il faut souvent pouvoir travailler sur des tables en mémoire.
L'avantage de l'accès dichotomique est de ramener le nombre moyen de consultations pour trouver un élément présent une fois par fichier à : $LOG_2(N)$. La compatibilité des tables d'index triées avec le rangement consécutif peut rendre ce mode d'accès utile. Celui-ci est associé au rangement séquentiel ou à un rangement quelconque avec table d'index consécutive, triée.

Accès direct

Comme son nom le laisse supposer, cet accès donne directement l'article cherché sans qu'il soit nécessaire de tenir compte du reste du fichier. Mais si le terme employé est simple, il recouvre en fait de nombreuses méthodes.
On peut accéder à l'élément soit par son adresse, soit après un calcul qui donne immédiatement cette adresse (accès calculé par adressage direct), soit après un calcul qui fournit l'index d'un élément d'une table d'adresse (accès calculé avec adressage indirect). On peut aussi y accéder en connaissant un indicatif, en ayant recherché cet indicatif dans une table. Celle-ci met en relation l'indicatif avec l'adresse de l'élément recherché.
N'approfondissons pas davantage ce sujet, afin de ne pas rendre ces éléments trop confus. Signalons seulement que ce mode d'accès peut être soit pré-défini, le programmeur n'ayant pas à reprogrammer la méthode mais seulement à utiliser le cadre fourni (séquentiel indexé et accès direct sur la table d'index), soit totalement à définir et à programmer.

Exemple 5-1



ACCES aux enregistrements d'un fichier RELATIF
Méthode du HASHING

Modifications

Mise à jour de valeurs

Elle nécessite un accès et une éventuelle modification de la longueur de la zone occupée par l'article. Si on ne dispose pas de l'espace nécessaire, il faut créer une zone de même type que la zone de débordement d'une adjonction puis supprimer l'ancien élément modifié.

Ajout

Lorsque l'organisation est consécutive, les seules adjonctions faciles à faire sont celles en tête ou en queue de fichier. Sinon, il faut créer la place soit en rompant la contiguïté par un jeu de pointeurs (adjonction en verrue), soit en recopiant le fichier. On remarque à ce propos l'allègement en terme de durée pour des fichiers triés si le adjonction le sont aussi, car ainsi l'on peur faire toutes les opérations lors d'une seule recopie (cf. fichier mouvement). Les adjonctions sur un fichier chaîné ou chaîné par morceau sont peu coûteuses puisqu'elles consistent à trouver un espace disponible et à mettre à jour des pointeurs (2 liens de chaînage). Sur un fichier aléatoire, il suffit de trouver un espace libre et de mettre à jour les moyens de le retrouver. Cela peut impliquer l'adjonction d'un élément rangé dans une table à organisation consécutive (la table d'index).

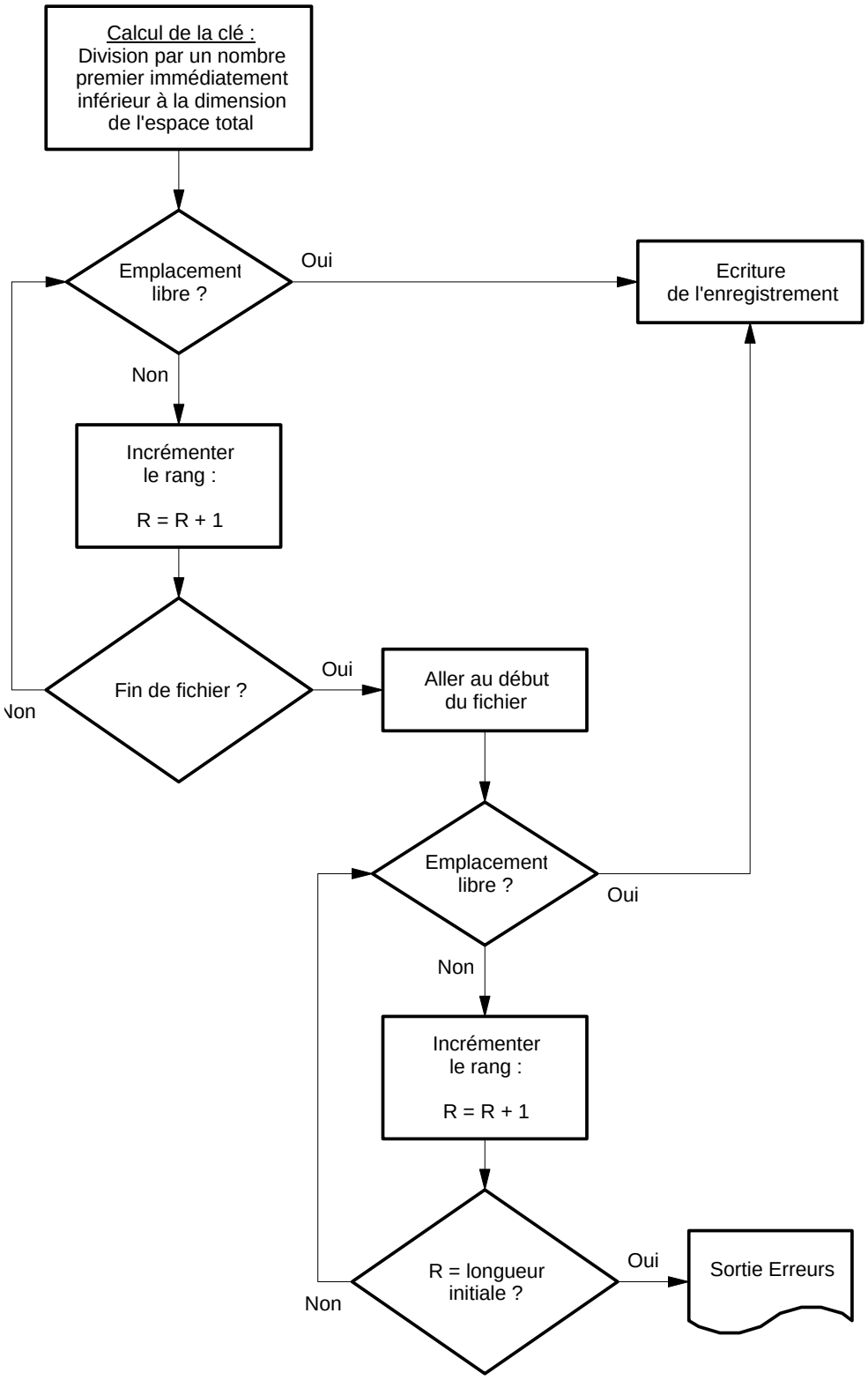
Suppression

Opération symétrique de la précédente, elle est néanmoins, en général, plus facile à réaliser car elle ne nécessite pas la recherche d'espace libre. Elle consiste souvent seulement à indiquer dans un octet prévu à cet effet que l'article associé n'existe plus. Dans le cas d'une représentation chaînée, il faut procéder à une mise à jour de pointeurs. En adoptant ce système, nous ne libérons pas vraiment l'espace occupé par le fichier. Il faut alors prévoir périodiquement une libération effective, sinon on risque de manquer de place sur le support. Sur une bande, ce sera par recopie des seuls articles vivants, sur un disque par une procédure généralement automatique de retassement (garbage collect).

Exemple 5-2 : Gestion des fichiers relatifs sous UNIX (COBOL MICRO FOCUS)

Le format des fichiers relatifs est similaire à celui des fichiers séquentiels, à l'exception d'un octet de contrôle qui se trouve à la fin de l'enregistrement. Lorsqu'on détruit un enregistrement, cet octet passe de la valeur **hex 0A** (l'enregistrement existe et peut être accédé par le programme) à **hex 00** (l'enregistrement a été détruit et ne peut pas être accédé par le programme). Cependant les données se trouvent toujours dans le fichier à leur position originale. Il est possible de les lire en déclarant le fichier comme étant un fichier séquentiel avec la taille d'un enregistrement égal à n + 1 (n est la longueur d'un enregistrement du fichier relatif). Si pour des raisons de sécurité on veut s'assurer que les données sont inaccessibles, on doit écrire par dessus les données avant de les « détruire ».

Exemple 5-3 : fichier relatif



ALGORITHME D'AJOUT DANS UN FICHIER RELATIF
méthode du hashing

Dégénérescence de l'organisation par modifications

Nous venons de voir qu'à la suite d'adjonctions, on pouvait être amené à transformer peu à peu la structure séquentielle d'un fichier, de même que les suppressions le transforment éventuellement lentement en gruyère.
Lors d'organisations directes ou chaînées, les divers éléments qui sont liés les uns aux autres peuvent finalement être très dispersés sur le support.
L'espace finit aussi par se morceler en ne laissant comme zone libre que des morceaux trop petits pour être utilisés.
La modification de taille des enregistrements du fichier consécutif, la multiplication des zones de débordement (auxquelles on accède par pointeur) et donc l'augmentation du volume que celles-ci occupent, conduisent souvent à un doublement de la taille du fichier initial.
Tous ces phénomènes de dégénérescence doivent être périodiquement éliminés par la cure de jouvence des fichiers qui est leur **réorganisation**. Cela consiste à optimiser l'implantation en fonction du nouveau fichier, à libérer l'espace disponible, à retrouver l'organisation principale choisie.

Définition de fichiers

SELECT

Effet : Chaque fichier que l'on veut utiliser ou créer dans le programme doit être défini dans la `INPUT-OUTPUT SECTION` de la `ENVIRONMENT DIVISION`. La définition se fait par l'inclusion dans le paragraphe `FILE-CONTROL` de la clause `SELECT`.

Format : *format 1 :*

```
SELECT nom-fich-interne ASSIGN TO nom-fich-sur-disque

  [ORGANIZATION IS SEQUENTIAL] [ACCESS MODE IS SEQUENTIAL]
                               [FILE STATUS IS nd2].
```

format 2 :

```
SELECT nom–fich –interne ASSIGN TO nom –fich – sur –disque

[ORGANIZATION IS {RELATIVE}] [ACCESS MODE IS {SEQUENTIAL}]
                 {INDEXED }                  {RANDOM    }
                                             {DYNAMIC   }
   [{RELATIVE} KEY IS nd1] [FILE STATUS IS nd2].
    {RECORD  }
```

Règles :

- On peut utiliser le même nom de fichier interne que le nom de fichier externe.
- Un même fichier ne peut être déclaré qu'une seule fois et à chaque déclaration doit correspondre un seul paragraphe `FD` (file description).
- Les modes d'accès `RANDOM` et `DYNAMIC` s'utilisent pour les fichiers d'organisation relative (accès direct) ou indexée. Le mode d'accès `DYNAMIC` permet en plus l'accès séquentiel.
- Les paramètres par défaut sont : organisation séquentielle, accès séquentiel.
- S'il s'agit d'un fichier indexé ou relatif, il faut préciser la clause `RELATIVE KEY` (fichier relatif) ou `RECORD KEY` (fichier indexé) suivie d'un nom-donnée-1 défini au format numérique entier non signé.

Règles :
- Le nom-donnée-2 doit être défini dans la WORKING-STORAGE SECTION au format XX. Ce champ est très important puisqu'il contiendra le statut du fichier après chaque opération (les codes d'erreurs sont décrits en annexes, **page 96**)
- En général une phrase SELECT ne tient pas sur une seule ligne ; il faut donc faire attention où l'on place les points : à la fin de la phrase, et non après chaque ligne !

Exemple 5-4

```
...
INPUT-OUTPUT SECTION.
FILE-CONTROL.

    SELECT FICHIER-SEQ ASSIGN TO "fich1"
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL
    FILE STATUS IS fichier-seq-st.

    SELECT FICHIER-REL ASSIGN TO "fich2"
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS RANDOM
    RELATIVE KEY IS JOUR
    FILE STATUS IS fichier-rel-st.

    SELECT FICHIER-IDX ASSIGN TO "fich3"
    ORGANIZATION IS INDEXED
    ACCESS IS RANDOM
    RECORD KEY IS TITRE
    FILE STATUS IS fichier-idx-st.
```

Description de fichiers

FILE DESCRIPTION (FD)

Effet : Pour chaque fichier défini dans la ENVIRONMENT DIVISION il faut décrire sa structure. Ceci se fait dans la FILE SECTION de la DATA DIVISION. Les règles sont les mêmes que pour la définition de données dans la WORKING-STORAGE SECTION.

Format :

```
FD nom-fichier-interne
    [ DATA { RECORD IS  } nom-de-donnée ]
           { RECORDS ARE}
    [ LABEL { RECORD IS  } STANDARD ] [ RECORDING MODE IS { F } ].
            { RECORDS ARE}             [                  { V } ]
```

Description des données...

Règles :
- Le nom-fichier-interne doit correspondre au nom de fichier interne de la phrase SELECT.
- La clause DATA sert à la documentation, on donne donc les noms des données et la description des enregistrements.
- La clause RECORDING spécifie la taille de l'enregistrement : F pour fixe, V pour variable. Par défaut les enregistrements sont variables.

Exemple 5-5

```
...
DATA DIVISION
FILE SECTION.

FD  FICHIER-SEQ.
01  CHAMP-SEQ.
    02  TITRE     PIC X(20).
    02  DATE1     PIC X(8).
    02  VALEUR    PIC S9(6)V99.

FD  FICHIER-REL.
01  CHAMP-REL.
    02  NOTE-1    PIC X(80).
    02  NOTE-2    PIC X(80).
    02  INFO      PIC X(80).

FD  FICHIER-IDX.
01  CHAMP-IDX.
    02  NOM       PIC X(10).
    02  PRENOM    PIC X(15).
    02  NUM-TEL   PIC X(12).
```

Exemple 5-6

```
 ...
 INPUT-OUTPUT SECTION.

 SELECT MYFILE ASSIGN TO DISK.

 DATA DIVISION.
 FILE SECTION.

 FD MYFILE VALUE OF FILE-ID IS FILE-NAME.
 01  enreg.
     02  champ PIC X(80).

 WORKING-STORAGE SECTION.
 77 FILE-NAME PIC X(25).

 PROCEDURE DIVISION.
 Saisie-nom-fichier.
      DISPLAY "Nom du fichier ? "
          WITH NO ADVANCING.
      ACCEPT FILE-NAME.

 Ouverture-du-fichier.
      OPEN OUTPUT MYFILE.
      MOVE "abcdefg" TO champ.
      WRITE enreg.

 Fermeture-du-fichier.
      CLOSE MYFILE.
      STOP RUN.
```

Rermarque

Les noms de fichiers sont fixés une fois pour toutes dans la phrase SELECT. Si l'on décide de renommer un fichier il faudra recompiler tous les programmes qui l'utilisent. Dans une entreprise qui développe des centaines de programmes cela est impensable, il faut donc trouver un moyen de gérer dynamiquement les noms des fichiers. En voici un exemple[10].

[10] Cette syntaxe est spécifique au COBOL MICRO FOCUS.

Instructions pour la gestion des fichiers

OPEN

Effet : Ouverture / création d'un fichier.

Format :

```
OPEN { INPUT  }
     { OUTPUT }
     { I - O  } nom-fichier .
     { EXTEND }
```

Règles :

- Le fichier est ouvert en lecture (`INPUT`), écriture (`OUTPUT`) ou en lecture et écriture (`I-O`). `EXTEND` est autorisé uniquement avec les fichiers séquentiels et permet d'ajouter des enregistrements en fin de fichier.
- Si le fichier n'existe pas et qu'il est ouvert en `INPUT` ou `I-O`, il est créé, autrement on provoque une erreur.
- ATTENTION : Si le fichier existe, l'ouverture en `OUTPUT` a pour conséquence la perte de toutes les données, le fichier étant réinitialisé pour permettre l'ajout de nouvelles données.

CLOSE

Effet : Fermeture d'un fichier.

Format :

```
CLOSE nom-fichier.
```

Règles :

- Les opérations d'entrées-sorties étant réalisées avec des tampons, les dernières opérations ne seront sans doute effectuées qu'après l'instruction `CLOSE`. C'est aussi à ce moment-là que sont modifiées les tables d'index. Il est donc fortement conseillé de refermer les fichiers le plus rapidement possible après chaque opération d'entrées-sorties. En effet, lors d'un arrêt impromptu du système, on risque de perdre des données si les fichiers ne sont pas fermés.

WRITE

Effet : Ecriture d'enregistrements.

Format :

```
WRITE champ de données-1 [FROM champ de données-2]

    [ {BEFORE} ADVANCING { {nd1 } [ {LINE } ] }
      {AFTER }           { {lit1} [ {LINES} ] }
                         { PAGE               } ]

  [INVALID KEY phrase impérative ].
```

Règles :

- La clause `INVALID KEY` n'est pas autorisée avec les fichier séquentiels, par contre elle est obligatoire pour les fichiers indexés et relatifs.
- La clause `FROM` permet d'économiser une instruction `MOVE` en transférant directement le champ de données-2 dans le fichier.

Règles :

- Les options `BEFORE / AFTER ADVANCING` servent à créer des fichiers d'édition (préciser `LINE SEQUENTIAL`)
- Si l'enregistrement est écrit avec succès (`FILE STATUS` = "00") le champ de données-1 n'est plus accessible, dans le cas contraire c'est la phrase impérative après `INVALID KEY` qui est exécutée.

Exemple 5-7

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADRESSES.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

    SELECT FICHIER-IDX ASSIGN TO "fich1"
    ORGANIZATION IS INDEXED
    ACCESS MODE SEQUENTIAL
    RECORD KEY IS NOM
    FILE STATUS IS fichier-idx-st.

DATA DIVISION.
FILE SECTION.

FD  FICHIER-IDX.
01  CHAMP-IDX.
    02  NOM        PIC X(10).
    02  PRENOM     PIC X(15).
    02  NUM-TEL    PIC X(12).

WORKING-STORAGE SECTION.

77  fichier-idx-st PIC XX.

PROCEDURE DIVISION.
Ouverture.
    OPEN OUTPUT FICHIER-IDX.

Saisie.
    DISPLAY "Nom = ?".
    ACCEPT NOM.

    IF NOM = "          "
        CLOSE FICHIER-IDX
        STOP RUN.

    DISPLAY "Prenom = ?".
    ACCEPT PRENOM.
    DISPLAY "Num. Tel. = ?"
    ACCEPT NUM-TEL.
    WRITE CHAMP-IDX INVALID KEY PERFORM Erreur.
    GO TO Saisie.

Erreur.
    DISPLAY "ERREUR D'ECRITURE !".
```

READ

Effet : Lecture d'enregistrements.

Format :
```
READ nom-fichier [NEXT] RECORD [INTO champ de données ]
                               { AT END phrase impérative       }
                               { INVALID KEY phrase impérative  }.
```

Règles :

- La clause AT END est conseillée avec les fichiers séquentiels. Les fichiers relatifs ou indexés peuvent utiliser soit AT END (lecture séquentielle) soit INVALID KEY (lecture par une clé).
- AT END exécute automatiquement la phrase impérative si l'on se trouve à la fin du fichier.
- La clause INTO transfère les données vers un autre champ de données, en plus du champ décrit dans la FILE SECTION. Mais ce transfert n'initialise pas correctement les données et il risque d'y avoir des problèmes de compatibilité des classes.
- La lecture séquentielle en mode d'accès DYNAMIC nécessite la locution NEXT.

AT END

Effet : Après chaque instruction de lecture sur les fichiers séquentiels ou les fichiers indexés en accès séquentiel on peut rajouter la clause AT END. Si l'on a atteint la fin de fichier, la phrase impérative est exécutée.

Format :
```
AT END phrase impérative.
```

Règles :

- Une fois la phrase impérative exécutée, il ne faut plus traiter d'enregistrement. En fait, le programme a essayé de lire la fin de fichier, cela n'a rien donné et le contenu du champ est imprévisible.
- Cette clause ne nécessite pas la définition explicite du FILE STATUS.

INVALID-KEY

Effet : Après toute instruction d'entrées-sorties sur les fichiers indexés ou relatifs on peut rajouter la clause INVALID KEY. Si l'accès à un enregistrement est sans succès, la phrase impérative est exécutée.

Format :
```
INVALID KEY phrase impérative.
```

Règles :

- Cette clause ne nécessite pas la définition explicite du FILE STATUS.

Les causes d'erreur les plus fréquentes sont :

pour les fichiers relatifs : dépassement des limites du domaine
pour les fichiers indexés : clé double, inconnue, non consécutive en écriture séquentielle

Exemple 5-8

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LITREL.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

    SELECT FICHIER-REL ASSIGN TO "fich2"
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS RANDOM
    RELATIVE KEY IS JOUR
    FILE STATUS IS fichier-rel-st.

DATA DIVISION.
FILE SECTION.

FD  FICHIER-REL.
01  CHAMP-REL.
    02  NOTE-1     PIC X(80).
    02  NOTE-2     PIC X(80).
    02  INFO       PIC X(80).

WORKING-STORAGE SECTION.

77  fichier-rel-st PIC XX.
77  JOUR           PIC 999.

PROCEDURE DIVISION.

Ouverture.
    OPEN INPUT FICHIER-REL.
    DISPLAY "Jour = ?".
    ACCEPT JOUR.

Recherche.
    READ FICHIER-REL INVALID KEY
        DISPLAY "ENREGISTREMENT NON TROUVE !"
        GO TO Fermeture.
    DISPLAY CHAMP-REL.

Fermeture.
    CLOSE FICHIER-REL.
    STOP RUN.
```

REWRITE

Effet : Réécriture d'un enregistrement.

Format : **REWRITE** champ de données-1 [**FROM** champ de données-2]
[**INVALID KEY** phrase impérative]**.**

Règles :

- La clause INVALID KEY n'est pas autorisée avec les fichiers séquentiels, en revanche elle est obligatoire pour les fichiers indexés et relatifs.
- Pour la réécriture d'enregistrements de fichiers d'organisation quelconque, mais de mode d'accès séquentiel, l'instruction REWRITE doit être précédée d'une instruction READ (lecture) réussie.
- La clause FROM permet d'économiser une instruction MOVE en transférant directement le champ de données-2 dans le fichier.
- Si l'enregistrement est écrit avec succès (FILE STATUS = "00") le champ de données-1 n'est plus accessible, dans le cas contraire c'est la phrase impérative après INVALID KEY qui est exécutée.

Exemple 5-9

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MODIF.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

    SELECT FICHIER-SEQ ASSIGN TO "fich1"
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL
    FILE STATUS IS fichier-seq-st.

DATA DIVISION.
FILE SECTION.

FD  FICHIER-SEQ.
01  CHAMP-SEQ.
    02  TITRE      PIC X(20).
    02  DATE1      PIC X(8).
    02  VALEUR     PIC S9(6)V99.

WORKING-STORAGE SECTION.

77  fichier-seq-st PIC XX.
77  T              PIC X(20).

PROCEDURE DIVISION.
Saisie-titre.
    DISPLAY "Titre = ?".
    ACCEPT T.

Ouverture.
    OPEN I-O FICHIER-SEQ.

Recherche.
    READ FICHIER-SEQ AT END
        DISPLAY "TITRE NON TROUVE !"
        GO TO Fermeture.
    IF TITRE NOT = T
        GO TO Recherche.

Modifier.
    DISPLAY CHAMP-SEQ.
    DISPLAY "Nouveau titre = ?".
    ACCEPT TITRE.
    DISPLAY "Nouvelle date = ?".
    ACCEPT DATE1.
    DISPLAY "Nouvelle valeur = ?".
    ACCEPT VALEUR.
    REWRITE CHAMP-SEQ.

Fermeture.
    CLOSE FICHIER-SEQ.
    STOP RUN.
```

DELETE

Effet : Effacement d'un enregistrement indexé ou relatif.

Format : **DELETE** nom-fichier RECORD [**INVALID KEY** phrase impérative].

Règles :
- L'instruction DELETE n'est pas autorisée avec les fichiers séquentiels.
- Le fichier doit être ouvert en I-O.
- La clause INVALID KEY ne doit pas figurer pour les fichiers d'organisation indexée en accès séquentiel, dans tous les autres cas elle est obligatoire.
- Dans le cas de fichiers d'organisation indexée en accès séquentiel, l'instruction DELETE doit être précédée d'une instruction READ (lecture) réussie.
- Le contenu de la structure de données décrite en FILE SECTION reste inchangé.
- Les enregistrements ne sont jamais détruits physiquement, mais marqués par un octet de contrôle (voir Gestion des fichiers relatifs sous UNIX**, page 57**).

Exemple 5-10

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DELIDX.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FICHIER-IDX ASSIGN TO "fich1"
    ORGANIZATION IS INDEXED
    ACCESS IS RANDOM
    RECORD KEY IS NOM
    FILE STATUS IS fichier-idx-st.

DATA DIVISION.
FILE SECTION.

FD  FICHIER-IDX.
01  CHAMP-IDX.
    02  NOM        PIC X(10).
    02  PRENOM     PIC X(15).
    02  NUM-TEL    PIC X(12).

WORKING-STORAGE SECTION.
77  fichier-idx-st PIC XX.

PROCEDURE DIVISION.
Ouverture.
    OPEN I-O FICHIER-IDX.

Saisie.
    DISPLAY "Nom = ?".
    ACCEPT NOM.
    DELETE FICHIER-IDX INVALID KEY PERFORM Erreur.

Fermeture.
    CLOSE FICHIER-IDX.
    STOP RUN.

Erreur.
    DISPLAY "CE NOM N'EXISTE PAS !".
```

START

Effet : Positionnement à l'intérieur d'un fichier indexé ou relatif.

Format :

```
START nom-fichier [ KEY IS { EQUAL TO
                           { =
                           { GREATER THAN
                           { >
                           { NOT LESS THAN
                           { NOT <          } nom-de-donnée ]

INVALID KEY phrase impérative.
```

Règles :

- L'instruction START n'est pas autorisée avec les fichiers séquentiels.
- Le nom de donnée doit correspondre à la zone définie pour la clé.
- Si KEY est omis, START se positionne sur la valeur actuelle de la clé.

Tableau de synthèse

Mode d'accès	Instruction	Mode d'ouverture			
		Input	Output	Input-Output	Extend (séquentiel)
Séquentiel	READ	X		X	
	WRITE		X		X
	REWRITE			X	
	START	X		X	
	DELETE			X	
Random (fichiers non séquentiels)	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
Dynamic (fichiers non séquentiels)	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

Les combinaisons de modes d'accès et d'ouverture autorisées sont marquées par des X.

Programmation avancée

Plus de convivialité

Programme de menu

Ce modeste programme de menu n'est qu'un petit exemple de ce qu'il est possible de faire en COBOL. Ainsi, en utilisant un tableau à deux dimensions, on peut créer une barre de menus déroulants contenant un nombre variable de choix.
Nous avons voulu rester simples de manière à faciliter la compréhension de cet exemple. Remarquez néanmoins la technique (subtile !) utilisée pour tracer des lignes...

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. MENU1.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 77  I                  PIC 9      VALUE ZERO.
 77  L                  PIC 99     VALUE ZERO.
 77  ligne-courante     PIC 9      VALUE 1.
 77  entree             PIC X      VALUE "A".
 77  touche             PIC X      VALUE SPACE.

 01  constantes.
     02  MAXM           PIC 9      VALUE 5.
     02  HAUTEUR-MENU   PIC 9      VALUE 4.

 01  lignes-texte                  VALUE SPACE.
     02  mess           PIC X(40).
     02  vide           PIC X(79).

 01  menu.
     02  texte OCCURS 5 PIC X(20).

 01 coordonnees.
     02 coord-aide      PIC 9(4)   VALUE 1603.
     02 coord-divers    PIC 9(4)   VALUE 1901.

 PROCEDURE DIVISION.

*****************************************************************
*                        GESTION DU MENU                        *
*****************************************************************

 Initialise-Menu.
     MOVE "  Entrées de stock  " TO texte (1).
     MOVE "  Sorties de stock  " TO texte (2).
     MOVE "  Statistiques      " TO texte (3).
     MOVE "  Inventaire        " TO texte (4).
     MOVE "  Fin du programme  " TO texte (5).

 Principal.
     PERFORM Efface-Ecran.
     PERFORM Affiche-Aide.
     MOVE "A" TO entree.
     PERFORM Choix-Menu UNTIL entree = SPACE.
     GO TO   Entrees-Stock
             Sorties-Stock
             Statistiques
             Inventaire
             Fin-Programme DEPENDING ON ligne-courante.
```

Expert

```
 Efface-ecran.
     DISPLAY SPACE WITH BLANK SCREEN.
     SUBTRACT 1 FROM HAUTEUR-MENU GIVING L.
     DISPLAY vide AT LINE L COL 1 WITH UNDERLINE.
     ADD 1 HAUTEUR-MENU TO MAXM GIVING L.
     DISPLAY vide AT LINE L COL 1 WITH UNDERLINE.

 Affiche-Menu.
     ADD HAUTEUR-MENU TO I GIVING L.
     IF I = ligne-courante
         THEN DISPLAY texte (I) AT LINE L COL 4
                  WITH REVERSE-VIDEO
     ELSE
         DISPLAY texte (I) AT LINE L COL 4.

 Affiche-Aide.
     DISPLAY SPACE.
     DISPLAY "[ Appuyez sur H / B pour HAUT / BAS ]"
         AT coord-aide.

 Choix-Menu.
     PERFORM Affiche-Menu VARYING I FROM 1 BY 1 UNTIL I > MAXM.
     MOVE SPACE TO entree.
     ACCEPT entree AT 2479 WITH NO-ECHO AUTO-SKIP.

     IF ( entree = "h" OR "H" ) AND ligne-courante > 1
         SUBTRACT 1 FROM ligne-courante
     ELSE
         IF ( entree = "b" OR "B" ) AND ligne-courante < MAXM
             ADD 1 TO ligne-courante.

*****************************************************************
*                        PARTIE PROGRAMME                       *
*****************************************************************

 Affiche-Traitement-En-Cours.
     STRING "Traitement  :" texte (ligne-courante)
         DELIMITED BY SIZE INTO mess.
     DISPLAY mess AT coord-divers WITH HIGHLIGHT.
     ACCEPT touche AT 2479 WITH NO-ECHO AUTO-SKIP.

 Entrees-Stock.
     PERFORM Affiche-Traitement-En-Cours.
*    ...
     GO TO Principal.
 Sorties-Stock.
     PERFORM Affiche-Traitement-En-Cours.
*    ...
     GO TO Principal.
 Statistiques.
     PERFORM Affiche-Traitement-En-Cours.
*    ...
     GO TO Principal.
 Inventaire.
     PERFORM Affiche-Traitement-En-Cours.
*    ...
     GO TO Principal.
 Fin-Programme.
     STOP RUN.
```

Génération de pages écran

On fait souvent le reproche au COBOL de ne pas permettre pas la création rapide d'une belle présentation. C'est effectivement le cas si l'on se met à calculer les positions sur l'écran de tous les titres et messages que l'on veut afficher. La solution la plus simple consiste à sauver des pages d'écran et de les rappeler ensuite au moment opportun. Combre de raffinement : ces pages d'écran peuvent être composées et modifiées avec un éditeur de texte.

Création des pages écran

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CREECRAN.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

SPECIAL-NAMES.
    CONSOLE IS CRT.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FICHIER-PAGES ASSIGN TO DISK "fpages.seq"
    ORGANIZATION IS LINE SEQUENTIAL
    ACCESS SEQUENTIAL
    FILE STATUS fichier-pages-st.

DATA DIVISION.
FILE SECTION.

FD  FICHIER-PAGES.
01  ligne-fichier    PIC X(79) OCCURS 24.

WORKING-STORAGE SECTION.

77  fichier-pages-st PIC XX.
77  I                PIC 99    VALUE ZERO.

01  ecran.
    02  ligne-ecran  PIC X(80) OCCURS 24.

PROCEDURE DIVISION.
Creation-Pages.
    MOVE SPACE TO ecran.
    ACCEPT ecran AT 0101.
    IF ecran = SPACE
        STOP RUN
    ELSE
        OPEN EXTEND FICHIER-PAGES
        PERFORM Ecriture VARYING I FROM 1 BY 1 UNTIL I > 24.

    CLOSE FICHIER-PAGES.
    GO TO Creation-pages.

Ecriture.
    WRITE ligne-fichier FROM ligne-ecran (I).
```

Affichage des pages écran

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LITECRAN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    CONSOLE IS CRT.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FICHIER-PAGES ASSIGN TO DISK "fpages.seq"
    ORGANIZATION IS LINE SEQUENTIAL
    ACCESS SEQUENTIAL
    FILE STATUS fichier-pages-st.

DATA DIVISION.
FILE SECTION.
FD  FICHIER-PAGES.
01  ligne-fichier     PIC X(80).

WORKING-STORAGE SECTION.

77  fichier-pages-st  PIC XX.
77  touche            PIC X       VALUE SPACE.
77  EOF               PIC 9       VALUE ZERO.
77  i                 PIC 99      VALUE ZERO.
77  L                 PIC 99      VALUE ZERO.
77  n                 PIC 9       VALUE 1.
77  ECRAN             PIC X(1920) VALUE SPACE.

01  TABLEAU.
    02  un-ecran                  OCCURS 9.
        03  une-ligne PIC X(80)   OCCURS 24.

PROCEDURE DIVISION.
Remplit-Tableau.
    OPEN INPUT FICHIER-PAGES.
    PERFORM Lire-Pages VARYING i FROM 1 BY 1
                               UNTIL EOF = 1 OR i > 9
                       AFTER L FROM 1 BY 1
                               UNTIL EOF = 1 OR L > 24.
    CLOSE FICHIER-PAGES.

Principal.
    PERFORM Affiche-Pages UNTIL n = ZERO.
    STOP RUN.

Lire-Pages.
    READ FICHIER-PAGES RECORD AT END MOVE 1 TO EOF.
    IF EOF = ZERO
        MOVE ligne-fichier TO une-ligne (i L).

Affiche-Pages.
    DISPLAY "Quelle page voulez-vous afficher (0:quitter) ?"
        WITH BLANK SCREEN HIGHLIGHT.
    ACCEPT n AT 0148 WITH NO-ECHO AUTO-SKIP.
    IF n NOT = ZERO
        MOVE un-ecran (n) TO ECRAN
        DISPLAY ECRAN AT 0101 WITH BLANK SCREEN
        ACCEPT touche AT 2479 WITH NO-ECHO AUTO-SKIP.
```

Expert

Nombres aléatoires / opérations sur les chaînes

Cet exemple utilise l'expression `STRING` et un tableau à deux dimensions. Les '+' servent de délimiteurs de chaîne, dans ce cas précis il suffit évidemment d'en mettre un seul à la fin. La variable pointeur permet de poursuivre la concaténation sur la même chaîne `ligne`.

Pour générer des nombres aléatoires nous avons pensé récupérer l'heure système ; un seul problème : le programme met moins d'un centième de seconde pour s'exécuter, donc `ACCEPT TIME` renvoie toujours la même valeur... On a donc créé une boucle de temporisation qui contient quelques petites astuces. Par exemple, le fait de multiplier par 10000000 provoque une troncature au niveau des centièmes de seconde. On divise ensuite par 1000000 pour que le nombre d'itérations ne soit pas trop important et on lui ajoute un minimum de 10 pour être à peu près sûr d'avoir dans tous les cas une valeur aléatoire[11]. Toutefois, ces valeurs dépendent de la machine[12].

```
IDENTIFICATION DIVISION.
PROGRAM-ID. POEMS.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

77  T            PIC 9(8)  VALUE ZERO.
77  N            PIC 9     VALUE ZERO.
77  G            PIC 9     VALUE ZERO.
77  poubelle     PIC 9(8)  VALUE ZERO.

77  proba-virg   PIC 9     VALUE 2.
77  proba-alin   PIC 9     VALUE 1.
77  proba-para   PIC 9     VALUE 1.
77  proba-f-ph   PIC 9     VALUE 1.

77  compteur     PIC 99    VALUE ZERO.
77  max-lignes   PIC 99    VALUE 20.
77  ligne        PIC X(79) VALUE SPACES.
77  ptr          PIC 99    VALUE 1.
77  fin-ligne    PIC 9     VALUE ZERO.

01  repertoire.
    02  groupe OCCURS 4.
        03  phrase PIC X(20) OCCURS 5.

PROCEDURE DIVISION.
Initialisation.
    MOVE "MINUIT LUGUBRE++++++" TO phrase (1 1).
    MOVE "YEUX ETINCELANTS++++" TO phrase (1 2).
    MOVE "OISEAU OU DEMON+++++" TO phrase (1 3).
    MOVE "CREATURE DU MALIN+++" TO phrase (1 4).
    MOVE "PROPHETE++++++++++++" TO phrase (1 5).

    MOVE "QUI M'ENSORCELE+++++" TO phrase (2 1).
    MOVE "QUI M'A TERRORISE+++" TO phrase (2 2).
    MOVE "TOUJOURS IMMOBILE+++" TO phrase (2 3).
    MOVE "DANS L'OBSCURITE++++" TO phrase (2 4).
    MOVE "MON AME SE CONSUME++" TO phrase (2 5).
```

Expert

[11] La valeur zéro impliquerait le renvoi par `ACCEPT TIME` des valeurs précédentes.

[12] Peut-être remarquerez-vous que le poète est plus inspiré quand ça rame...

```
    MOVE "SANS UN MOT+++++++++" TO phrase (3 1).
    MOVE "ILS M'EMPORTENT+++++" TO phrase (3 2).
    MOVE "UN CORBEAU++++++++++" TO phrase (3 3).
    MOVE "SIGNE DE SEPARATION+" TO phrase (3 4).
    MOVE "RAMPANT LENTEMENT+++" TO phrase (3 5).

    MOVE "RIEN DE PLUS++++++++" TO phrase (4 1).
    MOVE "DANS LES TENEBRES+++" TO phrase (4 2).
    MOVE "DANS LA NUIT++++++++" TO phrase (4 3).
    MOVE "... A TOUT JAMAIS+++" TO phrase (4 4).
    MOVE "SANS UN CRI+++++++++" TO phrase (4 5).

Principal.
    DISPLAY SPACE WITH BLANK SCREEN.
    MOVE ZERO TO compteur.
    PERFORM Ecrit-Ligne UNTIL compteur = max-lignes.
    STOP RUN.

Ecrit-Ligne.
    ADD 1 TO compteur.
    PERFORM Nombre-Aleatoire.
    IF N <= proba-para DISPLAY SPACE.
    PERFORM Nombre-Aleatoire.
    IF N <= proba-alin
        MOVE "    " TO ligne
        MOVE 5 TO ptr.
    MOVE ZERO TO fin-ligne.
    PERFORM Composition VARYING G FROM 1 BY 1
        UNTIL ( G > 4 ) OR ( fin-ligne = 1 ).

    DISPLAY ligne.
    MOVE SPACES TO ligne.
    MOVE 1 TO ptr.

Composition.
    PERFORM Nombre-Aleatoire.
    STRING SPACE phrase (G N)
        DELIMITED BY "+"
        INTO ligne
        WITH POINTER ptr.
    PERFORM Virgule.
    PERFORM Nombre-Aleatoire.
    IF N <= proba-f-ph MOVE 1 TO fin-ligne.

Virgule.
   PERFORM Nombre-Aleatoire.
   IF N <= proba-virg
       STRING "," DELIMITED BY SIZE
           INTO ligne
           WITH POINTER ptr.

Nombre-Aleatoire.
    ACCEPT T FROM TIME.
    DIVIDE T BY 5 GIVING poubelle REMAINDER N.
    ADD 1 TO N.
    MULTIPLY 10000000 BY T.
    COMPUTE T = T / 100000 + 10.
    PERFORM TIMER UNTIL T EQUAL ZERO.
Timer.
    SUBTRACT 1 FROM T.
```

Test de validité de la date

Voici un classique de la programmation qui devra figurer dans tous les programme ayant affaire avec des dates. Nous tenons compte des années bissextiles et nous testons si la date est supérieure à la date système – dans ce cas on affiche un message d'erreur. Si au contraire la date est valide, elle est affichée en haut à droite de l'écran.
Nous utilisons `DIVIDE GIVING REMAINDER` comme fonction modulo. Le champ `poubelle`, qui sert uniquement à respecter la syntaxe, doit être défini assez grand pour ne pas créer d'erreur de dépassement de taille (`SIZE ERROR`).

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TESTDATE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.

77  jourmax      PIC 99        VALUE ZERO.
77  pliage       PIC 99        VALUE ZERO.
77  parite       PIC 99        VALUE ZERO.
77  reste        PIC 99        VALUE ZERO.
77  mod-004      PIC 9(4)      VALUE ZERO.
77  mod-100      PIC 9(4)      VALUE ZERO.
77  mod-400      PIC 9(4)      VALUE ZERO.
77  poubelle     PIC 9(9)      VALUE ZERO.
77  date-ok      PIC 9         VALUE ZERO.
77  date-systeme PIC 9(8)      VALUE ZERO.

77  date-affiche PIC 99/99/9(4).

01  date-normale.

    02  jour     PIC 99        VALUE ZERO.
    02  FILLER   PIC X         VALUE SPACE.
    02  mois     PIC 99        VALUE ZERO.
    02  FILLER   PIC X         VALUE SPACE.
    02  annee    PIC 9(4)      VALUE ZERO.

01  date-inversee.

    02  annee    PIC 9(4)      VALUE ZERO.
    02  mois     PIC 99        VALUE ZERO.
    02  jour     PIC 99        VALUE ZERO.

PROCEDURE DIVISION.
Principal.
    DISPLAY SPACE WITH BLANK SCREEN.
    DISPLAY "PROGRAMME DE GESTION DE STOCK - SORTIES".
    MOVE ZERO TO date-ok.
    PERFORM Saisie-Date UNTIL date-ok = 1.

    MOVE date-normale TO date-affiche.
    DISPLAY date-affiche AT 0170 WITH HIGHLIGHT.
    STOP RUN.

Saisie-Date.
    DISPLAY "Date de sortie de stock :   /  /" AT 1005.
    ACCEPT date-normale AT 1031.
    PERFORM Test-Date.
```

```
Test-Date.
    ACCEPT date-systeme FROM DATE.
    ADD 19000000 TO date-systeme.

    MOVE CORRESPONDING date-normale TO date-inversee.

    IF ( mois OF date-inversee <= 12 AND >= 1 )
         AND ( date-inversee <= date-systeme )
              PERFORM Calcul-Jourmax
              IF jour OF date-inversee <= jourmax AND > ZERO
                  MOVE 1 TO date-ok.

    DISPLAY SPACE AT 1044.
    IF date-ok EQUAL ZERO DISPLAY "DATE ERRONEE..." AT 1044.

Calcul-Jourmax.
    IF mois OF date-inversee = 2
        DIVIDE annee OF date-inversee BY 4
            GIVING poubelle REMAINDER mod-004
        DIVIDE annee OF date-inversee BY 100
            GIVING poubelle REMAINDER mod-100
        DIVIDE annee OF date-inversee BY 400
            GIVING poubelle REMAINDER mod-400

        IF ( mod-004 EQUAL ZERO AND mod-100 NOT EQUAL ZERO )
          OR mod-400 EQUAL ZERO
            MOVE 29 TO jourmax
        ELSE
            MOVE 28 TO jourmax
    ELSE
        SUBTRACT 1 FROM mois OF date-inversee
        DIVIDE mois OF date-inversee BY 7
            GIVING poubelle REMAINDER pliage
        DIVIDE pliage BY 2 GIVING reste REMAINDER reste
        SUBTRACT reste FROM 31 GIVING jourmax.
```

Algorithmes de tri

Recopie dans un fichier indexé

Cette méthode permet de trier des fichier de taille infinie puisqu'on ne crée pas de table en mémoire. De plus l'algorithme est extrêmement simple car on se contente de recopier le fichier.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TRIDX.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.

FILE-CONTROL.
    SELECT F-SOURCE ASSIGN TO DISK "voitures.seq".

    SELECT F-DESTIN ASSIGN TO DISK "voitures.idx"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS IMMATRICULATION.
```

... /...

```
DATA DIVISION.
FILE SECTION.

FD  F-SOURCE.
01  champ-source.

    02  immatriculat-source PIC 9(4)AAA99.
    02  proprietaire-source PIC A(20).

FD  F-DESTIN.
01  champ-destin.

    02  IMMATRICULATION     PIC 9(4)AAA99.
    02  proprietaire-destin PIC A(20).

WORKING-STORAGE SECTION.

 01  DRAPEAU                 PIC 9 VALUE ZERO.
     88 FIN                       VALUE 1.

PROCEDURE DIVISION.
Ouverture.
    OPEN INPUT  F-SOURCE.
    OPEN OUTPUT F-DESTIN.

Lecture-et-tri.
    READ F-SOURCE AT END MOVE 1 TO DRAPEAU.
    PERFORM Transfert UNTIL FIN.
    CLOSE F-SOURCE F-DESTIN.
    OPEN INPUT F-DESTIN.

Lecture-et-affichage.
    MOVE ZERO TO DRAPEAU.
    READ F-DESTIN NEXT AT END MOVE 1 TO DRAPEAU.
    PERFORM Affichage UNTIL FIN.
    CLOSE F-DESTIN.
    STOP RUN.

Transfert.
    WRITE champ-destin FROM champ-source
        INVALID KEY DISPLAY "ERREUR".
    READ F-SOURCE AT END MOVE 1 TO DRAPEAU.

Affichage.
    DISPLAY "immatr. " IMMATRICULATION "  "
            "prop.   " proprietaire-destin.
    READ F-DESTIN NEXT AT END MOVE 1 TO DRAPEAU.
```

Bubble Sort

Ce tri a l'avantage d'être simple et utilisable dans tous les contextes. Après avoir changé quelque peu la définition des données on pourra comparer sa vitesse d'exécution avec le tri par arbre binaire présenté un peu plus loin.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TRIBULLE.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.

FILE-CONTROL.
    SELECT FICHIER ASSIGN TO DISK "voitures.seq".

DATA DIVISION.
FILE SECTION.

FD  FICHIER.
01  champ-fichier.
    02  immatriculat-fichier PIC 9(4)AAA99.
    02  proprietaire-fichier PIC A(20).

WORKING-STORAGE SECTION.

01  DRAPEAU                 PIC 9 VALUE ZERO.
    88 FIN                       VALUE 1.

01  TABLEAU.
    02  champ-tableau OCCURS 100 INDEXED BY I J HAUT ECHANGE.
        03  immatriculat    PIC 9(4)AAA99.
        03  proprietaire    PIC A(20).

PROCEDURE DIVISION.
Principal.
    OPEN INPUT FICHIER.
    READ FICHIER AT END MOVE 1 TO DRAPEAU.
    PERFORM Transfert VARYING I FROM 1 BY 1 UNTIL FIN.
    CLOSE FICHIER.

    SET I DOWN BY 1.
    SET ECHANGE TO 100.
    PERFORM Bubble VARYING HAUT FROM I BY -1 UNTIL HAUT < 2
                        AFTER J FROM 1 BY 1 UNTIL J = HAUT.
    PERFORM Affichage VARYING J FROM 1 BY 1 UNTIL J > I.
    STOP RUN.

Transfert.
    MOVE champ-fichier TO champ-tableau (I).
    READ FICHIER AT END MOVE 1 TO DRAPEAU.

Bubble.
    IF  immatriculat (J) > immatriculat (J + 1)
        MOVE champ-tableau (J)       TO champ-tableau (ECHANGE)
        MOVE champ-tableau (J + 1)   TO champ-tableau (J)
        MOVE champ-tableau (ECHANGE) TO champ-tableau (J + 1).

Affichage.
    DISPLAY immatriculat (J) "  " proprietaire (J).
```

Arbre binaire

Explications

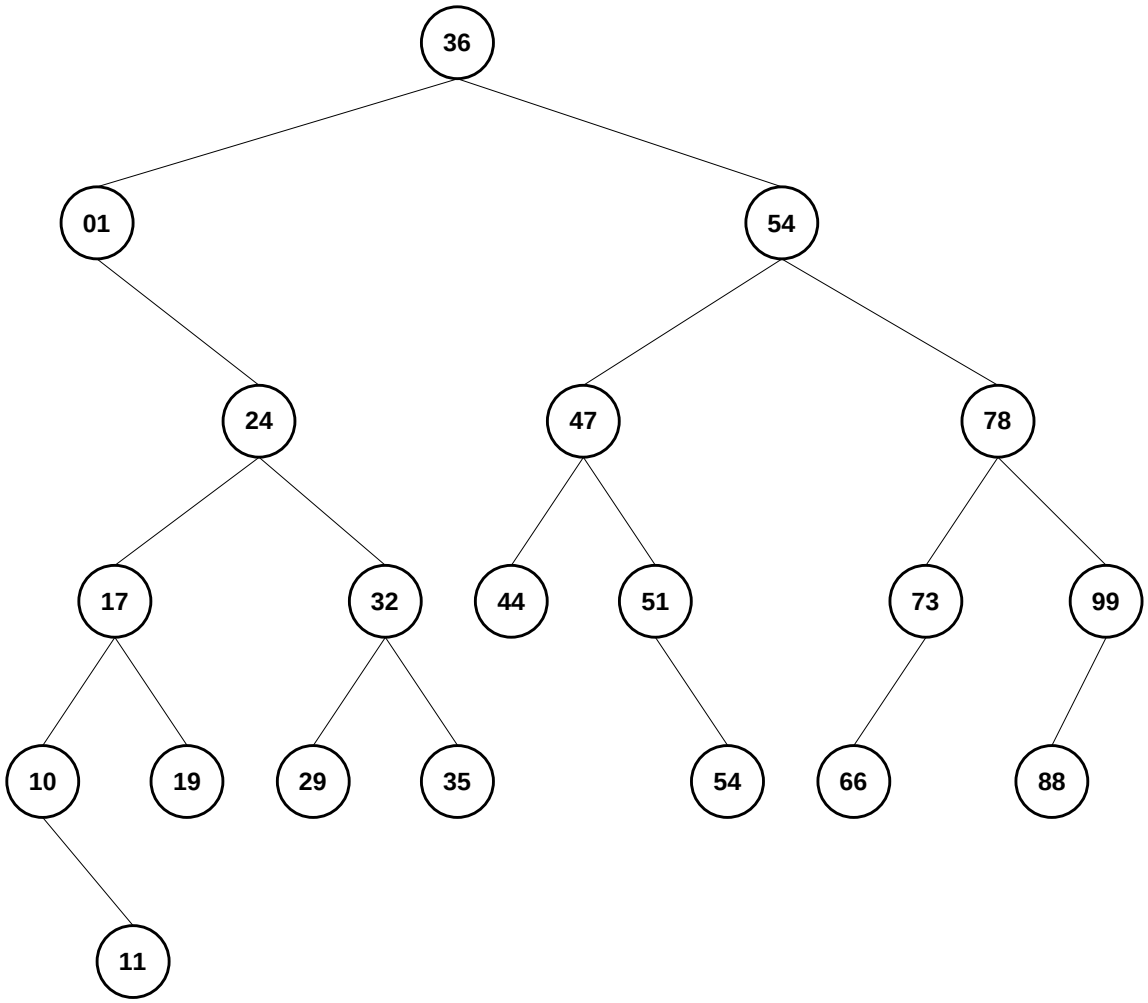
La méthode est simple : On lit un premier nombre (la racine), puis un deuxième. Si le deuxième est inférieur au premier, on le met dans la branche gauche, sinon dans la branche droite. Pour ce faire on crée un tableau qui contient pour chaque élément la branche gauche (G), la branche droite (D) et la racine (R).
La lecture du tableau se fait dans un ordre précis : on lit d'abord la branche gauche, puis la racine, puis la branche droite (parcours infixé). Le programme COBOL correspondant est peut-être un peu compliqué, mais il est efficace. Pour ceux qui ne seraient pas convaincus de son rôle pédagogique : c'est une application très intéressante pour l'option de débogage TRACE.
Sur les pages qui suivent nous donnons un tableau de nombres non triés, l'arbre binaire correspondant, son implantation sous forme de tableau, les algoritmes de création et de recherche et enfin le tableau trié.

tableau des nombres non triés

36	54	78	01	24	47	32	17	19	35	44	51	99	10	88	73	66	54	11	29

Expert

représentation graphique de l'arbre binaire

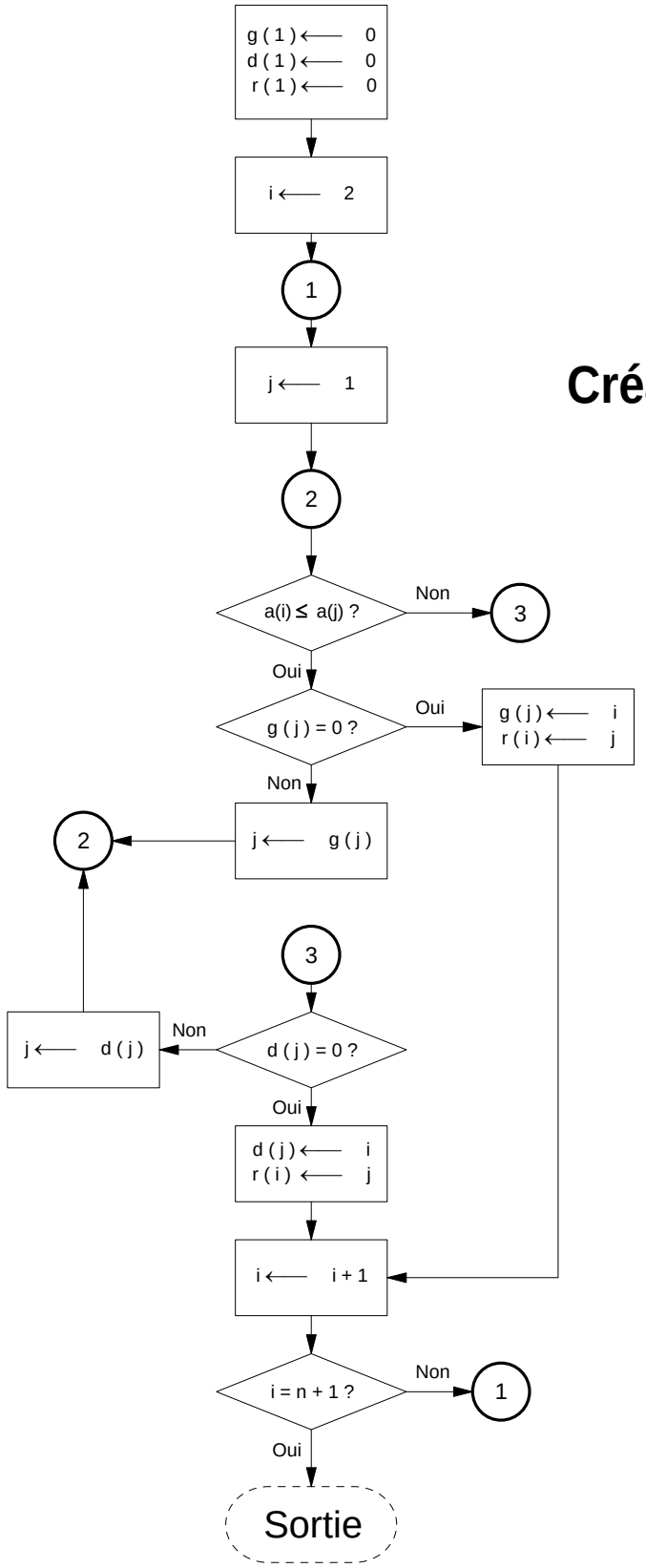


représentation sous forme de tableau de l'arbre binaire

Ai	i	G	D	R
36	1	4	2	—
54	2	6	3	1
78	3	16	13	2
01	4	—	5	1
24	5	8	7	4
47	6	11	12	2
32	7	20	10	5
17	8	14	9	5
19	9	—	—	8
35	10	—	—	7
44	11	—	—	6
51	12	—	18	6
99	13	15	—	3
10	14	—	19	8
88	15	—	—	13
73	16	17	—	3
66	17	—	—	16
54	18	—	—	12
11	19	—	—	14
29	20	—	—	7

tableau des nombres triés

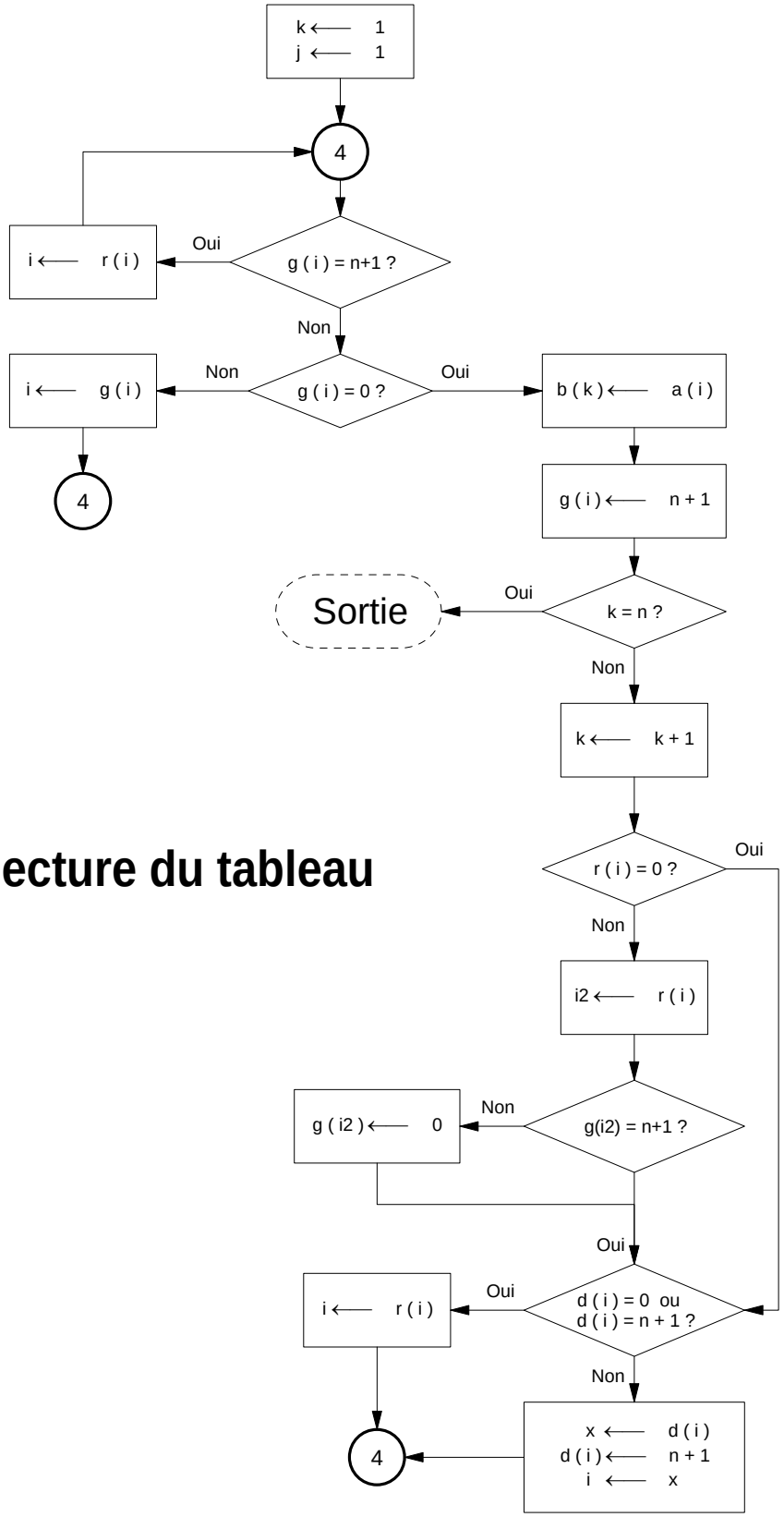
01	10	11	17	19	24	29	32	35	36	44	47	51	54	54	66	73	78	88	99



Création du tableau

Expert

Lecture du tableau



Utilisation des indices

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TRIARBIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT F-NOMBRES ASSIGN TO DISK "nombres".

DATA DIVISION.
FILE SECTION.
FD  F-NOMBRES.
01  un-nombre PIC 99.

WORKING-STORAGE SECTION.

77  N         PIC 9(3)            VALUE 998.
01  DRAPEAU   PIC 9               VALUE ZERO.
    88  FIN                       VALUE 1.

01  indices            USAGE COMP.
    02  i     PIC 9(4)            VALUE ZERO.
    02  j     PIC 9(4)            VALUE ZERO.
    02  k     PIC 9(4)            VALUE ZERO.
    02  i2    PIC 9(4)            VALUE ZERO.
    02  dy    PIC 9(4)            VALUE ZERO.
01  tableau            USAGE COMP.
    02  a     PIC 99   OCCURS 999 VALUE ZERO.
    02  g     PIC 9(4) OCCURS 999 VALUE ZERO.
    02  d     PIC 9(4) OCCURS 999 VALUE ZERO.
    02  r     PIC 9(4) OCCURS 999 VALUE ZERO.
    02  b     PIC 99   OCCURS 999 VALUE ZERO.

PROCEDURE DIVISION.
Principal.
    MOVE ZERO TO DRAPEAU.
    OPEN INPUT F-NOMBRES.
    READ F-NOMBRES AT END MOVE 1 TO DRAPEAU.
    PERFORM Initialise VARYING i FROM 1 BY 1
        UNTIL FIN OR i > N.
    CLOSE F-NOMBRES.
    PERFORM Make-Tab THROUGH Fin-Tri.
    STOP RUN.

Initialise.
    MOVE un-nombre TO a(i).
    READ F-NOMBRES AT END MOVE 1 TO DRAPEAU.

Make-Tab.
    SUBTRACT 1 FROM i GIVING N.
    MOVE 2 TO i.

Label-1.
    MOVE 1 TO j.

Label-2.
    IF a(i) > a(j) GO TO Label-3.
    IF g(j) = ZERO GO TO Label-6.
    MOVE g(j) TO j.
    GO TO Label-2.
```

Expert

```
Label-3.
    IF d(j) = ZERO GO TO Label-4.
    MOVE d(j) TO j.
    GO TO Label-2.

Label-4.
    MOVE i TO d(j).
    MOVE j TO r(i).
    GO TO Label-5.

Label-6.
    MOVE i TO g(j).
    MOVE j TO r(i).

Label-5.
    ADD 1 TO i.
    IF i NOT = N + 1 GO TO Label-1.

Make-Tri.
    MOVE 1 TO i k.

Aff-1.
    IF g(i) NOT = N + 1 GO TO Aff-2.
    MOVE r(i) TO i.
    GO TO Aff-1.

Aff-2.
    IF g(i) = ZERO GO TO Aff-3.
    MOVE g(i) TO i.
    GO TO Aff-1.

Aff-3.
    MOVE a(i) TO b(k).
    MOVE N TO g(i).
    ADD 1 TO g(i).
    IF k = N GO TO Aff-5.
    ADD 1 TO k.
    MOVE r(i) TO i2.
    IF i2 NOT = ZERO
        IF g(i2) NOT = N + 1 MOVE ZERO TO g(i2).
    IF d(i) = ZERO OR N + 1 GO TO Aff-4.
    MOVE d(i) TO dy.
    MOVE N TO d(i).
    ADD 1 TO d(i).
    MOVE dy TO i.
    GO TO Aff-1.

Aff-4.
    MOVE r(i) TO i.
    GO TO Aff-1.

Aff-5.
    MOVE 1 TO i.

Aff-6.
    DISPLAY b(i) " " WITH NO ADVANCING.
    ADD 1 TO i.
    IF i NOT = N + 1 GO TO Aff-6.
Fin-Tri.
    EXIT.
```

Utilisation des index

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TRIDXBIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT F-NOMBRES ASSIGN TO DISK "nombres".

DATA DIVISION.
FILE SECTION.

FD  F-NOMBRES.
01  un-nombre PIC 99.

WORKING-STORAGE SECTION.

77  N         PIC 9(4) VALUE 998.
77  i-test    PIC 9(4) VALUE ZERO.

01  DRAPEAU   PIC 9    VALUE ZERO.
    88  FIN            VALUE 1.

01  tableau            VALUE ZERO.
    02  colonne        OCCURS 5.
        03  c PIC 9(4) OCCURS 999 INDEXED BY i j k i2 dy.

PROCEDURE DIVISION.
Principal.
    MOVE ZERO TO DRAPEAU.
    OPEN INPUT F-NOMBRES.
    READ F-NOMBRES AT END MOVE 1 TO DRAPEAU.
    PERFORM Initialise VARYING i FROM 1 BY 1
        UNTIL FIN OR i > N.
    CLOSE F-NOMBRES.
    PERFORM Make-Tab THROUGH Fin-Tri.
    STOP RUN.

Initialise.
    MOVE un-nombre TO c(1 i).
    READ F-NOMBRES AT END MOVE 1 TO DRAPEAU.

Make-Tab.
    SET N TO i.
    SUBTRACT 1 FROM N.
    SET i TO 2.

Label-1.
    SET j TO 1.

Label-2.
    IF c(1 i) > c(1 j) GO TO Label-3.
    IF c(2 j) = ZERO GO TO Label-6.
    SET j TO c(2 j).
    GO TO Label-2.

Label-3.
    IF c(3 j) = ZERO GO TO Label-4.
    SET j TO c(3 j).
    GO TO Label-2.
```

```
Label-4.
    SET c(3 j) TO i.
    SET c(4 i) TO j.
    GO TO Label-5.

Label-6.
    SET c(2 j) TO i.
    SET c(4 i) TO j.

Label-5.
    SET i UP BY 1.
    SET i-test TO i.
    IF i-test NOT = N + 1 GO TO Label-1.

Make-Tri.
    SET i k TO 1.

Aff-1.
    IF c(2 i) NOT = N + 1 GO TO Aff-2.
    SET i TO c(4 i).
    GO TO Aff-1.

Aff-2.
    IF c(2 i) = ZERO GO TO Aff-3.
    SET i TO c(2 i).
    GO TO Aff-1.

Aff-3.
    MOVE c(1 i) TO c(5 k).
    MOVE N TO c(2 i).
    ADD 1 TO c(2 i).
    IF k = N GO TO Aff-5.
    SET k UP BY 1.
    IF c(4 i) NOT = ZERO
        SET i2 TO c(4 i)
        IF c(2 i2) NOT = N + 1 MOVE ZERO TO c(2 i2).
    IF c(3 i) = ZERO OR N + 1 GO TO Aff-4.
    SET dy TO c(3 i).
    MOVE N TO c(3 i).
    ADD 1 TO c(3 i).
    SET i TO dy.
    GO TO Aff-1.

Aff-4.
    SET i TO c(4 i).
    GO TO Aff-1.

Aff-5.
    SET i TO 1.

Aff-6.
    DISPLAY c(5 i) " " WITH NO ADVANCING.
    SET i UP BY 1.
    SET i-test TO i.
    IF i-test NOT = N + 1 GO TO Aff-6.

Fin-Tri.
    EXIT.
```

Annexes

Mots réservés

A

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABET
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALSO
ALTER
AND
ANY
APPLY
ARE
AREA
AREA-VALUE
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
AUTO
AUTO-SKIP
AUTOMATIC

B

BACKGROUND-COLOR
BACKGROUND-COLOUR
BACKWARD
BASIS
BEEP
BEFORE
BEGINNING
BELL
BINARY
BLANK
BLINK
BLOCK
BOTTOM
BY

C

C01
C02
C03
C04
C05
C06
C07
C08
C09
C10
C11
C12
CALL
CANCEL
CBL
CD
CF
CH
CHAIN
CHAINING
CHANGED
CHARACTER
CHARACTERS
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COL
COLLATING
COLOR
COLUMN
COM-REG
COMMA
COMMAND-LINE
COMMIT
COMMON
COMMUNICATION
COMP
COMP-0
COMP-1
COMP-2
COMP-3
COMP-4
COMP-5
COMP-X
COMPUTATIONAL
COMPUTATIONAL-0
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTATIONAL-5
COMPUTATIONAL-X
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONVERTING
COPY
CORE-INDEX
CORR
CORRESPONDING
COUNT
CRT
CRT-UNDER
CSP
CURRENCY
CURRENT-DATE
CURSOR

D

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DE
DEBUG
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISABLE
DISK
DISP
DISPLAY
DISPLAY-1
DISPLAY-ST
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

E

EGCS
EGI
EJECT
ELSE
EMI
EMPTY-CHECK
ENABLE
END
END-ACCEPT
END-ADD
END-CALL
END-COMPUTE
END-DELETE
END-DIVIDE
END-EVALUATE
END-IF
END-MULTIPLY
END-OF-PAGE
END-PERFORM
END-READ
END-RECEIVE
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END-SUBTRACT
END-UNSTRING
END-WRITE
ENDING
ENTER
ENTRY
ENVIRONMENT
EOP
EQUAL
ERASE

ERROR
ESCAPE
ESI
EVALUATE
EVERY
EXAMINE
EXCEPTION
EXCESS-3
EXCLUSIVE
EXEC
EXECUTE
EXHIBIT
EXIT
EXTEND
EXTERNAL

F

FALSE
FD
FILE
FILE-CONTROL
FILE-ID
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FIXED
FOOTING
FOR
FOREGROUND-COLOR
FOREGROUND-COLOUR
FROM
FULL

G

GENERATE
GIVING
GLOBAL
GO
GOBACK
GREATER
GRID
GROUP

H

HEADING
HIGH-VALUE
HIGH-VALUES
HIGHLIGHT

I

I-O
I-O-CONTROL
ID
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSERT
INSPECT
INSTALLATION
INTO
INVALID
IS

J

JAPANESE
JUST
JUSTIFIED

K

KEPT
KEY
KEYBOARD

L

LABEL
LAST
LEADING
LEAVE
LEFT
LEFT-JUSTIFY
LEFTLINE
LENGTH
LENGTH-CHECK
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

M

MANUAL
MEMORY
MERGE
MESSAGE
MODE
MODULES
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY

N

NAME
NAMED
NATIVE
NEGATIVE
NEXT
NO
NO-ECHO
NOMINAL
NOT
NOTE
NULL
NULLS
NUMBER
NUMERIC
NUMERIC-EDITED

O

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OTHERWISE
OUTPUT
OVERFLOW
OVERLINE

P

PACKED-DECIMAL
PADDING
PAGE
PAGE-COUNTER
PALETTE
PASSWORD
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIONING
POSITIVE
PREVIOUS
PRINT
PRINT-SWITCH
PRINTER
PRINTER-1
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROCESSING
PROGRAM
PROGRAM-ID
PROMPT
PROTECTED
PURGE

Q

QUEUE
QUOTE
QUOTES

R

RANDOM
RANGE
RD
READ
READY
RECEIVE
RECORD
RECORD-OVERFLOW
RECORDING
RECORDS
REDEFINES
REEL
REFERENCE
REFERENCES
RELATIVE
RELEASE
RELOAD
REMAINDER
REMARKS
REMOVAL
RENAMES
REORG-CRITERIA
REPLACE
REPLACING
REPORT
REPORTING
REPORTS
REQUIRED
REREAD
RERUN
RESERVE
RESET
RETURN
RETURN-CODE
REVERSE-VIDEO
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
RIGHT-JUSTIFY
ROLLBACK
ROUNDED
RUN

S

S01
S02
SAME
SCREEN
SD
SEARCH
SECTION
SECURE
SECURITY
SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
SELECTIVE
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SERVICE
SET
SIGN
SIZE
SKIP-1
SKIP-2
SKIP-3
SORT
SORT-CONTROL
SORT-CORE-SIZE
SORT-FILE-SIZE
SORT-MERGE
SORT-MESSAGE
SORT-MODE-SIZE
SORT-RETURN
SOURCE
SOURCE-COMPUTER
SPACE
SPACE-FILL
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
START
STATUS
STOP
STORE
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBPROGRAM
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONISED
SYSIN
SYSIPT
SYSLIST
SYSLST
SYSOUT
SYSPNCH
SYSPUNCH

T

TABLE
TALLY
TALLYING
TAPE
TERMINAL
TERMINATE
TEST
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIME-OF-DAY
TIMES
TITLE
TO
TOP
TOTALED
TOTALING
TRACE
TRACK-AREA
TRACK-LIMIT
TRACKS
TRAILING
TRAILING-SIGN
TRANSFORM
TRUE
TYPE

U

UNDERLINE
UNIT
UNLOCK
UNSTRING
UNTIL
UP
UPDATE
UPON
USAGE
USE
USER
USING

V

VALUE
VALUES
VARIABLE
VARYING

W

WHEN
WHEN-COMPILED
WITH
WORDS
WORKING-STORAGE
WRITE
WRITE-ONLY

Z

ZERO
ZERO-FILL
ZEROES
ZEROS

Tableau récapitulatif des formats

Caractère	Elément origine		Elément d'édition	
	PICTURE	CONTENU	PICTURE	CONTENU
Z	9(5)	12345	ZZZ99	12345
	9(5)	00123	ZZZ99	□□123
	9(5)	00100	ZZZ99	□□100
	9(5)	00000	ZZZ99	□□□00
	9(3)	100	Z(5)	□□100
	9(3)	000	Z(5)	□□□□□
.	99999	12345	ZZZ.99	345.00
	99V999	12345	ZZZ.99	□12.34
	99V999	12345	ZZZZZZ	□□□□12
	99V999	00123	ZZZ.99	□□□.12
	99V999	00123	Z(6)	□□□□□□
	99V999	00012	ZZ.ZZZ	□□.012
	99V999	00001	Z.ZZZZ	□.0010
	99V999	00000	Z.ZZZZ	□□□□□□
,	9(5)	12345	99,999	12,345
	9(5)	00012	ZZ,999	□□,012
	9(5)	00001	ZZ,ZZZ	□□□□□1
	9(5)	00000	ZZ,ZZZ	□□□□□□
	99V999	12345	99,999	00,012
	99V999	12345	ZZ,999	□□,012
	99V999	12345	ZZ,ZZZ	□□□□12
0 B	9(5)	12345	BB999.00	□□345.00
	9(5)	12345	00099.00	00045.00
	999V99	12345	00099.00	00023.00
	9(2)	02	00099.00	00002.00
	9(2)	02	000ZZ.00	000□2.00
	999	123	9B9B9	1□2□3
	9V99	123	909B9	102□3
	999V99	01234	ZZ0.ZZ	120.34
	9(5)	12345	ZZBZ0Z0Z	12□30405
	9(5)	12030	ZZBZ0Z0Z	12□00300
	9(5)	00001	ZZBZ0Z0Z	□□□□□01
*	9(5)	12345	***99	12345
	9(5)	00123	***99	**123
	9(5)	00000	***99	***00
	9(5)	00000	*(5)	*****
	99V999	00012	**.***	**.012
	9(5)	02345	**,***	*2,345
	9(5)	00012	$**,***	$****12
	99V999	00000	$**.***	$**.***
	9(5)	12345	$**,**9.99	$12,345.00
	9(5)	00123	$**,**9.99	$***123.00
$ fixe	9(5)	12345	$9(5)	$12345
	9(5)	12345	$9(4)	$2345
	9(5)	00012	$Z(5)	$□□□12
	9(4)V99	123456	$ZZ,ZZZ.ZZ	$□1,234.56
	9(5)	00000	$ZZ,ZZ9.99	$□□□□□0.00
	9(5)	00000	$ZZ,ZZZ.99	$□□□□□□.00
	9(5)	00000	$ZZ,ZZZ.ZZ	$□□□□□□□□□□

+ - fixes	S99	+12	+99	+12
	S99	-12	+99	-12
	S99	+12	99+	12+
	9(5)	12345	+Z(5)	+12345
	9(5)	00123	+Z(5)	+□□123
	S9(5)	+00123	+Z(5)	+□□123
	S9(5)	-00123	+Z(5)	-□□123
	S9(5)	+00123	Z(5)+	□□123+
	S9(5)	-00123	Z(5)+	□□123-
	S9(5)	-00123	$+Z(5)	$-□□123
	S99	-12	-99	-12
	S99	+12	-99	□12
	99	12	-Z9	□12
	S99	-12	99-	12-
	S99	+12	99-	12□
	S9(3)	+012	-ZZ9	□□12
	S9(3)	-012	-Z(3)	-□12
	S99	-00	-ZZ	□□□
	S99	+00	-ZZ	□□□
	S99	-00	ZZ-	□□-
	S9(5)	-00123	$Z(5)-	$□□123-
	S99V9	-123	$Z(3).ZZ-	$□12.30-
CR DB	S99	-12	99CR	12CR
	S99	+12	99CR	12□□
	S9(5)	-12345	$9(5)CR	$12345CR
	S99	-12	99DB	12DB
	S99	+12	99DB	12□□
$ flottant	9(5)	12345	$$$99	$2345
	9(4)	0123	$$99	$123
	9(4)	1234	$$99	$234
	9(4)	0001	$$99	□$01
	9(4)	0001	$$$$	□□$1
	99V99	0001	$$.99	□$.01
	99V99	-0001	+$$.$$	-□$.01
	99V99	0000	$$.$$	□□□□
	S9(4)	+2345	$$$$$.99CR	$2345.00□□
+ - flottants	S9(5)	+12345	+++99	+2345
	S9(4)	+1234	+++99	+1234
	S9(4)	-1234	+++99	-1234
	S9(4)	-0123	+(3)99	□-123
	S9(4)	-0001	+(3)99	□□-01
	S9(4)	-0001	+(5)	□□□-1
	S9(4)	+0000	+++++	□□□□
	9(4)V99	000001	++,+++.++	□□□□+.01
	9(4)V9	00001	++++.+	□□□+.1
	S9(5)	-12345	-(3)99	-2345
	S9(5)	+12345	---99	□2345
	S9(4)	-1234	---99	-1234
	S9(4)	+1234	---99	□1234
	S9(2)	-12	---99	□□-12
	S9(2)	+12	$---99	$□□□12
	9(4)	0001	-----	□□□1
	S9(4)	-0001	-(5)	□□□-1
	9(4)	0000	-----	□□□□

Codes d'erreurs des entrées-sorties[13]

Domaine	Signification
00	exécution avec succès
01-09	exécution avec succès, mais un warning
10-19	fin de fichier
20-29	erreur de clé d'enregistrement
30-39	erreur permanente
40-49	erreur logique
90-99	erreur système

Valeur	Signification
00	opération réussie
04	lecture réussie, longueur d'enregistrement incohérente
10	fin de fichier
21	la clé n'est pas en séquence
22	la clé existe déjà pour un autre enregistrement
23	enregistrement non trouvé
24	dépassement de limites dans la zone d'écriture du fichier
30	erreur permanente d'entrées-sorties
34	écriture hors-limites en séquentiel
35	essai d'ouverture d'un fichier non existant
37	essai d'ouverture d'un fichier dont on n'a pas les droits
38	essai d'ouverture d'un fichier verrouillé
41	essai d'ouverture d'un fichier déjà ouvert
42	essai de fermeture d'un fichier non ouvert
43	`DELETE` / `REWRITE` sans lecture préalable (accès séquentiel)
44	erreur de taille de l'enregistrement en réécriture
46	essai de lecture d'un article inexistant
47	essai de lecture dans un fichier non ouvert en `INPUT` ou `I-O`
48	essai d'écriture dans un fichier non ouvert en `OUTPUT` ou `I-O`
49	`DELETE` / `REWRITE` dans un fichier non ouvert

[13] Compilateur Micro Focus

Index

A

ACCEPT, 19
ADD, 25
AFTER, 36
AT END, 64

B

BINARY, 18
BLANK WHEN ZERO, 16
Branchements, *voir Sauts*

C

Calculs, 24
CALL, 42
Caractères
  autorisés, 1
  chaînes, 27-32; 76
CLOSE, 62
COMPUTATIONAL, 18
COMPUTE, 27
Concaténation de chaînes, 29
Constantes
  alphanumériques, 3
  figuratives, 1
  numériques, 2
Conventions
  de codage, 3
  typographiques, v
COPY, 22

D

DATA DIVISION, 7
Date
  système, 19; 21
  vérification, 78
Débogage, 45
DECLARATIVES, 45
DELETE, 68
DISPLAY, 19
DIVIDE, 26
Données
  affectation, 11; 20
  classes, 10
  description, 8

E

ENVIRONMENT DIVISION, 7
Erreurs
  codes, 96
  traitement, 45
EXAMINE, 28
EXIT, 37
EXIT PROGRAM, 42

F

FD, 60
Fichiers
  accès, 55; 64; 69
  fermeture, 62
  modifications, 57; 66-68
  organisation, 55
  ouverture, 62; 69
  relatifs
    méthode du hashing, 56
    représentation sous UNIX, 57
  réorganisation, 59
FILLER, 15
Formats
  de données, 9
  internes, 18
  tableau récapitulatif, 94

G

GO TO, 23
GOBACK, *voir EXIT PROGRAM*

I

IDENTIFICATION DIVISION, 6
IF, 32
Illegal character in numeric field, 46
Index, 50; 88
Indices, 48; 85
INITIALIZE, 22
INVALID-KEY, 64

J

JUSTIFIED-RIGHT, 17

L

Ligne (poursuite de), 3
LINKAGE SECTION, 42
Littéraux
  alphanumériques, 3
  numériques, 2

M

Modules
  appel, 42
  sortie, 42
Modulo, 78
Mots
  réservés, 1
  réservés (liste des), 91-93
  utilisateur, 2; 9
MOVE, 20
MOVE CORRESPONDING, 21
MULTIPLY, 26

N

Nombres aléatoires, 76
Noms-condition, 13; 33

O

OCCURS, 47
OPEN, 62
Opérateurs
  arithmétiques, 27
  de comparaison, 32
Options de compilation, 45; 46

P

PACKED-DECIMAL, 18
PERFORM, 34-41
PICTURE, 8
PROCEDURE DIVISION, 8
Procédures
  appel, 34-41
  sortie, 37
Programmes
  calcul de puissances, 35
  factorielle, 36
  génération de pages-écran, 74
  menu, 72
  poèmes, 76
  recherche dichotomique, 35
  vérification de dates, 78

R

READ, 64
Recherche
  dans un fichier, 69
  dans un tableau, 52
REDEFINES, 13
RENAMES, 12
REWRITE, 66
ROUNDED, 24

S

Sauts
  conditionnels, 23
  inconditionnels, 23
SEARCH, 52
SELECT, 59
SET, 51
SIGN, 17
START, 69
STOP RUN, 23
STRING, 29; 76
Structure
  d'un programme, 6
  de fichiers, 60
SUBTRACT, 25

T

Tableaux, 47-53
TRACE, 46
TRANSFORM, 31
Tri
  arbre binaire, 82
  bubble sort, 81
  par fichier indexé, 79

U

UNSTRING, 30
USAGE, 18
USE, 45

V

VALUE, 16
VARYING, 36

W

WITH DEBUGGING MODE, 45
WORKING-STORAGE SECTION, 8
WRITE, 62