



Daniel Tischer - Laurent Crivello 1994

Nous tenons à remercier :

Michaël Clauss

Gabrielle Limousin

Francine Meyer

Joseph Roller

Georges Rosenthal

Malika Saulnier

Julien Schnell

Introduction

Cet ouvrage est un manuel pour débutant confirmé. Il s'adresse à toutes les personnes connaissant déjà les rudiments de l'algorithmique et qui désirent apprendre un langage de programmation simple et efficace.

Le langage de ce manuel est à la norme ANS 1985 (American National Standard). Toutefois, certains domaines ne seront pas approfondis car trop complexes et donc sans intérêt pour le débutant. En particulier les modules d'édition et de communication, même s'ils sont très passionnants, ont été écartés.

L'originalité de cet ouvrage réside dans ses exemples commentés qui permettent à tout le monde d'accéder aux fonctions les plus compliquées.

Nous vous souhaitons donc bon courage !

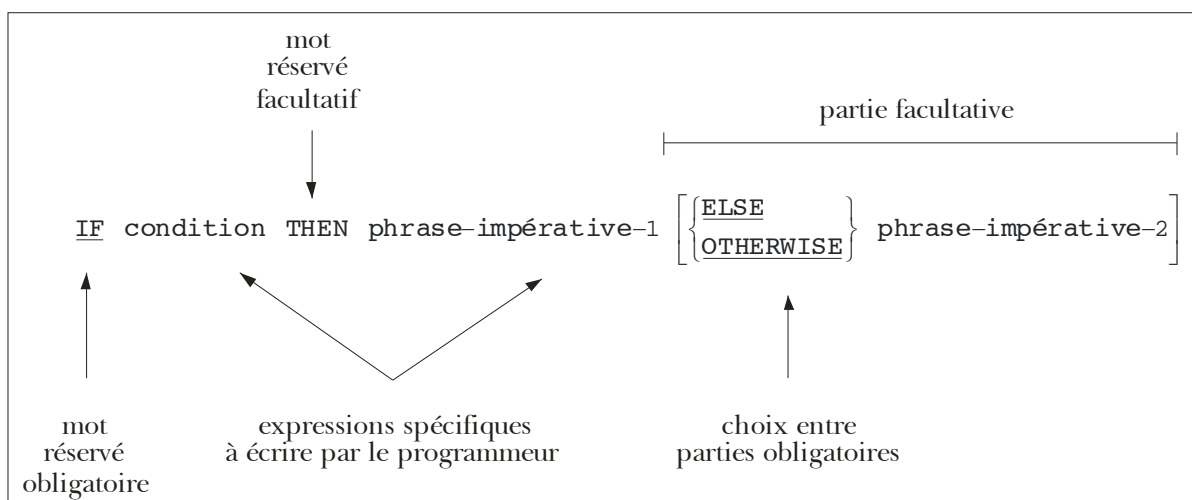
Organisation

Ce manuel suit une approche progressive. On distingue six grands chapitres :

- | | |
|-------------------------------------|----------------------------------|
| 1. les connaissances de base | 4. les fichiers |
| 2. la programmation en Cobol | 5. la gestion des erreurs |
| 3. les tableaux | 6. les annexes |

Chaque chapitre se décompose en plusieurs sous-parties, pour lesquelles nous avons adopté une organisation thématique, et au sein de chaque sous-partie les fonctions sont classées par ordre alphabétique. Tout le manuel a été conçu pour faciliter la recherche ; il mérite que l'on s'en serve, à toute occasion, comme manuel de référence.

Conventions typographiques



Petit historique

Dans l'histoire de l'informatique, 30 années représentent les cycles de vie de plusieurs générations. Un ancêtre du temps des premiers grands progrès dans le domaine de l'informatique est le Cobol. Toutefois, ce langage n'a rien perdu de son charme – au contraire. Son histoire commença, comme tant d'autres, en pleine guerre froide : en mai 1959, le Pentagone demandait à Washington l'élaboration d'un langage universel et facile d'utilisation destiné à gérer de grandes quantités d'informations. La CODASYL (*Conference on Data System Languages*) fut fondée, elle définit en septembre 1959 un nouveau langage, qui fut présenté au public sous le nom de COBOL. Cet acronyme signifie **CO**mmun **B**usiness **O**riented **L**anguage. A partir de ce moment, le Cobol a connu une grande diffusion et dans certains domaines un véritable boom. Aujourd'hui encore, la plupart des applications dans les administrations, les banques, les assurances et les grandes entreprises sont écrites en Cobol. Mais un langage qui a été développé il y a 30 ans peut-il encore faire face aux exigences actuelles ? Le Cobol n'est-il pas une langue morte uniquement maintenue en vie par des intérêts économiques, un vestige d'une autre ère informatique ? Mais contrairement à ce que pensent certains, le Cobol est toujours présent et connaît même une nouvelle jeunesse ! En particulier le PC contribue à cette renaissance en ouvrant de nouvelles perspectives de programmation.

Les standards

Depuis le début on a essayé de standardiser le Cobol et pour aucun autre langage un tel niveau de standardisation n'a été atteint. Ceci apporte une grande indépendance vis-à-vis du matériel et permet de porter facilement les applications sur d'autres systèmes. Cette propriété du Cobol abaisse les coûts de réalisation car le produit développé peut être vendu à des clients ayant des configurations matérielles différentes.

Environ tous les 5 ans, un nouveau standard est publié. Ce petit lifting rafraîchit les cellules du Cobol qui reste ainsi maître de l'évolution. Malgré le développement intempestif du langage, on fait très attention à garantir la plus grande compatibilité possible. Ainsi on pourra toujours utiliser d'anciens programmes sur de nouveaux compilateurs, ce qui réduit les coûts de maintenance.

A quoi sert le COBOL ?

Le Cobol n'est pas un langage universel. Il n'est adapté ni à la commande des robots dans l'industrie, ni à la programmation système. Mais dans le domaine de la gestion il existe peu de langages qui répondent aussi bien aux problèmes économiques. Le Cobol se caractérise par sa facilité de compréhension. Celle-ci est due en partie à l'utilisation de mots facultatifs (de remplissage) et à des instructions très explicites. La ressemblance avec le langage parlé augmente la transparence des programmes et facilite les modifications. Les possibilités de calcul ont été étendues et adaptées à la problématique des différents domaines économiques. Toutefois, la puissance de calcul est insuffisante pour des applications scientifiques.

Les fichiers

L'atout principal du Cobol est la gestion des fichiers. Au premier abord, les conventions de codage peuvent paraître très compliquées à utiliser. Mais ce qui apparaît d'abord comme une faiblesse se transforme avec un peu de pratique en avantage certain. Le programmeur a ici la possibilité de choisir l'organisation et le mode d'accès des fichiers. Fichiers séquentiels, fichiers relatifs et fichiers indexés avec clés multiples, le programmeur n'a que l'embarras du choix. Le Cobol contient aussi des options pour les environnements multi-utilisateurs. Une fois les fichiers définis, leur utilisation est d'une simplicité enfantine.

La description de données

La partie la plus fastidieuse dans la réalisation d'un programme Cobol est l'écriture de l'en-tête qui contient la définition de toutes les variables. Sa taille est parfois plus importante que la partie instructions elle-même. En particulier, la définition des données diffère des autres langages tels que le Basic, le C ou le Pascal. Chaque champ de données doit être défini à part. Il faut en préciser le type, la longueur et la position exacte du point décimal dans le champ. Mais cela présente l'avantage de pouvoir faire correspondre exactement la description de données aux besoins des utilisateurs.

Une propriété très agréable du Cobol est qu'il refuse la saisie de valeurs ne correspondant pas au format défini. Il n'est donc pas nécessaire d'écrire soi-même des routines de limitation du champ d'entrée. L'affichage des données a aussi quelques particularités intéressantes. Ainsi on peut définir des formats édités permettant par exemple de supprimer les zéros en tête ou d'afficher des étoiles devant les nombres. Chaque définition de données est précédée d'un numéro de niveau. Ceci permet de décrire minutieusement n'importe quelle structure de données.

A priori, l'utilisateur ne s'occupe pas de la représentation interne des données, mais il peut en préciser le format comme par exemple la représentation des nombres en binaire.

Performances

Le Cobol satisfait naturellement les exigences de la programmation structurée. La modularité est réalisée par l'appel de sous-programmes internes et externes. On évite ainsi le "code-spaghetti" (typique du Basic). Pour les sous-programmes externes, la transmission de paramètres est possible. Comme le Cobol ne résout pas tous les problèmes, on a la possibilité de lier des modules écrits dans d'autres langages (par exemple en C ou en assembleur).

Actuellement, les compilateurs Cobol proposent une optimisation du code. Ainsi les modules exécutables deviennent plus petits et plus performants. La programmation de commandes SQL est possible tout comme la réalisation d'applications sous l'interface graphique d'OS/2. Dans certains cas on peut même simuler un environnement mini du type CICS, IMS/VS d'IBM et IMS-COBOL. L'élaboration de programmes pour les minis peut donc être transplantée sur un PC.

Table des matières

Chapitre I - Connaissances de base	1
Eléments du langage	1
Conventions de codage	3
Structure du COBOL	6
Description de données en WORKING-STORAGE SECTION	12
Elargissement de la description de données	19
 Chapitre II - Programmation en COBOL	25
Aide à l'écriture	27
Fonctions d'entrées-sorties	28
Mouvements de données	30
Initialisation de données	36
Opérations arithmétiques	37
La condition	42
Instructions de saut	46
Appel de procédures	48
Appel de modules externes	52
Programmes contenus	56
 Chapitre III - Les tableaux	57
Introduction	59
Définition de tableaux	59
Accès aux éléments d'un tableau	61
Différences entre indices et index	65
Exemple : Tri bulle (bubble sort)	66
 Chapitre IV - Les fichiers	67
Généralités	69
Description de fichiers	74
Instructions pour la gestion des fichiers	78
Tri-fusion	89
 Chapitre V - Traitement des erreurs	97
Aide à la mise au point	99
 Annexes	103
Mots réservés	105
Tableau récapitulatif des formats	109
Codes d'erreurs des entrées-sorties	111
 Index alphabétique	I
Bibliographie	V

Connaissances de base

Partie I - Éléments du langage	1
Jeu de caractères utilisable	1
Mots réservés	1
Noms de variables et de procédures	2
 Partie II - Conventions de codage	3
Règles.	3
Suggestions de présentation	4
 Partie III - Structure du COBOL	6
Structure générale	6
IDENTIFICATION DIVISION	6
ENVIRONMENT DIVISION	7
DATA DIVISION	10
PROCEDURE DIVISION	12
 Partie IV - Description de données	12
Les différentes classes de données.	15
Qualification avec OF ou IN	16
REDEFINES	17
RENAMES	17
Noms-de-condition	18
 Partie V - Elargissement de la description de données	19
BLANK WHEN ZERO	19
FILLER	19
JUSTIFIED RIGHT	20
SIGN	21
SYNCHRONIZED	22
USAGE	23
VALUE	24

Connaissances de base



Éléments 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 soulignés dans la description du langage.

mots facultatifs (optional words)

Ils permettent de rendre plus lisibles certaines instructions, par exemple IS et THEN¹.

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 <i>Littéral</i>	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

¹ THEN peut être utilisé à partir de la norme ANSI 1985.

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 ne 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 : $\begin{bmatrix} + \\ - \end{bmatrix}$ mantisse E $\begin{bmatrix} + \\ - \end{bmatrix}$ Exposant
- La mantisse contient au plus 16 chiffres et le point décimal ; elle peut être précédée d'un signe.
- L'exposant comporte un ou deux chiffres et peut être précédé d'un signe.

Exemple 1-1

littéraux autorisés	littéraux interdits
123456789.12345678	1234567890123456789
6789	6 789
3.1415	5.
+0.7	0.7+
-6354.0	5-
0	1,234.56
001	1,2
+1234567890.123456E33	1234567890.1234567E9
1.3E01	1.3-E01
1.3E1	1.3E001
0.13E02	E11
3.3E00	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

"THAT'S ALL, FOLKS !"

"12345"

QUOTE

SPACE

Tous les caractères sont autorisés, même ! et '

Les calculs sont interdits avec cette constante !

QUOTE et SPACE sont des constantes figuratives, une forme 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). * ou / ligne de commentaires, n'est pas évaluée. / a pour effet complémentaire de provoquer un saut de page dans le fichier liste lors de la compilation. 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
	1	2	3	4	5	6	7	8
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 débiter en colonne 12. Ceci permet alors d'aligner le texte par rapport à la chaîne du dessus.

Suggestions de présentation

Indentation

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

Procédure Division

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

Comment rédiger

- Les commentaires alourdissent le programme et ne doivent pas expliquer ce que font des instructions déjà claires par elles-mêmes.
- 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.
- Ecrivez en toutes lettres les opérateurs relationnels, même si aujourd'hui les caractères < et > sont disponibles sur presque tous les terminaux.

Modèles d'écriture²

```

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

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

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

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

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

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

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.

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.

```

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

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 <i>phrase</i> COBOL.
PARAGRAPH	Regroupe plusieurs phrases COBOL reliées entre elles par un lien logique. Chaque paragraphe est introduit par un titre commençant en colonne 8, il s'arrête alors au prochain titre (sinon à la fin du programme).
SECTION	La section est constituée d'unités 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 prochaine section (ou en fin de programme).
DIVISION	Tout programme COBOL comporte quatre divisions ayant chacune un rôle spécifique : <ul style="list-style-type: none">■ IDENTIFICATION DIVISION Identification du programme■ ENVIRONMENT DIVISION Description des périphériques■ DATA DIVISION Description des fichiers / des données■ PROCEDURE DIVISION Partie des traitements

Identification du programme (IDENTIFICATION DIVISION)

IDENTIFICATION DIVISION.

PROGRAM - ID. nom-de-programme.

[AUTHOR. [rubrique-commentaire.]...]
[INSTALLATION. [rubrique-commentaire.]...]
[DATE - WRITTEN. [rubrique-commentaire.]...]
[DATE - COMPILED. [rubrique-commentaire.]...]

- L'en-tête et le nom du programme après PROGRAM-ID sont obligatoires.
- 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.
- Le champ DATE-COMPILED est mise à jour dans le fichier liste au moment de la compilation³.

³ Le standard A.N.S. 87 a placé toutes les clauses optionnelles AUTHOR, DATE-WRITTEN, DATE-COMPILED dans la catégorie *obsolète* – à supprimer dans la prochaine norme.

Description des périphériques (ENVIRONMENT DIVISION)

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE - COMPUTER. nom-de-calculateur [WITH DEBUGGING MODE].

OBJECT - COMPUTER. nom-de-calculateur

[MEMORY SIZE entier {WORDS
CHARACTERS
MODULES}]

[PROGRAM COLLATING SEQUENCE IS nom-d' alphabet]

[SEGMENT - LIMIT IS numéro-de-segment].

SPECIAL - NAMES.

[nom-de-réalisateur

{IS nom-mnémonique [ON STATUS IS nom-de-condition-1
[OFF STATUS IS nom-de-condition-2]]

{IS nom-mnémonique [OFF STATUS IS nom-de-condition-2
[ON STATUS IS nom-de-condition-1]]} ...

[ON STATUS IS nom-de-condition-1 [OFF STATUS IS nom-de-condition-2]

[OFF STATUS IS nom-de-condition-2 [ON STATUS IS nom-de-condition-1]]]

[STANDARD - 1

NATIVE

nom-de-réalisateur

[THROUGH] littéral-2

[THRU]

littéral-1 ALSO littéral-3

[ALSO littéral-4]...

[THROUGH] littéral-6

[THRU]

littéral-5 ALSO littéral-7

[ALSO littéral-8]...]

[CURRENCY SIGN IS littéral-9]

[DECIMAL - POINT IS COMMA].

INPUT – OUTPUTSECTION.

FILE – CONTROL.

SELECT [OPTIONAL] nom-de-fichier ASSIGN TO nom-de-réalisateur-1
[nom-de-réalisateur-2]...

[RESERVE entier-1 [AREA]]
[AREAS]]

[ORGANIZATION IS SEQUENTIAL] [ACCESS MODE IS SEQUENTIAL]
[FILE STATUS IS nom-de-donnée].

SELECT nom-de-fichier ASSIGN TO nom-de-réalisateur-1
[nom-de-réalisateur-2]...

[RESERVE entier-1 [AREA]]
[AREAS]]

ORGANIZATION IS RELATIVE

[ACCESS MODE IS {SEQUENTIAL [RELATIVE KEY IS nom-de-donnée-1]
RANDOM | RELATIVE KEY IS nom-de-donnée-1
DYNAMIC }]]
[FILE STATUS IS nom-de-donnée-2].

SELECT nom-de-fichier ASSIGN TO nom-de-réalisateur-1
[nom-de-réalisateur-2]...

[RESERVE entier-1 [AREA]]
[AREAS]]

ORGANIZATION IS INDEXED [ACCESS MODE IS {SEQUENTIAL
RANDOM
DYNAMIC }]]

RECORD KEY IS nom-de-donnée-1 [ALTERNATIVE RECORD KEY IS
nom-de-données-2 [WITH DUPLICATES]] ...
[FILE STATUS IS nom-de-donnée-3].

- 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). 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)

DATA DIVISION.

```
[FILE SECTION.
  [FD nom-de-fichier
    [
      [BLOCK CONTAINS [entier-1 TO] entier-2 {RECORDS
                                                CHARACTERS} ]
      [RECORD CONTAINS [entier-3 TO] entier-4 CHARACTERS]
      LABEL {RECORD IS } {STANDARD }
              {RECORDS ARE} {OMITTED }
      [
        [VALUE OF nom-de-réalisateur-1 IS {nom-de-donnée-1
                                                littéral-1 }
          [
            [nom-de-réalisateur-2 IS {nom-de-donnée-2
                                         littéral-2 } ] ...
          ]
        ]
      [DATA {RECORD IS } nom-de-donnée-3 [nom-de-donnée-4] ...
            {RECORDS ARE}
      ]
      [LINAGE IS {nom-de-donnée-5
                    entier-5 } LINES [WITH FOOTING AT {nom-de-donnée-6
                                                            entier-6 } ]
        [
          [LINES AT TOP {nom-de-donnée-7
                           entier-7 } ]
          [LINES AT BOTTOM {nom-de-donnée-8
                              entier-8 } ]
        ]
      [CODE - SET IS nom-d' alphabet]
      [
        [REPORT IS } nom-d' état-1 [nom-d' état-2] ...
        [REPORTS ARE}
      ]
      (rubrique de description d' enregistrements)
    ]
  ]
[SD nom-de-fichier [RECORD CONTAINS [entier-1 TO] entier-2 CHARACTERS]
  [
    [DATA {RECORD IS } nom-de-donnée-1 [nom-de-donnée-2] ...
    [RECORDS ARE}
  ]
  (rubrique de description d' enregistrements)
]
```

```
[WORKING - STORAGE SECTION.
  (rubrique de description d' enregistrement) ...
]
```

```
[LINKAGE SECTION.
  (rubrique de description d' enregistrement) ...
]
```

```
[COMMUNICATION SECTION.
  [Rubrique de description de communication]
  [Rubrique de description d' enregistrement] ...
]
```

```

REPORT SECTION.
RD nom-d' état
[CODE littéral-1]
[ { CONTROL IS } { nom-de-donnée-1 [nom-de-donnée-2] ... } ]
[ { CONTROLS ARE } { FINAL [nom-de-donnée-1 [nom-de-donnée-2] ...] } ]
[ PAGE [ { LIMIT IS } ] entier-1 [ { LINE } ] [HEADING entier-2]
  [FIRST DETAIL entier-3] [LAST DETAIL entier-4]
  [FOOTING entier-5] ] .

```

(rubrique de description de groupe d' édition)

- 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.
- Dans la REPORT SECTION, la clause RD donne des informations sur la structure physique des états imprimés, et indique la hiérarchie des niveaux⁴.
- Lorsqu'on écrit la clause CODE, littéral-1 est automatiquement placé dans les deux premiers caractères de chaque enregistrement, permettant ainsi d'identifier l'état.
- La clause CONTROL indique la hiérarchie des niveaux, en nommant les données produisant des ruptures de contrôle.
- La clause PAGE LIMIT indique les dimensions et la présentation d'une page physique.

⁴ La description du module d'impression dépasse de loin le cadre de cet ouvrage ; on pourra cependant se référer au livre de M. Koutchouk, *COBOL Perfectionnement et Pratique*, où tout un chapitre est consacré aux états.

Partie des traitements (PROCEDURE DIVISION)

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

```
PROCEDURE DIVISION [USING nom-de-donnée-1 [nom-de-donnée-2] ...].
```

```
[DECLARATIVES.  
  nom-de-section SECTION [numéro-de-segment].  
    phrase-déclarative  
  [nom-de-paragraphe. [phrase] ...] ...  
END DECLARATIVES.]  
nom-de-section SECTION [numéro-de-segment].  
[nom-de-paragraphe. [phrase] ...] ... ...
```

```
PROCEDURE DIVISION [USING nom-de-donnée-1 [nom-de-donnée-2] ...].
```

```
nom-de-paragraphe. [phrase]. ... ...
```

- 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, elles 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**.

$$88 \text{ nom-de-condition } \left\{ \begin{array}{l} \text{VALUE IS} \\ \text{VALUES ARE} \end{array} \right\} \text{littéral-1} \left[\begin{array}{l} \text{THROUGH} \\ \text{THRU} \end{array} \right] \text{littéral-2} \\ \text{littéral-3} \left[\begin{array}{l} \text{THROUGH} \\ \text{THRU} \end{array} \right] \text{littéral-4} \dots$$

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 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 noms de variables sont laissés au choix du programmeur. Les restrictions qui s'appliquent aux noms de variables sont décrites à la 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).

Caractères de formatage :

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

- Les formats 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 XXXXXXXXXX.
- 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**

77 N PIC IS S99999.

Définit une variable N à cinq positions, signée.

Si N contient 123, l'affichage sera : 00123

77 P PIC 999V99.

La variable P a trois positions entières et deux positions décimales, non signée. Si P contient 123.45 l'affichage sera : 13245

☒ Remarque :

Le rôle du point et de la virgule étant inversé aux USA par rapport à la France, on pensera à entrer les nombres décimaux avec un point.

2. données alphabétiques Symboles de formatage permis : **A B**

77 NOM PIC IS AAAAAA.

Définit un champ alphabétique NOM à 7 positions. L'affichage sera le contenu du champ aligné à gauche.

77 NOM PIC A(7).

Même effet pour le deuxième champ, mais écriture plus courte.

3. données alphanumériques⁵ Symboles de formatage permis : **X A 9**

77 CPVILLE PIC 9(5)A(20).

Définition d'une zone de données formée d'un nombre à cinq chiffres et de 20 caractères alphabétiques. Attention cependant aux *illegal character in numeric field* dans le code postal.

77 CPVILLE PIC X(15).

Le deuxième format est équivalent au premier, sans restriction.

4. champs numériques édités Symboles de formatage permis : **9 . + - , B 0 / Z ***

77 NOMBRE PIC +9999.99

Si NOMBRE contient 123.10 l'affichage sera : +0123.10

Attention : les calculs sont interdits sur ces formats.

5. champs alphanumériques édités Symboles de formatage permis : **9 . + - , B 0 / Z ***

77 TITRE PIC XBXBXBXB.

Définition d'un champ alphanumérique édité de 5 caractères séparés par des blancs. Si TITRE contient RAMBO : R A M B O

⁵ Les masques de formatage contenant exclusivement les signes A ou 9 ne sont pas alphanumériques.

Affectation de valeurs littérales et résultat à l'affichage

Format	Taille	Valeur affectée	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	0.10	
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	1993	
XB9B9B9	9	"TITRE"	TITRE	
X0X0X	5	"12345"	10203	

Qualification avec OF ou IN

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, on peut qualifier les données par OF ou IN suivi d'un nom de donnée de niveau supérieur.

Exemple 1-4

<pre> 01 ADRESSE-1. 02 NOM 03 PRENOM PICTURE X(20). 03 NOMFAM PICTURE X(20). 02 LIEU 03 RUE PICTURE 999X(20). 03 CPOSTAL PICTURE 99999. 03 VILLE PICTURE X(30). </pre>	<pre> 01 ADRESSE-2. 02 PRENOM PICTURE X(20). 02 NOMFAM PICTURE X(20). 02 RUE PICTURE 999X(20). 02 CPOSTAL PICTURE 99999. 02 VILLE PICTURE X(30). </pre>
---	--

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.

Remarque :

On pourra même utiliser la qualification pour les noms de condition ou les noms de paragraphe :

$$\text{nom-de-paragraphe} \left[\left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\} \text{nom-de-section} \right]$$

REDEFINES

Permet d'utiliser la même zone mémoire pour stocker des données différentes, de noms différents.

Attention : L'emplacement en mémoire est le même !

$$\text{Numéro-de-niveau} \left\{ \begin{array}{l} \text{nom-de-donnée-1} \\ \text{FILLER} \end{array} \right\} [\text{REDEFINES nom-de-donnée-2}]$$

$$\left[\begin{array}{l} \text{PICTURE} \\ \text{PIC} \end{array} \right\} \text{IS chaîne-de-caractères} .$$

- 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 1-5

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

résultat : 0123/0123

Exemple 1-6

```
01 UNE-DATE PICTURE X(8).
01 DATE-BIS REDEFINES UNE-DATE.
02 JOUR PICTURE 99.
02 FILLER PICTURE X.
02 MOIS PICTURE 99.
02 FILLER PICTURE X.
02 ANNEE PICTURE 99.
```

RENAMES

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

$$66 \text{ nom-de-donnée-1 } \text{RENAMES} \text{ nom-de-donnée-2 } \left[\begin{array}{l} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{ nom-de-donnée-3} .$$

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

Exemple 1-7

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

Niveau 88 : Noms-de-condition

La définition de noms-de-condition permet de donner des noms à des valeurs fixes de champs. On peut directement tester ces noms avec la clause `IF` ce qui permet des branchements plus élégants. Un nom-de-condition a la valeur booléenne *vrai* si le champ de niveau supérieur contient une des valeurs définies dans la clause `VALUE` associée au nom-de-condition⁶.

$$88 \text{ nom-de-condition} \left\{ \begin{array}{l} \text{VALUE IS} \\ \text{VALUES ARE} \end{array} \right\} \text{littéral-1} \left[\left\{ \begin{array}{l} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{littéral-2} \right] \\ \text{littéral-3} \left[\left\{ \begin{array}{l} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{littéral-4} \right]. \dots$$

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

Exemple 1-8

```
...
WORKING-STORAGE SECTION.
01 age PICTURE 999 VALUE ZERO.
   88 enfant      VALUE 1 THROUGH 17.
   88 jeune       VALUE 18 THROUGH 25.
   88 adulte      VALUE 26 THROUGH 65.
   88 vieux       VALUE 66 THROUGH 200.
   88 plaisantin  VALUE 201 THROUGH 999.

PROCEDURE DIVISION.
Recompenser-Personne.
    PERFORM Demander-Age-Personne
        UNTIL age NUMERIC AND age NOT ZERO AND NOT plaisantin.7

    IF enfant DISPLAY "Tu reçois une sucette, mon petit.".
    IF jeune  DISPLAY "Vous recevez un compilateur COBOL.".
    IF adulte DISPLAY "Vous recevez le débogueur associé.".
    IF vieux  DISPLAY "Vous recevez la solution du problème.".
    STOP RUN.

Demander-Age-Personne.
    DISPLAY "Quel âge avez-vous ? (1-200) " WITH NO ADVANCING.
    ACCEPT age.
```

⁶ Le COBOL 85 ajoute une option relative aux variables logiques à l'instruction `SET` (cf. index).

`SET nom-condition-1 [nom-condition-2] ... TO TRUE.`

La variable associée au nom-condition est initialisée en conformité avec la clause `VALUE` associée, de façon à rendre vraie la valeur logique du nom-condition. Si plusieurs littéraux suivent la clause `VALUE`, c'est la première qui est choisie.

⁷ La structure `PERFORM UNTIL` sera étudiée plus loin, au chapitre *Programmation en COBOL*.

Elargissement de la description de données

BLANK WHEN ZERO

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`.

- 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 champs numériques édités, on ne pourra pas l'utiliser conjointement à `BLANK WHEN ZERO`.

Exemple 1-9

```
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. Cependant, dans la deuxième ligne, on est limité dans la définition du format (obligatoirement **Z**).

FILLER

Remplit un champ de nom `FILLER` avec des blancs ou du texte destinés à améliorer la présentation d'une page d'écran ou d'un état imprimé. `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.

- 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 1-10

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

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

Exemple 1-11

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          PICTURE X(12).

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

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

JUSTIFIED RIGHT

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

$$\left. \begin{array}{l} \text{JUSTIFIED} \\ \text{JUST} \end{array} \right\} \text{RIGHT}$$

- La clause `JUSTIFIED RIGHT` est placée après la description de donnée dans la `WORKING-STORAGE SECTION`.
- Cette clause n'est autorisée que pour les champs alphanumériques et alphanumériques.
- `JUSTIFIED` ne s'applique qu'aux zones élémentaires et non pas aux groupes de données pourtant considérés implicitement alphanumériques.

Exemple 1-12

```
77  champ-1 PICTURE XXXXXX.
...

MOVE ABC TO champ-1.
DISPLAY champ-1.
```

Résultat :

ABC□□□

```
77  champ-2 PIC XXXXXX JUSTIFIED RIGHT.
...

MOVE ABC TO champ-2.
DISPLAY champ-2.
```

Résultat :

□□□ABC

SIGN

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.

$$[\text{SIGN IS}] \left\{ \begin{array}{l} \text{LEADING} \\ \text{TRAILING} \end{array} \right\} [\text{SEPARATE CHARACTER}]$$

- 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.

SYNCHRONIZED

La mémoire d'un ordinateur est organisée en cellules adressables. Selon les machines et le format interne des données, l'alignement d'une donnée élémentaire sur une frontière de cellule permet d'optimiser le programme résultant en évitant au compilateur de générer des instructions de cadrage. La clause `SYNCHRONIZED` permet d'aligner une donnée élémentaire sur une de ces bornes naturelles de la mémoire ; elle est cependant peu utilisée car pratiquement incompatible d'une machine à l'autre.

$$\left\{ \begin{array}{l} \text{SYNCHRONIZED} \\ \text{SYNC} \end{array} \right\} \left[\begin{array}{l} \text{LEFT} \\ \text{RIGHT} \end{array} \right]$$

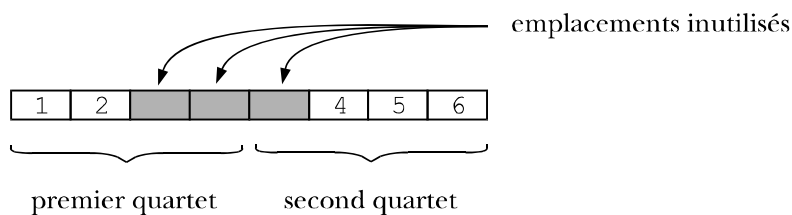
- La clause `SYNCHRONIZED` ne s'applique qu'aux données élémentaires.
- `LEFT` (respectivement `RIGHT`) indique que la donnée est positionnée de façon à commencer à la borne gauche (respectivement droite). Le cas échéant, des zones de mémoires inutilisées sont intercalées entre les données, selon des règles définies par le constructeur.

Exemple 1-13

Supposons que nous utilisons un ordinateur dans lequel les caractères sont rangés en mémoire par groupe de quatre. La description suivante :

```
01 groupe.  
02 zone-1 PICTURE 9(2) SYNCHRONIZED LEFT VALUE 12.  
02 zone-2 PICTURE 9(3) SYNCHRONIZED RIGHT VALUE 456.
```

provoquera l'implantation en mémoire illustrée ci-dessous :



Remarque :

La clause `SYNCHRONIZED` agit sur la représentation interne des données, mais n'agit pas sur le cadrage des données à l'intérieur de la zone elle-même : c'est la clause `JUSTIFIED` qui effectue ce cadrage, si l'on ne veut pas employer le cadrage normal.

USAGE

La clause USAGE sert à la représentation interne des données.

`USAGE IS` {
 DISPLAY
BINARY
COMPUTATIONAL
PACKED – DECIMAL
INDEX

- La clause USAGE n'est pas autorisée pour les niveaux 66 et 88.
- Si la clause USAGE est écrite derrière un champ de données *groupe* de niveau 01, elle s'applique à chaque zone élémentaire subordonnée à *groupe*⁸.
- La clause USAGE d'un champ de données élémentaire ne doit pas contredire celle du groupe auquel il appartient.
- `USAGE IS DISPLAY` correspond aux codifications ASCII (American Standard Code for Information Interchange) ou EBCDIC, avec un caractère par octet. C'est le codage par défaut.
- `USAGE IS BINARY` précise une représentation en base deux.
- `USAGE IS PACKED-DECIMAL` indique que la représentation s'effectue en base dix tout en utilisant le nombre minimum de caractères (en général un chiffre représenté sur quatre bits).
- Pour respecter les règles de frontières imposées par la mémoire de l'ordinateur, les longueurs de zones binaires numériques (BINARY et COMPUTATIONAL) sont normalisées sur 2, 4 et 8 octets.

Quantités binaires sur 2 octets : Elles correspondent aux images de 4 chiffres au maximum. Par exemple S9 (4). La valeur maximum pouvant être en fait stockée sur 2 octets est 32767.

Quantités binaires sur 4 octets : Elles correspondent aux images de 9 chiffres au maximum. Par exemple S9 (7) V99. La valeur maximum pouvant être stockée sur 4 octets est 2 147 483 647.

Quantités binaires sur 8 octets : Elles correspondent aux nombres de plus de 9 chiffres jusqu'à un maximum de 18 chiffres, limite imposée par le Cobol.

Pour un nombre compris dans une tranche, on aura toujours intérêt à lui donner la taille maximum pour éviter les dépassements de capacité dans les calculs intermédiaires, d'autant plus que le compilateur prendra de toute façon des zones de 2, 4 ou 8 octets.

 Remarque :

La clause USAGE agissant sur le format interne des données, elle n'affecte pas l'utilisation normale des données dans le programme. Certains constructeurs en ont profité pour ajouter aux représentations normalisées d'autres représentations. A titre d'exemple citons celles de l'IBM 370.

⁸ Rappelons que *groupe* est implicitement de format alphanumérique, de ce fait il n'est pas admis dans les calculs.

DISPLAY : Chaque caractère numérique sera représenté sur un groupe de huit bits (positions binaires) appelé octet. Le signe éventuel sera codé sur la partie gauche de l'octet le plus à droite.

77 NUM-DISP PICTURE 9(3) USAGE DISPLAY VALUE 128.	11110001 11110010 11111000
---	----------------------------

COMPUTATIONAL : La donnée sera représentée en binaire, sur 2, 4 ou 8 octets selon le nombre de caractères numériques suivant PICTURE. Le signe sera le bit le plus à gauche, les nombres négatifs étant représentés en complément à 2.

77 NUM-DISP PICTURE S9(3) USAGE COMP VALUE 128.	00000000 10000000
---	-------------------

COMPUTATIONAL-1 et COMPUTATIONAL-2 : Ces options indiquent toutes deux une représentation interne des données en virgule flottante, c'est-à-dire à l'aide d'un exposant et d'une mantisse, respectivement en simple précision (COMP-1) ou double précision (COMP-2).

COMPUTATIONAL-3 : Dans cette notation, non normalisée, chaque chiffre sera représenté sur un demi-octet, le demi-octet le plus à droite figurant le signe. On appelle ce format *décimal interne* ou *décimal condensé*. Ce format est en fait le même que PACKED-DECIMAL (ajouté à la norme en 1985).

77 NUM-DISP PICTURE S9(3) USAGE COMP-3 VALUE 128.	00010010 10001111
---	-------------------

VALUE

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.

- 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 1-14

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

Programmation en COBOL

Partie I - Aide à l'écriture	27
COPY	27
REPLACE	27
Notation par référence	28
Partie II - Fonctions d'entrées-sorties	28
ACCEPT	28
DISPLAY	29
Partie III - Fonctions d'affectation	30
MOVE	30
MOVE CORRESPONDING	31
INSPECT	32
STRING	34
UNSTRING	34
Partie IV - Initialisation de données	36
INITIALIZE	36
Partie V - Opérations arithmétiques	37
ADD	37
SUBTRACT	38
MULTIPLY	39
DIVIDE	39
COMPUTE	41
Partie VI - La condition	42
EVALUATE	42
IF	44
Partie VII - Instructions de saut	46
GO TO	46
Branchements modifiés - ALTER	47
STOP	47
Partie VIII - Appel de procédures	48
PERFORM	48
EXIT	51
Partie IX - Appel de modules externes	52
CALL	52
CANCEL	54
ENTRY	55
EXIT PROGRAM	55
GOBACK	55
Partie X - Programmes contenus	56

Programmation COBOL



Aide à l'écriture

COPY

COPY permet l'incorporation de texte dans un programme source. On l'utilise en particulier pour insérer des en-têtes standards dans plusieurs programmes. Cela nous épargne une recopie fastidieuse, évite les erreurs, les oublis et facilite la maintenance. Il est possible de créer des bibliothèques contenant des éléments de programme. La création et la gestion de ces bibliothèques étant définies par le constructeur, nous ne verrons que leur utilisation par la clause COPY.

$$\text{COPY nom-de-texte} \left[\begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right] \text{nom-de-bibliothèque}$$
$$\left[\text{REPLACING} \left\{ \begin{array}{l} \text{== pseudo-texte-1 ==} \\ \text{identificateur-1} \\ \text{littéral-1} \\ \text{mot-1} \end{array} \right\} \text{BY} \left\{ \begin{array}{l} \text{== pseudo-texte-2 ==} \\ \text{identificateur-2} \\ \text{littéral-2} \\ \text{mot-2} \end{array} \right\} \right].$$

- Si plusieurs bibliothèques sont disponibles au cours de la compilation, le nom de la bibliothèque contenant le texte doit être indiqué après OF ou IN (qualification).
- A l'intérieur d'une bibliothèque, le nom de texte doit être unique.
- pseudo-texte-1 ne doit pas être vide ou constitué uniquement d'espaces ou de commentaires. En revanche, pseudo-texte-2 peut être vide.
- Le séparateur == suit les mêmes règles que pour les parenthèses : []==texte==[]
- Le texte à inclure ne doit pas contenir lui-même la clause COPY.
- Si la clause REPLACING est spécifiée, chaque occurrence de pseudo-text-1, identificateur-1, littéral-1 ou mot-1 sera remplacée par pseudo-texte-2, identificateur-2, littéral-2 ou mot-2. Les lignes de commentaire et de débogage restent inchangées.

REPLACE

Le COBOL 85 ajoute l'instruction REPLACE permettant de modifier le texte d'un programme source. Il s'agit en fait d'une version *light* de la clause COPY qui agit directement sur le code source.

REPLACE == pseudo-texte-1 == BY == pseudo-texte-2 == ...

- Chaque occurrence de pseudo-texte-1 est remplacée par pseudo-texte-2 jusqu'à la prochaine instruction REPLACE si elle existe, sinon jusqu'à la fin du programme. Ce traitement est effectué après celui des éventuelles instructions COPY.

Notation par référence

Le COBOL 85 permet de référencer une partie d'un champ de données en précisant une position et une longueur – un peu à la façon de la fonction MID\$ du BASIC.

nom-de-donnée (position : [longueur])

- nom-de-donnée doit être en USAGE DISPLAY, position et longueur doivent être des expressions arithmétiques comprises dans les intervalles suivants :

$0 < \text{position} \leq \text{nombre de caractères de nom-de-donnée}$

$0 < \text{longueur} < \text{nombre de caractères de nom-de-donnée} - \text{position} + 1$

- Si la longueur est omise, la partie référencée s'étend jusqu'à la fin de la zone.

Exemple 2-1

```
01 ZONE PICTURE X(10) VALUE "ABCDEFGHIJ".
```

ZONE (3:4) permet d'accéder au contenu "CDEF"

ZONE (9:) permet d'accéder au contenu "IJ"

Fonctions d'entrées-sorties

ACCEPT

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

ACCEPT identificateur $\left[\text{FROM} \left\{ \begin{array}{l} \text{DATE} \\ \text{DAY} \\ \text{TIME} \\ \text{DAY - OF - WEEK} \end{array} \right\} \right]$.

- DATE est la date système définie en PIC 9(6), de la forme : AAMMJJ (année-mois-jour).
- DAY est le jour système défini en PIC 9(5), de la forme : AANN (année, jour dans l'année). Ainsi le 1^{er} juillet 1968 sera codé : 68183.
- TIME est l'heure système définie en PIC 9(8), de la forme : 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.
- DAY-OF-WEEK est défini en PIC 9 et représente le rang du jour de la semaine. Ainsi 1 représente lundi, et 7 signifie dimanche.

- Certains compilateurs acceptent le format :

ACCEPT identificateur [AT LINE y COL x] [WITH attribut].

Les attributs peuvent varier suivant l'implémentation, par exemple :

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

DISPLAY

Permet d'afficher des données à l'écran.

DISPLAY { identificateur-1 } [{ identificateur-2 }] ... [UPON nom-mnémonique].
 { littéral-1 } [{ littéral-2 }]

- nom-mnémonique désigne l'unité périphérique, spécifiée dans le paragraphe **SPECIAL-NAMES** de l'**ENVIRONMENT DIVISION**, où seront adressées les données.

Exemple 2-2

DISPLAY "Votre note d'examen est : " résultat "/20" **UPON** **TERMINAL**.

- Certains compilateurs acceptent le format :

DISPLAY { identificateur-1 } [AT LINE y COL x] [WITH attribut].
 { littéral-1 }

- 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.

Exemples d'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

Mouvements de données

MOVE

Transfert de données ou de groupes de données vers un autre champ :

identificateur-2 et identificateur-3 reçoivent identificateur-1.

MOVE { identificateur-1 } TO identificateur-2 [identificateur-3]...

- 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 valeurs transmises dans une **zone réceptrice numérique** sont alignées sur le 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 gauche, sauf si la clause JUSTIFIED-RIGHT est précisée. Si la zone de réception est :
 - **trop grande**, on remplit avec des blancs,
 - **trop petite**, la chaîne est tronquée.
 - Si l'on affecte une valeur numérique signée à un champ numérique non signé, la valeur reçue est sa valeur absolue.

Exemple 2-3

zone d'émission		zone de réception	
Masque de données	Contenu	Masque de données	Contenu
99999	12345	99999999	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 à l'exécution !)

zone d'émission	zone de réception					
	num-entier	num-frac	num-édité	alpha	alphanum	alphanum-édité
num-entier	+	+	+	-	+	+
num-frac	+	+	+	-	-	-
num-édité	-	-	-	-	+	+
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

Contrairement à MOVE qui traite les groupes de données comme un seul champ, MOVE CORRESPONDING est un transfert de données *intelligent*. Néanmoins, cet avantage peut s'avérer négligeable face à la quantité de qualifications (par ex. JOUR OF DATE-SYS) qu'il faudra traîner tout au long du programme.

MOVE { CORRESPONDING } identificateur-1 TO identificateur-2.
CORR

- 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.
- Toute zone élémentaire décrite avec une clause OCCURS, REDEFINES ou RENAME est ignorée.

Exemple 2-4

```

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.
Affiche-Date-Syst.
    ACCEPT DATE-SYS FROM DATE.
    MOVE CORRESPONDING DATE-SYS TO DATE-AFF.
    DISPLAY DATE-AFF.
  
```

INSPECT

Le verbe `INSPECT` permet de compter le nombre de fois qu'une configuration particulière apparaît dans une zone de données et de remplacer ces caractères par d'autres.

`INSPECT` identificateur-1

$$\left[\begin{array}{l} \text{TALLYING identificateur-2 FOR } \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{CHARACTERS} \end{array} \right\} \left\{ \begin{array}{l} \text{identificateur-3} \\ \text{littéral-1} \end{array} \right\} \end{array} \right]$$
$$\left[\begin{array}{l} \text{REPLACING } \left\{ \begin{array}{l} \text{ALL} \\ \text{LEADING} \\ \text{FIRST} \end{array} \right\} \left\{ \begin{array}{l} \text{identificateur-4} \\ \text{littéral-2} \end{array} \right\} \text{ BY } \left\{ \begin{array}{l} \text{identificateur-5} \\ \text{littéral-3} \end{array} \right\} \\ \text{CHARACTERS BY } \left\{ \begin{array}{l} \text{identificateur-6} \\ \text{littéral-4} \end{array} \right\} \end{array} \right]$$
$$\left[\text{CONVERTING } \left\{ \begin{array}{l} \text{identificateur-7} \\ \text{littéral-5} \end{array} \right\} \text{ TO } \left\{ \begin{array}{l} \text{identificateur-8} \\ \text{littéral-6} \end{array} \right\} \right]$$
$$\left[\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{ INITIAL } \left\{ \begin{array}{l} \text{identificateur-9} \\ \text{littéral-7} \end{array} \right\} \right] \dots$$

- Identificateur-1, donnée sur laquelle porte l'opération, doit être un groupe ou une donnée élémentaire d'usage `DISPLAY`.
- Identificateur-2, compteur lorsqu'on utilise l'option `TALLYING`, doit représenter une donnée élémentaire numérique, et doit être initialisé avant l'instruction `INSPECT`.
- Les autres identificateurs doivent être des données élémentaires d'usage `DISPLAY`.
- Les littéraux doivent être non numériques, et peuvent être des constantes figuratives (sauf `ALL`).
- Les options `TALLYING` et `REPLACING` peuvent être utilisées conjointement, dans ce cas l'instruction équivaut à un `INSPECT TALLYING` suivi de `INSPECT REPLACING`.
- `BEFORE` : la comparaison commence au premier caractère à gauche d'identificateur-1 et s'arrête dès que le délimiteur est rencontré, celui-ci n'intervenant pas dans le cycle de comparaison. Si ce délimiteur n'est pas rencontré, tout se passe comme si l'option `BEFORE` n'avait pas été écrite.
- `AFTER` : la comparaison commence au premier caractère suivant la fin de zone ayant provoqué l'égalité avec le contenu du délimiteur ; si ce délimiteur n'est pas trouvé dans identificateur-1, la comparaison ne peut alors avoir lieu.

TALLYING

- `ALL` : le compteur identificateur-2 est augmenté de 1 chaque fois qu'une occurrence d'identificateur-3 ou de littéral-1 est contenue dans identificateur-1.

- **LEADING** : le compteur identificateur-2 est augmenté de 1 pour toute occurrence contiguë d'identificateur-3 ou de littéral-1 repérée au début d'identificateur-1.
- **CHARACTERS** : le contenu du compteur identificateur-2 est incrémenté pour chaque caractère rencontré dans identificateur-1.

REPLACING

- **ALL** : toutes les occurrences d'identificateur-4 (ou de littéral-2) repérées dans l'identificateur-1 sont remplacées par le contenu d'identificateur-5 (ou de littéral-3).
- **LEADING** : seules les occurrences contiguës d'identificateur-4 (ou de littéral-2) sont remplacées par identificateur-5 (ou littéral-3) à condition qu'aucune comparaison négative n'ait été effectuée auparavant.
- **FIRST** : seule la première occurrence d'identificateur-4 (ou de littéral-2) est remplacée par le contenu d'identificateur-5 (ou de littéral-3). La longueur des deux zones doit être la même !
- **CHARACTERS** : chaque caractère d'identificateur-1 est remplacé par le contenu d'identificateur-5 (ou de littéral-3), celui-ci devant être d'une longueur d'un caractère.

CONVERTING

- Cette instruction est exécutée comme si l'on avait écrit une série d'INSPECT avec l'option REPLACING dans laquelle chaque apparition, dans identificateur-1, d'un caractère appartenant à identificateur-7 ou littéral-5 était remplacée par le caractère de rang correspondant dans identificateur-8 ou littéral-6.
- La taille d'identificateur-7 ou de littéral-5 doit être égale à la taille d'identificateur-8 ou de littéral-6.

Exemple 2-5

```

77  ZONE PICTURE X(8) VALUE "AAABCCAA".
77  CTR1 PICTURE 9(2) VALUE ZERO.

INSPECT ZONE TALLYING CTR1 FOR ALL "AA"          | CTR1=02
      REPLACING ALL "AA" BY "BB".                | ZONE="BBABCCBB"

INSPECT ZONE TALLYING CTR1 FOR LEADING "A"        | CTR1=03
      BEFORE INITIAL "B".                        |

INSPECT ZONE TALLYING CTR1 FOR LEADING "A"        | CTR1=02
      AFTER INITIAL "CC".                        |

INSPECT ZONE TALLYING CTR1 FOR ALL "A"            | CTR1=05
      REPLACING ALL "A" BY "O"                  | ZONE="AAABCCOO"
      AFTER INITIAL "C".                        |

77  ZONE PICTURE X(10) VALUE "ABACADXXAB".

INSPECT ZONE CONVERTING "CAB" TO "123" BEFORE "XX". | ZONE="23212DXXAB"

```

STRING

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

```
STRING {identificateur-1} [ {identificateur-2} ] ...  
      {littéral-1} [ {littéral-2} ]  
  
      DELIMITED BY {identificateur-3}  
                   {littéral-3}  
                   SIZE ...  
  
      INTO identificateur-4 [ WITH POINTER identificateur-5 ]  
      [ ON OVERFLOW instruction-impérative-1 ]  
      [ NOT ON OVERFLOW instruction-impérative-2 ] [ END - STRING ].
```

- 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.
- Identificateur-7 doit correspondre à 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 identificateur-3 ou littéral-3 (les caractères délimitant identificateur-3 ou littéral-3 ne sont pas transmis) ou par la taille (option SIZE) de la zone réceptrice.
- L'option POINTER permet de préciser la position de gauche du transfert en zone réceptrice. Identificateur-5, zone élémentaire numérique, doit alors avoir été chargé avec la valeur correspondante. Au fur et à mesure, le pointeur identificateur-5 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 identificateur-5 est inférieur à 1 ou supérieur au nombre de caractères de la zone réceptrice.

UNSTRING

Eclate une chaîne de caractères en plusieurs sous-chaînes.

```
UNSTRING identificateur-1 [ DELIMITED BY [ ALL ] {identificateur-2}  
                           {littéral-1} ] ...  
                           [ [ OR [ ALL ] {identificateur-3} ]  
                           {littéral-2} ] ]  
  
      INTO identificateur-4  
      [ DELIMITER IN identificateur-5 ] [ COUNT IN identificateur-6 ]
```

```
[identificateur-7
[DELIMITER IN identificateur-8] [COUNT IN identificateur-9]] ...
```

```
[WITH POINTER identificateur-10]
[TALLYING IN identificateur-11]
```

```
[ON OVERFLOW instruction-impérative]
[NOT ON OVERFLOW instruction-impérative] [END - UNSTRING].
```

- Les règles de transfert de données sont identiques à celles du verbe STRING. Identificateur-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 identificateur-2, littéral-1, identificateur-3, littéral-2, 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.
- DELIMITER IN et COUNT IN ne peuvent être spécifiées qu'avec l'option DELIMITED BY.
- POINTER : identificateur-10, zone numérique élémentaire, permet de compter le nombre de caractères examinés dans la zone émettrice (ne pas oublier de l'initialiser à zéro).
- TALLYING : identificateur-11, zone numérique élémentaire, permet de compter le nombre de zones émettrices créées (doit être initialisé).
- L'option OVERFLOW arrête le déroulement de l'instruction UNSTRING dans deux cas :
 - a) Lorsque le contenu du pointeur identificateur-10 est négatif ou supérieur à la taille de la zone émettrice identificateur-1.
 - b) Lorsque toute la zone émettrice n'a pas été examinée et qu'il n'y a plus de zone réceptrice disponible.

Si une de ces conditions se produit sans que l'option OVERFLOW ait été écrite, l'instruction UNSTRING se termine et l'instruction suivante est exécutée.

- Identificateur-4 et identificateur-7, sont les zones réceptrices de l'éclatement. Lorsque des délimiteurs ont été spécifiés, on peut les recueillir dans les zones identificateur-5 et identificateur-8 et compter le nombre de caractères transférés dans identificateur-6 et identificateur-9.

Exemple 2-6

```
77 zone    PICTURE X(12).
77 nom     PICTURE X(5).
77 annee   PICTURE X(4).
```

```
STRING "ANNEE" SPACE "1974" DELIMITED BY SIZE INTO ZONE.
UNSTRING zone DELIMITED BY SPACE INTO nom annee.
```

```
zone =
"ANNEE  1974  "

nom = "ANNEE"

annee = "1974"
```

Initialisation de données

INITIALIZE

En COBOL 85, l'instruction `INITIALIZE` permet d'initialiser ou de réinitialiser certains champs d'un type déterminé avec des valeurs données.

`INITIALIZE` identificateur-1 ...

$$\left[\begin{array}{c} \text{REPLACING} \left\{ \begin{array}{l} \text{ALPHABETIC} \\ \text{ALPHANUMERIC} \\ \text{NUMERIC} \\ \text{ALPHANUMERIC - EDITED} \\ \text{NUMERIC - EDITED} \end{array} \right\} \text{ DATA BY } \left\{ \begin{array}{l} \text{identificateur-2} \\ \text{littéral-1} \end{array} \right\} \dots \end{array} \right].$$

- Identificateur-1 est le récepteur. Il ne doit pas être un index, ni contenir l'option `DEPENDING ON` ou la clause `RENAMES`. Identificateur-2 et littéral-2 sont l'émetteur. Chaque catégorie précisée après `REPLACING` ne peut apparaître qu'une fois dans l'instruction, et doit être compatible avec la description de l'émetteur.
- Si l'option `REPLACING` n'est pas spécifiée, les données élémentaires alphabétiques ou alphanumériques appartenant au(x) récepteur(s) sont initialisées à blanc et les données élémentaires numériques sont initialisées à zéro.
- Si l'option `REPLACING` est précisée, l'instruction `INITIALIZE` fonctionne comme une série d'instructions `MOVE` entre l'émetteur et les données élémentaires du récepteur appartenant à la catégorie précisée.

Exemple 2-7

```
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.
```


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 !
- Le COBOL 85 a introduit la clause `NOT ON SIZE ERROR` qui est le pendant de la précédente, c'est-à-dire que dans le cas où l'opération s'est bien déroulée, on doit exécuter l'ordre impératif.
- Autre nouveauté : les délimiteurs d'instruction `END-ADD`, `END-SUBTRACT`, `END-MULTIPLY`, `END-DIVIDE`, `END-COMPUTE` qui indiquent, pour le cas où il y aurait un doute, que le texte d'une instruction est terminé.

ADD

$$\text{ADD } \left\{ \begin{array}{l} \text{identificateur-1} \\ \text{littéral-1} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{identificateur-2} \\ \text{littéral-2} \end{array} \right\} \right] \dots$$

`TO` `identificateur-m` `[ROUNDED]`

`[identificateur-n [ROUNDED]]...`

`[ON SIZE ERROR instruction-impérative-1]`

`[NOT ON SIZE ERROR instruction-impérative-2] [END - ADD].`

$$\text{ADD } \left\{ \begin{array}{l} \text{identificateur-1} \\ \text{littéral-1} \end{array} \right\} \left\{ \begin{array}{l} \text{identificateur-2} \\ \text{littéral-2} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{identificateur-3} \\ \text{littéral-3} \end{array} \right\} \right] \dots$$

`GIVING` `identificateur-m` `[ROUNDED]`

`[identificateur-n [ROUNDED]]...`

`[ON SIZE ERROR instruction-impérative-1]`

`[NOT ON SIZE ERROR instruction-impérative-2] [END - ADD].`

$$\text{ADD } \left\{ \begin{array}{l} \text{CORRESPONDING} \\ \text{CORR} \end{array} \right\} \text{identificateur-1 } \text{TO } \text{identificateur-2 } \text{[ROUNDED]}$$

`[ON SIZE ERROR instruction-impérative-1]`
`[NOT ON SIZE ERROR instruction-impérative-2] [END - ADD].`

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

- ADD CORRESPONDING fonctionne de la même façon que MOVE CORRESPONDING, les contenus des zones élémentaires de même nom étant cette fois additionnés. Rappelons que les zones décrites avec les clauses OCCURS, REDEFINES ou RENAMEs sont ignorées.

Exemple 2-8

```

77 A PIC 9V99 VALUE 1.19.  || ADD 1 C TO A B ROUNDED.  | A = 12.18 (1+C+A=12.18)
77 B PIC 99V9 VALUE 5.7.   ||                          | B = 16.7  (1+C+B=16.69)
77 C PIC 9V99 VALUE 9.99.  || ADD A B C GIVING D.   | D = 6.8   (A+B+C=16.88)
77 D PIC 9.9.              ||                          |

```

ADD A B C GIVING D ON SIZE ERROR MOVE ZERO TO D. | D = 0 (erreur !)

SUBTRACT

SUBTRACT {identificateur-1} [{identificateur-2}]
 {littéral-1} [{littéral-2}]
 FROM identificateur-m [ROUNDED]
 [identificateur-n [ROUNDED]]...
 [ON SIZE ERROR instruction-impérative-1]
 [NOT ON SIZE ERROR instruction-impérative-2] [END - SUBTRACT].

SUBTRACT {identificateur-1} [{identificateur-2}]...
 {littéral-1} [{littéral-2}]...
 FROM {identificateur-m}
 {littéral-m} GIVING identificateur-n [ROUNDED]
 [identificateur-o [ROUNDED]]...
 [ON SIZE ERROR instruction-impérative-1]
 [NOT ON SIZE ERROR instruction-impérative-2] [END - SUBTRACT].

SUBTRACT {CORRESPONDING}
 {CORR} identificateur-1 FROM identificateur-2 [ROUNDED]
 [ON SIZE ERROR instruction-impérative-1]
 [NOT ON SIZE ERROR instruction-impérative-2] [END - SUBTRACT].

Exemple 2-9

```

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

MULTIPLY

```

MULTIPLY {identificateur-1}
          {littéral-1} BY identificateur-2 [ROUNDED]
          [identificateur-3 [ROUNDED]] ...
          [ON SIZE ERROR instruction-impérative-1]
          [NOT ON SIZE ERROR instruction-impérative-2] [END - MULTIPLY].
  
```

```

MULTIPLY {identificateur-1} BY {identificateur-2}
          {littéral-1}        {littéral-2}
          GIVING identificateur-3 [ROUNDED]
          [identificateur-4 [ROUNDED]] ...
          [ON SIZE ERROR instruction-impérative-1]
          [NOT ON SIZE ERROR instruction-impérative-2] [END - MULTIPLY].
  
```

Exemple 2-10

```

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

```

DIVIDE {identificateur-1}
        {littéral-1} INTO identificateur-2 [ROUNDED]
        [identificateur-3 [ROUNDED]] ...
        [ON SIZE ERROR instruction-impérative-1]
        [NOT ON SIZE ERROR instruction-impérative-2] [END - DIVIDE].
  
```

```

DIVIDE {identificateur-1} INTO {identificateur-2}
        {littéral-1}          {littéral-2}
        GIVING identificateur-3 [ROUNDED]
        [identificateur-4 [ROUNDED]] ...
        [ON SIZE ERROR instruction-impérative-1]
        [NOT ON SIZE ERROR instruction-impérative-2] [END - DIVIDE].
  
```

```

DIVIDE {identificateur-1} BY {identificateur-2}
      {littéral-1} {littéral-2}
      GIVING identificateur-3 [ROUNDED]
      [identificateur-4 [ROUNDED]] ...
      [ON SIZE ERROR instruction-impérative-1]
      [NOT ON SIZE ERROR instruction-impérative-2] [END - DIVIDE].

```

```

DIVIDE {identificateur-1} INTO {identificateur-2}
      {littéral-1} {littéral-2}
      GIVING identificateur-3 [ROUNDED]
      REMAINDER identificateur-4
      [ON SIZE ERROR instruction-impérative-1]
      [NOT ON SIZE ERROR instruction-impérative-2] [END - DIVIDE].

```

```

DIVIDE {identificateur-1} BY {identificateur-2}
      {littéral-1} {littéral-2}
      GIVING identificateur-3 [ROUNDED]
      REMAINDER identificateur-4
      [ON SIZE ERROR instruction-impérative-1]
      [NOT ON SIZE ERROR instruction-impérative-2] [END - DIVIDE].

```

- Les champs résultats peuvent être de format édité.
- 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 2-11

```

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.
```

```
DIVIDE A BY B GIVING C REMAINDER D.
```

```
DIVIDE C BY A GIVING D ROUNDED
      REMAINDER C.
```

B = 0	(B/A=0.075)
C = 0.5	(C/A=0.45)
B = 13.3	(A/B=13.3333333)
D = 0	(A-B*C=20-19.95=0.05)
D = 0.4	(C/A=0.45)
C = 1	(C-A*D=9-20*0.4=9-8=1)
D = 0.5	(arrondi après le calcul du résultat !)

COMPUTE

Traite des formules entières. Les opérateurs sont :

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

```
COMPUTE identificateur-1 [ROUNDED] [identificateur-2 [ROUNDED]]...
    = expression-arithmétique
    [ON SIZE ERROR instruction-impérative-1]
    [NOT ON SIZE ERROR instruction-impérative-2] [END - COMPUTE].
```

- 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. Identificateur-1, en revanche, peut être de format édité.
- COMPUTE devrait être réservé aux formules complexes, car les calculs se font plus rapidement avec les instructions arithmétiques de base (ADD, SUBTRACT, etc.)

Exemple 2-12

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 !".	$X = \frac{1}{X}$

La condition

EVALUATE

En COBOL 85, l'instruction `EVALUATE` permet de généraliser la notion de structure alternative en permettant d'évaluer des conditions multiples, et d'exécuter des instructions différentes selon le résultat de l'évaluation.

```

EVALUATE {
  {
    {identificateur-1}
    littéral-1
    expression-1
    TRUE
    FALSE
  }
  ALSO {
    {identificateur-2}
    littéral-2
    expression-2
    TRUE
    FALSE
  }
  ...
}

WHEN {
  ANY
  condition-1
  TRUE
  FALSE
  [NOT] {
    {identificateur-3}
    littéral-3
    expression-3
  }
  {
    THROUGH {
      {identificateur-4}
      littéral-4
      expression-4
    }
    THRU
  }
}

ALSO {
  ANY
  condition-2
  TRUE
  FALSE
  [NOT] {
    {identificateur-5}
    littéral-5
    expression-5
  }
  {
    THROUGH {
      {identificateur-6}
      littéral-6
      expression-6
    }
    THRU
  }
}

```

phrase-impérative-1 ...


[WHEN OTHER phrase-impérative-2] [END - EVALUATE].

Les opérandes ou les mots `TRUE` ou `FALSE` précédant le premier `WHEN` sont appelés les sujets de la sélection, alors que les opérandes suivant `WHEN` sont appelés les objets de la sélection. Le nombre de sujets doit être égal au nombre d'objets, la correspondance s'opérant par position relative.

L'instruction s'exécute en évaluant les conditions, expressions ou valeurs logiques des sujets et des objets. Chaque objet de la sélection suivant le premier `WHEN` est comparé au sujet de même position relative. Si la comparaison est satisfaite pour chacun des couples sujet-objet (`ANY` satisfait toute condition), alors l'instruction impérative qui suit ce `WHEN` est exécutée et l'on sort ensuite de l'instruction `EVALUATE`.


Le processus est répété en cas d'inégalité pour trouver le premier WHEN satisfaisant l'ensemble des conditions. Si aucune phrase suivant WHEN n'a été sélectionnée, et si WHEN OTHER a été spécifié, la phrase-impérative-2 est exécutée, et l'instruction EVALUATE terminée.

Il est possible de préciser CONTINUE en tant que phrase-impérative-1 ; cela indique tout simplement qu'aucune action n'est à effectuer et que l'on passe à l'instruction suivante.

 Exemple 2-13


```
EVALUATE A
  WHEN 1    PERFORM TRAITEMENT-1
  WHEN 2    PERFORM TRAITEMENT-2
  WHEN 5    PERFORM CONTINUE
  WHEN OTHER PERFORM TRAITEMENT-3
END-EVALUATE.
```

Le paragraphe TRAITEMENT-1 est exécuté si A a la valeur 1, TRAITEMENT-2 si A a la valeur 2, et TRAITEMENT-3 si A a toute valeur autre que 1, 2 et 5.

 Exemple 2-14

```
EVALUATE A ALSO B ALSO C
  WHEN 1 ALSO 5 ALSO NOT 7    PERFORM TRAITEMENT-1
  WHEN 2 ALSO 4 THRU 7 ALSO 3 PERFORM TRAITEMENT-2
  WHEN 5 ALSO ANY ALSO ANY    PERFORM TRAITEMENT-2
  WHEN OTHER                  PERFORM TRAITEMENT-3
END-EVALUATE.
```


Les instructions de TRAITEMENT-1 sont exécutées pour A=1, B=5 et C différent de 7, celles de TRAITEMENT-2 pour A=2, B compris entre 4 et 7, et C=3 ainsi que pour A=5 quelles que soient les valeurs de B et C, et enfin celles de TRAITEMENT-3 sont exécutées dans les autres cas.

 Exemple 2-15

EVALUATE A = 1 ALSO B = 1		
WHEN TRUE ALSO TRUE	PERFORM TRAITEMENT-1	
WHEN TRUE ALSO FALSE	PERFORM TRAITEMENT-2	
WHEN FALSE ALSO TRUE	PERFORM TRAITEMENT-2	
WHEN OTHER	PERFORM TRAITEMENT-3	
END-EVALUATE.		

A	B
1	1
1	0
0	1
0	0

On a ici une illustration de la possibilité de programmer, directement dans une instruction EVALUATE, tous les cas d'une table de vérité.

 Exemple 2-16

```
EVALUATE TRUE ALSO TRUE
  WHEN A = 1 ALSO B = 1 PERFORM TRAITEMENT-1
  WHEN ANY ALSO B = 2   PERFORM TRAITEMENT-2
  WHEN OTHER            PERFORM TRAITEMENT-3
END-EVALUATE.
```

Les instructions de TRAITEMENT-1 sont exécutées pour A=1 et B=1, celles de TRAITEMENT-2 si B=2 quel que soit A, et enfin celles de TRAITEMENT-3 sont exécutées dans les autres cas.

IF

$$\text{IF condition THEN } \left\{ \begin{array}{l} \text{instruction-1 ...} \\ \text{NEXTSENTENCE} \end{array} \right\} \left[\begin{array}{l} \bullet \\ \text{ELSE } \left\{ \begin{array}{l} \text{instruction-2 ...} \\ \text{NEXT SENTENCE} \end{array} \right\} \\ \text{END - IF} \end{array} \right]$$

Condition de comparaison :

$$\left\{ \begin{array}{l} \text{identificateur-1} \\ \text{littéral-1} \\ \text{expression-arithmétique-1} \\ \text{nom-d'index-1} \end{array} \right\} \text{ IS } [\text{NOT}] \left\{ \begin{array}{l} \text{GREATERTHAN} \\ \text{LESSTHAN} \\ \text{EQUALTO} \\ > \\ < \\ = \end{array} \right\} \left\{ \begin{array}{l} \text{identificateur-2} \\ \text{littéral-2} \\ \text{expression-arithmétique-2} \\ \text{nom-d'index-2} \end{array} \right\}$$

Condition de classe :

$$\text{identificateur IS } [\text{NOT}] \left\{ \begin{array}{l} \text{NUMERIC} \\ \text{ALPHABETIC} \end{array} \right\}$$

Condition de signe :

$$\text{expression-arithmétique IS } [\text{NOT}] \left\{ \begin{array}{l} \text{POSITIVE} \\ \text{NEGATIVE} \\ \text{ZERO} \end{array} \right\}$$

Condition de nom de condition / de position d' inverseur :

nom-de-condition

Condition de simple négative :

[NOT] conditionsimple

Condition composée :

condition $\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$ condition ... (possibilité d' utiliser les parenthèses)

Condition de comparaison abrégée :

condition de comparaison $\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\} [\text{NOT}]$ opérateur de comparaison
objet ...

- Si la condition est remplie, instruction-1 sera exécutée, sinon instruction-2 si la clause ELSE est précisée.
- 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 (COBOL 85) 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.
- La comparaison d'opérandes non numériques se fait de gauche à droite sur le premier caractère différent et selon l'ordre lexicographique (BLANK < 0 < 1 < ... < 9 < A < B < ... < Z).
- Les champs numériques ne doivent pas être testés sur ALPHABETIC et les champs alphabétiques ne doivent pas être testés sur NUMERIC.
- 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) ¹
1	1	1	1	0	0
1	0	0	1	1	1
0	1	0	1	1	1
0	0	0	0	1	0

Exemple 2-17

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)

Exemple 2-18

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

¹ Une manière de réaliser l'opération XOR.

Instructions de saut

GO TO

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

```
GO TO [nom-de-paragraphe-1].
```

- Le programme poursuit son exécution à l'endroit précisé après GO TO.
- L'utilisation du GO TO est déconseillée (sauf pour les traitements d'exception, mais ceux-ci sont agencés par les déclaratives) ; préférez-lui le PERFORM. L'utilisation conjointe de GO TO et de PERFORM est à proscrire absolument (du moins pour les débutants).

GO TO DEPENDING ON

Extension de la commande précédente, elle peut être considérée comme un aiguillage. La valeur de l'identificateur 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.

```
GO TO nom-de-paragraphe-1 [nom-de-paragraphe-2]... nom-de-paragraphe-n  
  DEPENDING ON identificateur.
```

- L'identificateur doit être défini en tant qu'entier numérique.
- Les paragraphes doivent tous exister dans le programme.



Exemple 2-19

```
...  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
77 choix PICTURE 9 VALUE 9.  
  
PROCEDURE DIVISION.  
PRINCIPAL SECTION.  
Boucle.  
    PERFORM Menu UNTIL choix = ZERO.  
    DISPLAY "Bye."  
    STOP RUN.  
  
MENU SECTION.  
Affiche.  
    DISPLAY "Entrées en stock.....(1)".  
    DISPLAY "Sorties de stock.....(2)".  
    DISPLAY "Fin.....(0)".  
  
    ACCEPT choix.  
    GO TO Entree Sortie DEPENDING ON choix.  
    GO TO Fin-Menu.
```

```
Entree.  
    PERFORM Traitement-Entree.  
    GO TO Fin-Menu.  
  
Sortie.  
    PERFORM Traitement-Sortie.  
    GO TO Fin-Menu.  
  
Fin-Menu.  
    EXIT.  
  
TRAITEMENT SECTION.  
Traitement-Entree.  
    DISPLAY "TRAITEMENT ENTREE".  
  
Traitement-Sortie.  
    DISPLAY "TRAITEMENT SORTIE".
```

Ici, on verrouille les GO TO à une section particulière.

Branchements modifiés - ALTER

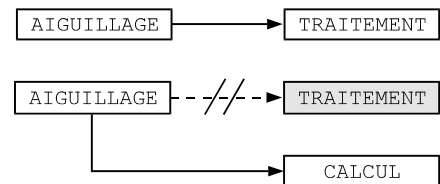
ALTER est une instruction de branchement qui permet d'orienter le traitement vers un paragraphe ou un autre suivant le positionnement de l'aiguillage, comme le ferait un aiguillage de chemin de fer.

```
ALTER nom-de-procédure-1 TO [PROCEED TO] nom-de-procédure-2
      [nom-de-procédure-3 TO [PROCEED TO] nom-de-procédure-4]...
```

L'aiguillage sera composé d'un nom de paragraphe et d'une instruction GO TO. Ensuite une instruction ALTER permet de le modifier.

Exemple 2-20

```
AIGUILLAGE. GO TO TRAITEMENT.
```



```
ALTER AIGUILLAGE TO PROCEED TO CALCUL.
```

Il existe encore un format particulier de GO TO qui découle directement des aiguillages, et dans notre exemple ci-dessus on aurait pu écrire :

```
AIGUILLAGE. GO TO.
```

```
AIGUILLAGE
```

```
ALTER AIGUILLAGE TO PROCEED TO TRAITEMENT.
```

```
AIGUILLAGE --> TRAITEMENT
```

```
ALTER AIGUILLAGE TO PROCEED TO CALCUL.
```

```
AIGUILLAGE --> CALCUL
```

Remarque :

L'instruction ALTER est un héritage du matériel mécanographique des années 1950-1960 qui était très limité en logique de programmation. Il est reconnu, depuis de nombreuses années, que les aiguillages sont très dangereux car, en cas d'incident, on ne sait jamais quel est leur positionnement exact à un moment donné. C'est pourquoi les experts ont décidé la suppression de la clause ALTER dans le prochain standard ANS.

STOP

Cette instruction marque la fin du traitement dans un programme et sert à terminer le programme puis à rendre le contrôle des opérations au superviseur du système d'exploitation.

STOP RUN termine le programme.

STOP littéral provoque une halte du programme et l'affichage d'un message.

Appel de procédures

PERFORM

En COBOL, une procédure est identifiée par le paragraphe (ou par 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 poursuit son déroulement juste après l'instruction PERFORM (à l'inverse de GO TO qui ne tient pas compte de cette adresse de retour).

PERFORM $\left[\text{nom-de-procédure-1} \left\{ \begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{nom-de-procédure-2} \right]$

[phrase-impérative END - PERFORM].

- Il ne faut utiliser que des paragraphes ou que des sections, et ne pas mélanger paragraphes et sections dans l'appel de procédures.
- 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

La procédure appelée est exécutée autant de fois que l'indique la valeur précisée avant TIMES.

PERFORM $\left[\text{procédure-1} \left\{ \begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{procédure-2} \right] \left\{ \begin{array}{c} \text{identificateur-1} \\ \text{entier-1} \end{array} \right\} \text{TIMES}$

[phrase-impérative END - PERFORM].

- La zone de données indiquée doit être de format numérique entier et ne pas dépasser 32767.
- 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 2-21

PUISSANCE.

```
...  
MOVE 1 TO résultat.  
PERFORM MULTIPLICATION N TIMES.  
DISPLAY A "puissance " N " est égal à " résultat.  
STOP RUN.
```

MULTIPLICATION.

```
MULTIPLY A BY résultat.
```

PERFORM UNTIL

```

PERFORM [ nom-de-procédure-1 [ { THROUGH } nom-de-procédure-2 ] ]
      [ WITH TEST { BEFORE } ] UNTIL condition-1
      [ phrase-impérative END - PERFORM ].
    
```

- Si l'on n'indique pas l'option `TEST`, ou si l'on précise `WITH TEST BEFORE`, le programme évaluera d'abord la condition avant de passer dans la boucle.
- Si on écrit `WITH TEST AFTER`, on exécute d'abord les instructions, puis on évalue la condition.
- 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.
- Le COBOL 85 ajoute l'option *inline* permettant de préciser directement dans l'instruction la ou les phrases impératives à exécuter sous le contrôle de l'instruction `PERFORM`. Ceci est aussi valable pour les formats précédents. On gagne ainsi en clarté et en lisibilité.

Exemple 2-22

On désire calculer une racine carrée avec une précision de 0,01 suivant l'algorithme de calcul itératif :

$$x = \frac{1}{2} \left(x_0 + \frac{A}{x_0} \right)$$

```

...
PERFORM UNTIL ECART < 0.01
  MOVE X TO X0
  COMPUTE X=(X0 + (A / X0)) / 2
  SUBTRACT X0 FROM X GIVING ECART
END-PERFORM.
    
```

PERFORM VARYING

`PERFORM VARYING` permet l'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. Identificateur-1, 4 et 7 sont les variables compteur dont les valeurs de départ sont respectivement identificateur-2, 5 et 8. Ces variables sont incrémentées respectivement de identificateur-3, 6 et 9 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.

PERFORM nom-de-procédure-1 $\left[\begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{nom-de-procédure-2} \right]$

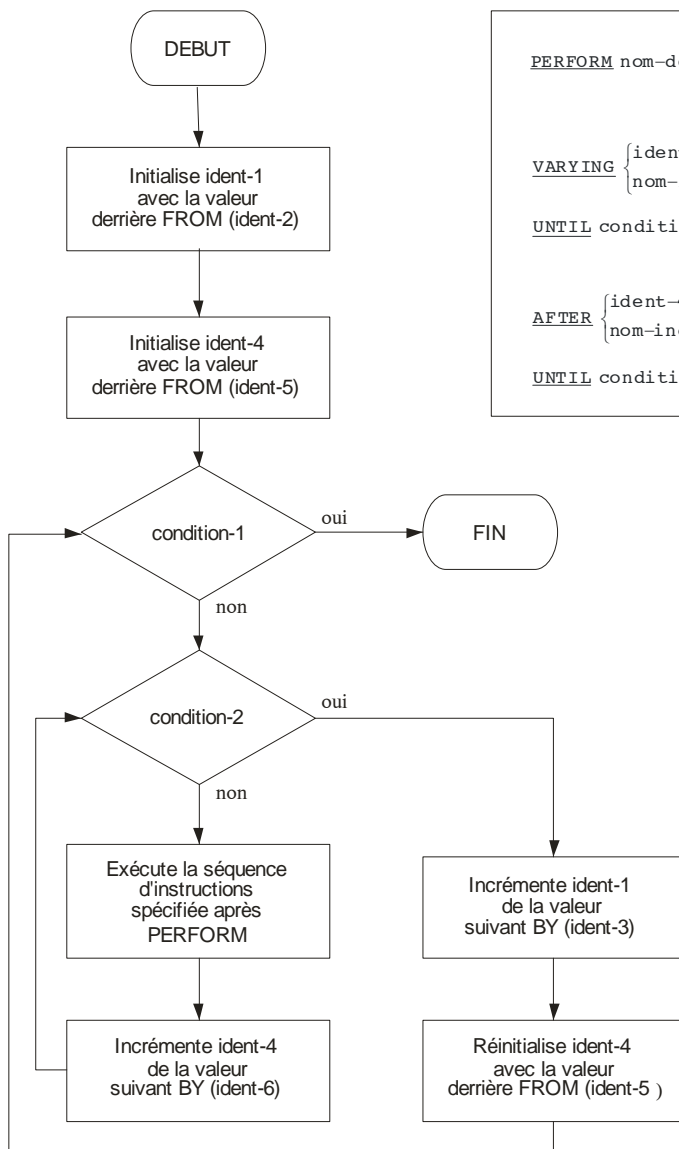
VARYING $\left\{ \begin{array}{c} \text{identificateur-1} \\ \text{nom-d' index-1} \end{array} \right\} \text{FROM} \left\{ \begin{array}{c} \text{identificateur-2} \\ \text{nom-d' index-2} \\ \text{littéral-1} \end{array} \right\} \text{BY} \left\{ \begin{array}{c} \text{identificateur-3} \\ \text{littéral-2} \end{array} \right\}$

UNTIL condition-1

$\left[\begin{array}{c} \text{AFTER} \left\{ \begin{array}{c} \text{identificateur-4} \\ \text{nom-d' index-3} \end{array} \right\} \text{FROM} \left\{ \begin{array}{c} \text{identificateur-5} \\ \text{nom-d' index-4} \\ \text{littéral-3} \end{array} \right\} \text{BY} \left\{ \begin{array}{c} \text{identificateur-6} \\ \text{littéral-4} \end{array} \right\} \\ \text{UNTIL condition-2} \end{array} \right]$

...

[phrase-impérative END - PERFORM].



PERFORM nom-de-procédure-1 $\left[\begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right\} \text{nom-de-procédure-2} \right]$

VARYING $\left\{ \begin{array}{c} \text{ident-1} \\ \text{nom-index-1} \end{array} \right\} \text{FROM} \left\{ \begin{array}{c} \text{ident-2} \\ \text{nom-index-2} \\ \text{littéral-1} \end{array} \right\} \text{BY} \left\{ \begin{array}{c} \text{ident-3} \\ \text{littéral-2} \end{array} \right\}$

UNTIL condition-1

AFTER $\left\{ \begin{array}{c} \text{ident-4} \\ \text{nom-index-3} \end{array} \right\} \text{FROM} \left\{ \begin{array}{c} \text{ident-5} \\ \text{nom-index-4} \\ \text{littéral-3} \end{array} \right\} \text{BY} \left\{ \begin{array}{c} \text{ident-6} \\ \text{littéral-4} \end{array} \right\}$

UNTIL condition-2.

PERFORM VARYING
avec deux conditions

Ci-contre sont représentés le format général de l'instruction `PERFORM` avec l'option `VARYING`, ainsi que l'organigramme relatif à l'utilisation de deux conditions et la clause implicite `TEST BEFORE`. Remarquez qu'on peut écrire jusqu'à six `AFTER` imbriqués pour initialiser une table à sept niveaux.

- 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.

EXIT

Permet de définir un paragraphe vide à un endroit donné du programme.

En Cobol l'adresse de retour du `PERFORM` est stockée sur la dernière instruction de la procédure. Comme cette adresse n'est pas visible, une technique consiste à réserver systématiquement un paragraphe que l'on nommera `FIN-Procédure` et qui contiendra uniquement l'instruction `EXIT`. Aussi, si l'on veut arrêter cette procédure (dans le cas d'une erreur, par exemple), on fera un `GO TO FIN-Procédure`. Si la procédure est appelée par un `PERFORM`, l'instruction `EXIT` garantit le retour normal vers l'instruction suivant le `PERFORM`.

- 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 2-23

```
...  
    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 NOT = "00"  
        PERFORM Affiche-Message-Erreur  
        GO TO Fin-Recherche.  
  
Affiche-Client.  
...  
Fin-Recherche.  
    EXIT.
```

Appel de modules externes

CALL

Appel de module. Le module est un autre fichier programme compilé séparément.

```
CALL {identificateur-1  
      littéral-1} USING {BY REFERENCE identificateur-2 ...}  
                        {BY CONTENT identificateur-2 ...} ...
```

```
[ON OVERFLOW phrase-impérative]  
[END - CALL].
```

```
CALL {identificateur-1} [ USING {BY REFERENCE identificateur-2 ...}  
      littéral-1          {BY CONTENT identificateur-2 ...} ... ]
```

```
[ON EXCEPTION phrase-impérative-1]  
[NOT ON EXCEPTION phrase-impérative-2]  
[END - CALL].
```

Le programme appelant

- L'option par défaut BY REFERENCE transmet les données de manière à ce qu'elles puissent être modifiées par le sous-programme.
- L'option BY CONTENT signifie que le sous-programme reçoit la valeur – en fait une copie de l'original – des données suivant CONTENT. Contrairement à BY REFERENCE, d'éventuelles modifications sur cette valeur ne seront pas répercutées dans le programme appelant.
- Après l'instruction CALL il faut toujours écrire toutes les données partagées par les deux programmes, 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.
- Le sous programme est initialisé lors du premier appel par l'instruction CALL. Lors des éventuels appels suivants, le sous-programme reste dans l'état dans lequel il se trouvait lorsqu'il avait été quitté pour la dernière fois : en particulier, le contenu des zones et le positionnement des fichiers y restent fixés, sauf si une instruction CANCEL a été exécutée entre-temps dans le programme principal.
- Les options ON OVERFLOW et ON EXCEPTION ont sensiblement le même usage, si ce n'est que ON OVERFLOW agit dans le cas où il n'y a pas suffisamment de place en mémoire pour charger le sous-programme en mémoire, alors que ON EXCEPTION est destiné à tout type d'anomalies détectées dans l'exploitation du sous-programme.

Le sous-programme

- Le module appelé doit définir dans la `LINKAGE SECTION` **toutes** les données partagées par le programme principal et le sous-programme. Les données ainsi déclarées doivent correspondre à leurs homologues dans le module appelant, où elles sont définies dans la `WORKING-STORAGE SECTION`². 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.
- Toutes les clauses de description de données sont admises dans la `LINKAGE SECTION`, à l'exception de la clause `VALUE`.
- Dans le module appelé, la `PROCEDURE DIVISION` doit être suivie de la clause :

`USING nom-de-donnée-1 [nom-de-donnée-2] . . .`

- La `PROCEDURE DIVISION` doit comporter l'instruction `GOBACK` ou `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.

Remarque :

Les adresses de zones de données étant communiquées sous forme d'une table d'adresses³, il sera possible d'avoir moins de zones dans la clause `USING` du sous-programme que dans le programme principal. Par contre, il ne sera pas possible d'en avoir plus car le sous-programme prendrait n'importe quelle donnée au-delà de la table comme adresse de zone, ce qui peut provoquer des résultats assez imprévisibles.

Exemple 2-24

L'exemple suivant donne un aperçu de ce qu'il est possible de réaliser avec l'instruction `CALL`. Si l'on regarde exclusivement le programme principal, on ne sait pas comment sont gérées la lecture et l'écriture. Ces opérations sont complètement transparentes pour le programme appelant. Avec des fichiers réels, la programmation modulaire apporte ainsi plus de clarté et facilite la maintenance.

² En fait, le sous programme ne réserve pas de mémoire pour les zones définies dans la `LINKAGE SECTION`.

³ Avis aux cracks en assembleur : cette table est transmise par le registre général 1 de l'ordinateur.

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.
```

CANCEL

Cette instruction libère la mémoire occupée par un ou plusieurs sous-programmes.

CANCEL { $\left\{ \begin{array}{l} \text{identificateur-1} \\ \text{littéral-1} \end{array} \right\}$ } [$\left[\begin{array}{l} \text{identificateur-2} \\ \text{littéral-2} \end{array} \right\}$]] ...

- Si après l'exécution de cette instruction, on appelle à nouveau par un CALL un des sous-programmes, celui-ci est rechargé dans son état initial. Toutes les données reprennent leur VALUE d'origine et les pointeurs de fichier sont réinitialisés.

ENTRY

Si l'entrée du sous-programme n'est pas le premier ordre de la PROCEDURE DIVISION, le point d'entrée dans le sous-programme est désigné par une clause ENTRY dont le format est le suivant :

```
ENTRY " Nom-symbolique-d'entrée" [USING identificateur-1 [identificateur-2] ...]
```

EXIT PROGRAM

Dans un sous-programme, la terminaison est réalisée par les clauses EXIT PROGRAM et GOBACK. Lorsque le traitement passe sur une telle clause, celui-ci est automatiquement rebranché juste après l'ordre CALL du programme principal appelant.

EXIT PROGRAM.

- 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.

EXTERNAL

L'attribut EXTERNAL signifie que ses enregistrements seront accessibles par tous les sous-programmes appelés par le programme principal, à condition de décrire ces enregistrements en WORKING-STORAGE SECTION des sous-programmes sous la même forme.

```
FD nom-de-fichier [EXTERNAL]
```

Description de fichier

GOBACK

Dans un sous-programme, la terminaison est réalisée par les clauses GOBACK et EXIT PROGRAM. Lorsque le traitement passe sur une telle clause, celui-ci est automatiquement rebranché juste après l'ordre CALL du programme principal appelant.

GOBACK.

- Si le programme n'a pas été appelé par un CALL, on a un effet identique à STOP RUN.

Programmes contenus

Structure des programmes

En COBOL 85, un programme source peut contenir d'autres programmes source pouvant accéder aux ressources du programme dans lequel ils sont contenus.

Exemple 2-25

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.  PROG-1.  
...  
PROCEDURE DIVISION.  
...  
IDENTIFICATION DIVISION.  
PROGRAM-ID.  PROG-2.  
...  
END PROGRAM PROG-2.  
...  
END PROGRAM PROG-1.
```

PROG-2 est contenu dans PROG-1

COMMON

Normalement, un programme contenu ne peut être appelé que par un programme qui le contient au niveau immédiatement supérieur.

L'attribut **COMMON** affecté au nom d'un programme contenu permet, au contraire, d'appeler ce programme depuis n'importe quel autre programme de l'entité compilée.

PROGRAM - ID. nom-du-programme [COMMON].

INITIAL

Jusqu'à présent, les données déclarées en COBOL ont toujours été de type *statique*, c'est-à-dire que l'état initial est affecté une seule fois à la compilation par la clause **VALUE**.

L'attribut **INITIAL** de la norme A.N.S. 85 affecté à un programme signifie que les données de ce programme seront de type *automatique*. Les valeurs initiales sont affectées à chaque appel du programme et les fichiers sont remis à l'état initial.

PROGRAM - ID. nom-du-programme [INITIAL].

GLOBAL

Un nom de donnée affecté de l'attribut **GLOBAL** est connu du programme dans lequel il est déclaré ainsi que de tous les programmes contenus dans ce programme, à condition que ce nom de donnée ne soit pas déclaré une seconde fois dans un programme contenu.

Les tableaux

Partie I - Introduction	59
Partie II - Définition de tableaux - OCCURS	59
Partie III - Accès aux éléments d'un tableau	61
Les indices	61
Définition des index	62
Traitement des index (SET)	63
Index relatifs	63
Recherches dans un tableau (SEARCH)	64
Partie IV - Différences entre indices et index	65
Partie V - Exemple : Tri bulle (bubble sort)	66

Les tableaux



Introduction

Le COBOL-85 permet de définir des tables ayant jusqu'à 7 niveaux, chaque élément pouvant être un champ ou un groupe de données. Pour implanter un tel tableau en mémoire nous avons besoin de connaître le nombre de dimensions et d'éléments qu'il comporte. Pour faire référence à un élément, il faut lui donner la position dans le tableau pour chaque dimension. Dans un tableau à trois dimensions on aura donc trois indicateurs de position. Nous verrons qu'il existe deux sortes d'indicateurs de position : les indices et les index.

Les tableaux sont particulièrement indiqués pour le traitement rapide d'un nombre peu important de données. Pour gérer des quantités massives de données on utilisera plutôt les fichiers.

Définition de tableaux

OCCURS

La clause OCCURS s'utilise dans la description des données. Une zone de données est répétée autant de fois que le précise entier-2 ou un nombre de fois compris entre entier-1 et entier-2 selon la valeur de nom-de-donnée-1.

La clause KEY donne la clé nom-de-donnée-4 et indique la méthode de tri pour la recherche rapide (SEARCH ALL), ASCENDING : croissant, DESCENDING : décroissant.

Si le tableau doit être traité selon la méthode des index, il faut préciser la clause INDEXED BY.

$$\text{OCCURS} \left\{ \begin{array}{l} \text{entier-1 TO entier-2 TIMES DEPENDING ON nom-de-donnée-1} \\ \text{entier-2 TIMES} \end{array} \right\}$$
$$\left[\left\{ \begin{array}{l} \text{ASCENDING} \\ \text{DESCENDING} \end{array} \right\} \text{ KEY IS nom-de-donnée-2 [nom-de-donnée-3] ...} \right] \dots$$
$$[\text{INDEXED BY nom-d' index-1 [nom-d' index-2] ...}]$$

- L'option DEPENDING ON s'emploie lorsqu'on ne veut pas fixer tout de suite le nombre d'éléments contenus dans une table.
- entier-1 (taille minimale) doit être un entier positif inférieur à entier-2 (taille maximale). nom-de-donnée-1 doit définir un entier positif et représente la taille exacte du tableau.
- 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 la clause ASCENDING ou DESCENDING est précisée, il est convenu que le tableau soit trié sur nom-donnée-2, **mais attention** : c'est le programmeur qui est responsable du tri.
- Les noms d'index suivant INDEXED BY sont automatiquement définis en binaire par le système et ne doivent pas, par conséquent, être définis dans la WORKING-STORAGE SECTION.

Exemple 3-1

01 VECTEUR. 02 A PICTURE 9(5) OCCURS 5.	Définition d'un tableau de nom VECTEUR avec 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).	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).	Tableau à deux dimensions, pour 5 jours et 8 heures par jour.
01 TABLE. 05 libmois. 10 FILLER PIC X(9) VALUE "janvier". 10 FILLER PIC X(9) VALUE "février". ... 10 FILLER PIC X(9) VALUE "décembre". 05 libelle REDEFINES libmois. 10 nomois OCCURS 12 PIC X(9).	OCCURS et VALUE s'excluent mutuellement ; il est cependant possible d'initialiser un tableau à l'aide de la clause REDEFINES.

Accès aux éléments d'un tableau

Format général

$$\text{nom-de-donnée-1} \left(\begin{array}{l} \text{entier-1} \\ \text{nom-de-donnée-2} \left[\{ + \text{ ou } - \} \text{entier-2} \} \dots \right] \\ \text{nom-d'index} \left[\{ + \text{ ou } - \} \text{entier-2} \} \dots \right] \end{array} \right)$$

- Pour un même nom de donnée on peut utiliser à la fois un indice et un index.
- Un nom de donnée indicé ou indexé ne peut lui-même être utilisé comme indice ou index¹.

Les indices

Pour traiter un tableau avec la méthode des indices, la clause INDEXED BY ne doit 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. Le COBOL 85 permet également d'effectuer de l'indigage relatif.

- 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 et/ou des virgules².
- La valeur d'un indice doit être supérieure à zéro et ne doit pas excéder la valeur précisée après la clause OCCURS.

Exemple 3-2

```
DISPLAY A (1) A (2) A (3) A (4) A (5).
```

ou

```
DISPLAY VECTEUR.
```

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

```
DISPLAY lieu-residence (15).
```

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).
```

Ajout du cours "COBOL" dans l'emploi du temps à la cinquième heure du troisième jour

¹ L'écriture A (B (I)) est incorrecte.

² Les règles d'écriture étant assez étriquées, l'utilisation des virgules est déconseillée.

Définition des index

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.

- 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.
- On peut éventuellement retrancher ou ajouter un entier à un index (voir *index relatifs*).
- 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 peut être fixée ou modifiée uniquement par les instructions SET, SEARCH et PERFORM (les opérations directes sur les index ne sont pas autorisées).
- Les index ne doivent pas être redéfinis dans la WORKING-STORAGE SECTION car ils sont définis automatiquement par l'ajout de la clause INDEXED BY. Cependant, il est possible de définir des données en USAGE INDEX. Dans ce cas, les clauses BLANK WHEN ZERO, JUSTIFIED, PICTURE, VALUE, SYNCHRONIZED sont interdites.

Exemple 3-3

```
01 VECTEUR.  
  02 A PICTURE 9(5) OCCURS 5 INDEXED BY I.
```

Définition d'un tableau de nom
VECTEUR avec 5 positions.

```
01 ADRESSES.  
  02 CHAMP-ADRESSES OCCURS 200 INDEXED BY I.  
    03 nom PICTURE X(15).  
    03 prenom PICTURE X(15).  
    03 rue PICTURE X(20).  
    03 lieu-de-residence.  
    04 c-postal PICTURE 9(5).  
    04 ville PICTURE X(20).
```

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).
```

Tableau à deux dimensions, pour
5 jours et 8 heures par jour.

```
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).
```

Cette matrice à 3 dimensions
comporte $5 \times 20 \times 10 = 1000$
éléments \times 5 positions alpha-
numériques = 5000 caractères.

Traitement des index (SET)

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.

$$\underline{\text{SET}} \left\{ \begin{array}{l} \text{identificateur-1} [\text{identificateur-2}] \dots \\ \text{nom-d' index-1} [\text{nom-d' index-2}] \dots \end{array} \right\} \underline{\text{TO}} \left\{ \begin{array}{l} \text{identificateur-3} \\ \text{nom-d' index-3} \\ \text{entier-1} \end{array} \right\}.$$

$$\underline{\text{SET}} \text{ nom-d' index-4} [\text{nom-d' index-5}] \dots \left\{ \begin{array}{l} \underline{\text{UP}} \\ \underline{\text{DOWN}} \end{array} \right\} \underline{\text{BY}} \left\{ \begin{array}{l} \text{identificateur-4} \\ \text{entier-2} \end{array} \right\}.$$

- Tous les noms d'index doivent être définis dans une clause INDEXED BY.
- Si identificateur-1 n'est pas en USAGE INDEX, on ne peut utiliser ni identificateur-3, ni entier-1. identificateur-1 prendra pour valeur le rang correspondant à la valeur de nom-d'index-3.
- Le contenu de nom-d'index-4 (nom-d'index-5) est augmenté (UP) ou diminué (DOWN) de la valeur correspondant au rang indiqué par la valeur d'entier-2 ou d'identificateur-4.
- 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 3-4

On veut faire la somme des carrés $1^2 + 2^2 + \dots + I^2$ 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.
```

Recherches dans un tableau (SEARCH)

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 (SEARCH ALL) 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.

```

SEARCH [ALL] identificateur-1 [ VARYING { identificateur-2 }
                                { nom-d' index-1 } ]
    [ AT END phrase-impérative-1 ]

    WHEN condition-1 { instruction-impérative-2 }
                     { NEXT SENTENCE }
    [ WHEN condition-2 { instruction-impérative-3 } ] ...
    [ END - SEARCH ].
    
```

- L'identificateur 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 après la clause AT END est exécutée en cas de recherche infructueuse.
- La clause NEXT SENTENCE implique la poursuite normale du programme.

Exemple 3-5

```

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

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

Recherche linéaire

Recherche binaire
(dichotomique)

☑ 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 $\text{LOG}_2(N)$ 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.

Différences entre indices et index

- Le traitement des tableaux par la méthode des index est environ 40 % plus rapide que par les indices³.
- 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`.

☑ Remarque concernant l'utilisation de l'instruction `SET` :

Lorsque l'on écrit :

```
SET A TO B
```

c'est A qui va prendre la valeur de B ($A \leftarrow B$). Or on a parfois tendance à confondre ce processus avec l'affectation `MOVE A TO B` ($A \rightarrow B$). Voici un truc mnémotechnique pour ne plus se tromper :

Quelle est la variable qui va être modifiée ? `SET A ...`

³ Si toutefois ceux-ci ne sont pas déclarés en `USAGE COMPUTATIONAL`.

Exemple : Tri bulle (bubble sort)

Ce tri a l'avantage d'être simple et utilisable dans tous les contextes. Ici des nombres sont lus dans une table à partir d'un fichier. On compare ensuite les deux premiers éléments. Si le premier est supérieur au second, on les inverse. Ce traitement se poursuit avec les éléments suivants jusqu'à ce que l'on soit arrivé à la fin du tableau. Le premier élément du tableau est maintenant le plus petit. On procède de même avec le deuxième élément, et ainsi de suite...

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

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

WORKING-STORAGE SECTION.
77  val-temp  PICTURE 99 VALUE ZERO.

01  drapeau   PICTURE  9 VALUE ZERO.
    88 FIN    VALUE 1.

01  TABLEAU.
    02 element OCCURS 100 INDEXED BY i j haut PICTURE 99.

PROCEDURE DIVISION.
Principal.
    OPEN INPUT F-NOMBRES.
    READ F-NOMBRES AT END MOVE 1 TO drapeau.
    PERFORM Lit-Nombre VARYING i FROM 1 BY 1 UNTIL FIN OR i > 99.
    CLOSE F-NOMBRES.
    SET i DOWN BY 1.
    PERFORM Bubble-Sort VARYING haut FROM i BY -1 UNTIL haut < 2
                        AFTER j FROM 1 BY 1 UNTIL j = haut.
    PERFORM Affiche-Nombre VARYING j FROM 1 BY 1 UNTIL j > i.
    STOP RUN.

Lit-Nombre.
    MOVE un-nombre TO element (i).
    READ F-NOMBRES AT END MOVE 1 TO drapeau.

Bubble-Sort.
    IF element (j) > element (j + 1)
        MOVE element (j)          TO val-temp
        MOVE element (j + 1) TO element (j)
        MOVE val-temp             TO element (j + 1).

Affiche-Nombre.
    DISPLAY element (j) " " WITH NO ADVANCING.
```

Les fichiers

Partie I - Généralités	69
Notion de fichier	69
Les différentes formes d'organisation	69
Les différentes formes d'accès	69
Modifications	72
Dégénérescence de l'organisation par modifications	73
 Partie II - Description de fichiers.	74
SELECT	74
FILE DESCRIPTION (FD)	76
 Partie III - Instructions pour la gestion des fichiers	78
AT END	78
CLOSE	78
DELETE	79
INVALID KEY	79
OPEN	80
READ	80
REWRITE	81
START	81
WRITE	82
Tableau de synthèse	82
Exemple : Fichier mouvement	83
 Partie IV - Tri-fusion.	89
SORT	89
RELEASE	91
RETURN	91
MERGE	92

Les fichiers



Généralités

Notion de fichier

Un fichier est la représentation d'une structure de données abstraite. Un fichier séquentiel, par exemple, est une forme de représentation d'une liste chaînée. Un tel fichier regroupe une collection de valeurs de même type ; ces valeurs s'appellent des enregistrements logiques. L'intérêt des fichiers est qu'ils procurent des espaces de mémorisation quasi infinis et permanents (rémanence).

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.

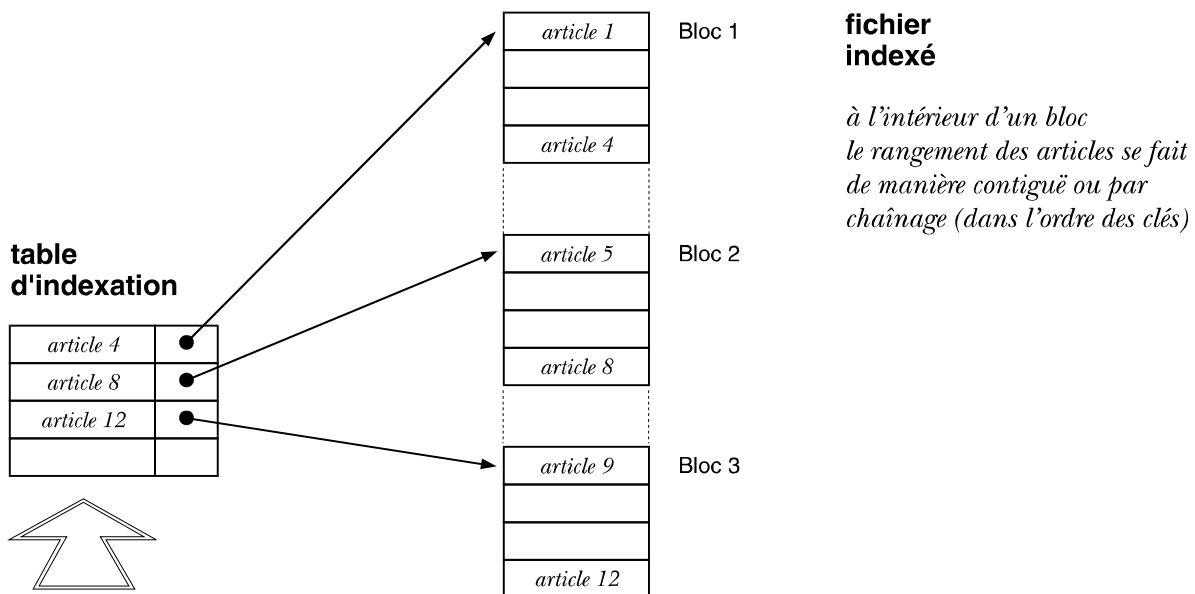
Directe, relative ou sélective Il n'y a pas de table d'indexation, mais une clé à travers laquelle on recherche l'élément désiré. Le rangement physique peut être partiellement consécutif mais on ne peut pas en tenir compte. Les éléments sont soit chaînés entre eux (la clé, si elle existe, servant à reconnaître l'élément cherché), soit rangés à des adresses connues que l'on sait calculer (hash code).

Les différentes formes d'accès

Accès séquentiel

L'accès séquentiel sera souvent utilisable, il est simple d'emploi en organisation séquentielle ou séquentielle indexée et plus complexe ailleurs. Partant d'une extrémité du fichier, 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 courant fournit automatiquement l'article suivant.

En séquentiel indexé, il faut balayer l'ensemble de la table d'index, puis tous les blocs, car les enregistrements ne sont pas nécessairement écrits *physiquement* dans l'ordre des clés, mais - plus logiquement - dans l'ordre de création (les enregistrements sont alors chaînés entre eux afin de maintenir *l'ordre* pour la clé). Mais rassurez-vous, en COBOL vous n'aurez pas à gérer de telles subtilités.



MOVE 6 TO clé.

READ fichier-indexé INVALID KEY DISPLAY "Article non trouvé !".

On accède au bloc où se trouve l'article grâce à la table d'index. Celle-ci contient la valeur de la clé la plus forte de chaque bloc. Ensuite, à l'intérieur d'un bloc, on recherche séquentiellement l'article (d'où le nom de *séquentiel indexé*).

☑ Exemple 4-1 : recherche de l'article 6

Table d'index :

article 6 > article 4	l'article 6 ne se trouve pas dans le bloc 1
article 6 < article 8	l'article 6 se trouve dans le bloc 2

Bloc 2 :

article 6 > article 5	recherche séquentielle
article 6 == article 6	article trouvé !

Accès direct

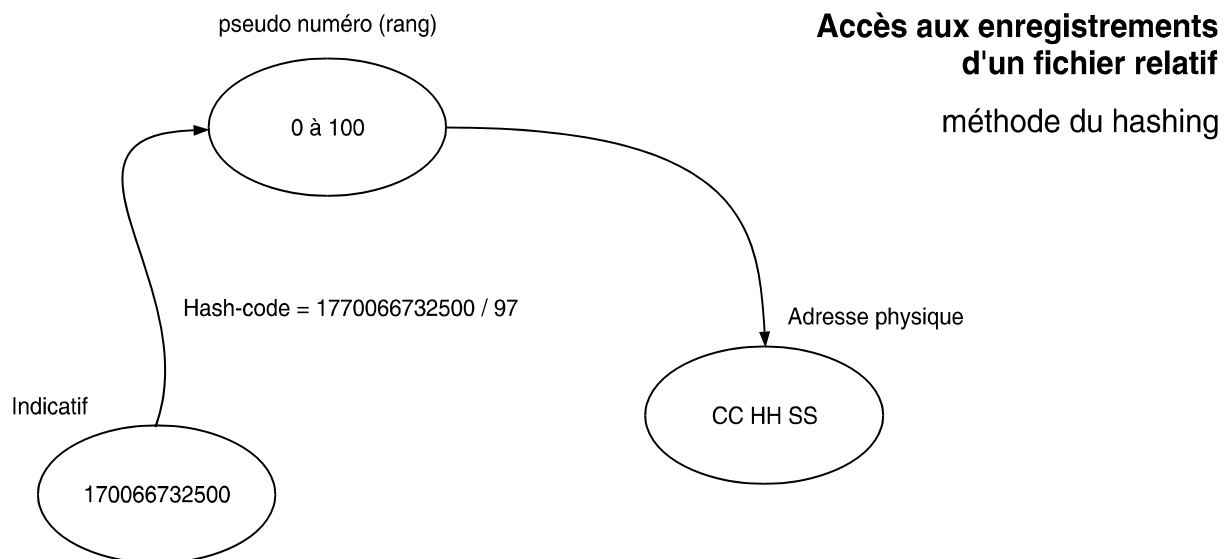
Comme son nom le laisse supposer, cet accès donne directement l'article cherché sans qu'il soit nécessaire de tenir compte du reste du fichier. Mais si le terme employé est simple, il recouvre en fait de nombreuses méthodes.

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). Une autre façon de faire serait d'utiliser une méthode dichotomique. Le fichier doit être trié, et pour ne pas perdre de temps, il faut pouvoir travailler sur des tables d'index en mémoire. L'avantage de l'accès dichotomique est de ramener le nombre moyen de consultations pour trouver un élément à $\log_2(n)$.

En résumé, le mode d'accès direct peut être soit prédéfini, 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 4-2

On veut stocker la liste des employés d'une entreprise qui sont identifiés par leur numéro INSEE. Il n'est cependant pas possible de réserver un enregistrement physique pour tous les numéros INSEE existants. On décide donc de réserver 100 enregistrements physiques identifiés par un pseudo numéro. Un hash-code très simple se calcule de la manière suivante : On divise le numéro INSEE par le nombre premier immédiatement inférieur à la dimension de l'espace total (dans notre cas ce nombre est 97).



Comme il existe plus de numéros INSEE que d'emplacements physiques, il se peut qu'à deux numéros INSEE corresponde un seul et même pseudo-numéro. On incrémente alors le rang de l'élément que l'on veut ajouter jusqu'à trouver un pseudo-numéro non utilisé (donc un emplacement libre). En fin de fichier on repart à 1, dès lors la nouvelle clé se calcule : *(ancienne clé modulo la taille du fichier) + 1*.

Si par la suite on revient à la position initiale, cela signifie qu'il n'y a plus de place dans l'organisation du fichier. Dans ce cas, il y a deux solutions :

La première consiste à gérer un *tas* en fin de fichier (pour ce faire, on utilisera les pseudo-numéros au-dessus de 100). Etant donné qu'il faudra parcourir l'ensemble du fichier pour trouver cet article, nous conseillons de chaîner cet article avec l'article précédent (pseudo-numéro immédiatement inférieur).

La seconde - plus recommandable - consiste à restructurer entièrement le fichier en lui accordant cette fois 200 emplacements au lieu de 100.

Accès dynamique

L'accès dynamique est la combinaison des deux précédents. Le COBOL permet à la fois l'accès séquentiel et l'accès indexé sur des fichiers d'organisation relative ou indexée, et cela sans réouverture du fichier. Il s'agit là d'une méthode très souple et très pratique pour la recherche d'un groupe de données triées. On fait un premier accès direct par la clé, puis on lit les enregistrements suivants en séquentiel.

Modifications

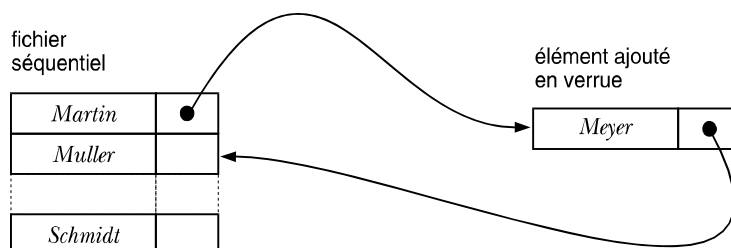
Le COBOL offre des fonctions très puissantes permettant de réaliser toutes sortes de modifications sur des fichiers. Toutefois, il est fondamental de connaître les mécanismes sous-jacents utilisés par le système.

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. Dans le dernier cas, on utilisera la technique des *fichiers mouvement* où les modifications sont triées dans le même ordre que les articles du fichier maître ; cela permet de faire toutes les opérations lors d'une seule recopie¹.

Sur un fichier aléatoire, il suffit de trouver un espace libre et de mettre à jour le moyen de le retrouver. Cela peut impliquer l'adjonction d'un élément rangé dans une table d'index ou la mise à jour d'un pointeur (organisation chaînée).

Exemple 4-3



Suppression

Opération symétrique de la précédente, elle est néanmoins plus facile à réaliser car elle ne nécessite pas la recherche d'espace libre. Elle consiste généralement à indiquer dans un octet prévu à cet effet que l'article associé n'existe plus. Dans le cas d'une représentation chaînée, on fait 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).

Mise à jour

Elle nécessite une éventuelle modification de la longueur de la zone occupée par l'article. Si on ne dispose pas de l'espace nécessaire, il faut ajouter le nouvel élément modifié puis supprimer l'ancien.

¹ cf. exemple page 83

Notions avancées

Confidentialité des données.

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, il faut écrire par-dessus les données avant de les « détruire ».

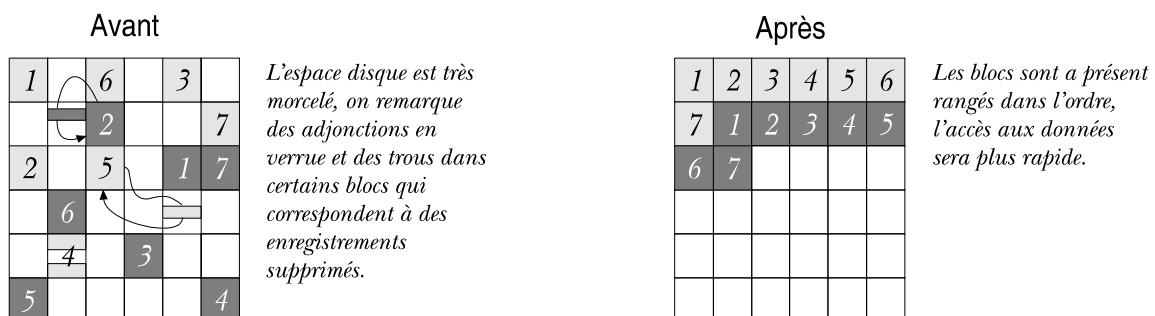
Dégénérescence de l'organisation par modifications

Nous venons de voir qu'à la suite d'adjonctions, le programme peut être amené à transformer progressivement la composition d'un fichier, de même que les suppressions le transforment peu à peu 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 débordement² (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.

Exemple 4-4



² cf. page 71 (*gestion d'un tas*)

Description de fichiers

SELECT

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.

INPUT – OUTPUT SECTION.

FILE – CONTROL.

Fichiers séquentiels

SELECT [OPTIONAL] nom-de-fichier ASSIGN TO nom-de-fichier-externe-1
[nom-de-fichier-externe-2]...

[RESERVE entier-1 [AREA]
[AREAS]]

[ORGANIZATION IS SEQUENTIAL] [ACCESS MODE IS SEQUENTIAL]

[PADDING CHARACTER IS {nom-de-donnée-1}
littéral-1}]

[RECORD DELIMITER IS {STANDARD-1
nom-de-fichier-externe-2}]

[FILE STATUS IS nom-de-donnée-3].

Fichiers indexés

SELECT [OPTIONAL] nom-de-fichier ASSIGN TO nom-de-fichier-externe-1
[nom-de-fichier-externe-2]...

[RESERVE entier-1 [AREA]
[AREAS]]

ORGANIZATION IS INDEXED [ACCESS MODE IS {SEQUENTIAL
RANDOM
DYNAMIC}]

RECORD KEY IS nom-de-donnée-1

[ALTERNATE RECORD KEY IS nom-de-donnée-2 [WITH DUPLICATES]]...

[FILE STATUS IS nom-de-donnée-3].

Fichiers relatifs

SELECT [OPTIONAL] nom-de-fichier ASSIGN TO [nom-de-fichier-externe-1] ...

[RESERVE entier-1 [AREA
AREAS]]

ORGANIZATION IS RELATIVE

[ACCESS MODE IS {SEQUENTIAL [RELATIVE KEY IS nom-de-donnée-1]
{RANDOM
DYNAMIC} RELATIVE KEY IS nom-de-donnée-1 }]
[FILE STATUS IS nom-de-donnée-3].

- Les paramètres par défaut sont : organisation séquentielle, accès séquentiel.
- 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).
- Le mot clé OPTIONAL indique qu'un fichier d'entrée n'est pas forcément présent. Dans ce cas, les instructions impératives suivant AT END sont exécutées lors de la première instruction READ.
- La clause RESERVE indique le nombre de zones tampon d'entrée-sortie affectées au fichier.
- La clause PADDING permet, en COBOL 85, de définir un caractère de remplissage pour compléter les blocs non intégralement remplis.
- La clause RECORD DELIMITER précise la façon dont la longueur des enregistrements variables est déterminée.
- 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.
- Pour un fichier indexé ou relatif, il faut préciser la clause RELATIVE KEY (fichier relatif) ou RECORD KEY (fichier indexé) ; nom-de-donnée-1 est de format numérique entier non signé.
- ALTERNATE KEY : avec les fichiers indexés, on a la possibilité d'avoir une clé secondaire³. La clause WITH DUPLICATES indique que la clé est multiple⁴.
- Le nom-de-donnée-3 doit être défini dans la WORKING-STORAGE SECTION au format XX. Ce champ est très important puisqu'il contiendra un code retour après chaque accès au fichier.

Exemple 4-5

SELECT FICHIER-SEQ ASSIGN TO "fich1"	FD FICHIER-SEQ.
ORGANIZATION IS SEQUENTIAL	01 CHAMP-SEQ.
ACCESS IS SEQUENTIAL	02 NOM PIC X(10).
FILE STATUS IS fichier-seq-st.	02 PRENOM PIC X(15).
	02 NUM-TEL PIC X(12).

³ Par exemple, un fichier de personnel ordonné suivant les matricules croissants comme clé principale peut aussi être classé suivant l'ordre alphabétique des noms de personnes comme clé secondaire.

⁴ S'il existe plusieurs personnes de même nom.

Exemple 4-6

```

SELECT FICHER-IDX ASSIGN TO "fich3"      FD  FICHER-IDX.
ORGANIZATION IS INDEXED                  01  CHAMP-IDX.
ACCESS IS RANDOM                         02  TITRE      PIC X(20).
RECORD KEY IS TITRE                     02  DATE1      PIC X(8).
FILE STATUS IS fichier-idx-st.           02  VALEUR     PIC S9(6)V99.

SELECT FICHER-REL ASSIGN TO "fich2"      FD  FICHER-REL.
ORGANIZATION IS RELATIVE                 01  CHAMP-REL.
ACCESS MODE IS RANDOM                   02  CLIENT     PIC X(30).
RELATIVE KEY IS JOUR                    02  MENU       PIC X(80).
FILE STATUS IS fichier-rel-st.           02  PRIX      PIC 9(6)V99.

```

Attention : JOUR sera à définir dans la WORKING-STORAGE SECTION, et non dans la rubrique.

FILE DESCRIPTION (FD)

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.

DATA DIVISION .

FILE SECTION .

FD nom-de-fichier

$$\begin{aligned}
 & \left[\text{BLOCK CONTAINS [entier-1 TO] entier-2 } \left\{ \begin{array}{l} \text{RECORDS} \\ \text{CHARACTERS} \end{array} \right\} \right] \\
 & \left[\text{RECORD CONTAINS [entier-3 TO] entier-4 CHARACTERS} \right] \\
 & \left[\text{LABEL } \left\{ \begin{array}{l} \text{RECORD IS} \\ \text{RECORDS ARE} \end{array} \right\} \left\{ \begin{array}{l} \text{STANDARD} \\ \text{OMITTED} \end{array} \right\} \right] \\
 & \left[\text{VALUE OF nom-de-réalisateur-1 IS } \left\{ \begin{array}{l} \text{nom-de-donnée-1} \\ \text{littéral-1} \end{array} \right\} \right. \\
 & \quad \left[\text{nom-de-réalisateur-2 IS } \left\{ \begin{array}{l} \text{nom-de-donnée-2} \\ \text{littéral-2} \end{array} \right\} \right] \dots \left. \right] \\
 & \left[\text{DATA } \left\{ \begin{array}{l} \text{RECORD IS} \\ \text{RECORDS ARE} \end{array} \right\} \text{ nom-de-donnée-3 [nom-de-donnée-4] ...} \right] \\
 & \left[\text{LINAGE IS } \left\{ \begin{array}{l} \text{nom-de-donnée-5} \\ \text{entier-5} \end{array} \right\} \text{ LINES } \left[\text{WITH FOOTING AT } \left\{ \begin{array}{l} \text{nom-de-donnée-6} \\ \text{entier-6} \end{array} \right\} \right] \right] \\
 & \quad \left[\text{LINES AT TOP } \left\{ \begin{array}{l} \text{nom-de-donnée-7} \\ \text{entier-7} \end{array} \right\} \right] \left[\text{LINES AT BOTTOM } \left\{ \begin{array}{l} \text{nom-de-donnée-8} \\ \text{entier-8} \end{array} \right\} \right] \\
 & \left[\text{CODE - SET IS nom-d' alphabet} \right] \\
 & \left[\left\{ \begin{array}{l} \text{REPORT IS} \\ \text{REPORTS ARE} \end{array} \right\} \text{ nom-d' état-1 [nom-d' état-2] ...} \right] .
 \end{aligned}$$

- FD est l'abréviation de **F**ile **D**escription et nom-de-fichier est un nom symbolique que le programmeur affecte au fichier considéré ; il doit correspondre au nom de fichier interne de la phrase SELECT.
- La clause BLOCK CONTAINS indique l'utilisation de blocs d'enregistrement. Elle permet également de définir la dimension de ces blocks.
- La clause RECORD CONTAINS sert à indiquer la longueur des enregistrements logiques.
- La clause LABEL RECORD donne des informations sur les labels (enregistrements spéciaux destinés à identifier avec précision les fichiers). OMITTED signifie que le fichier n'a pas de label (c'est le cas pour PRINTER) et STANDARD indique que l'on laisse au système d'exploitation le soin de gérer les labels.
- La clause DATA désigne la structure d'enregistrement associée au fichier.
- La clause LINAGE offre des possibilités de mise en page.

Remarques :

- Les clauses LABEL RECORD et DATA RECORD devraient disparaître dans la prochaine norme.
- Rappelons que la clé relative (RELATIVE KEY) ne se définit pas dans une rubrique du fichier.

Exemple 4-7

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 exemple⁵.

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    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.  
  
Traitement-du-Fichier.  
    OPEN OUTPUT MYFILE.  
    ...
```

⁵ Cette syntaxe est spécifique au COBOL MICRO FOCUS.

Instructions pour la gestion des fichiers

AT END

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

- 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 et cela n'a rien donné, donc le contenu du champ est imprévisible.
- Cette clause ne nécessite pas la définition explicite du `FILE STATUS`.

CLOSE

Fermeture d'un fichier. 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'entrée-sortie. En effet, lors d'un arrêt imprévu du système, on risque de perdre des données si les fichiers ne sont pas fermés.

$$\begin{array}{l} \text{CLOSE nom-de-fichier-1} \left[\begin{array}{l} \left[\begin{array}{l} \text{REEL} \\ \text{UNIT} \end{array} \right] \left[\begin{array}{l} \text{WITH NO REWIND} \\ \text{FOR REMOVAL} \end{array} \right] \\ \text{WITH} \left[\begin{array}{l} \text{NO REWIND} \\ \text{LOCK} \end{array} \right] \end{array} \right] \\ \\ \left[\begin{array}{l} \text{nom-de-fichier-2} \left[\begin{array}{l} \left[\begin{array}{l} \text{REEL} \\ \text{UNIT} \end{array} \right] \left[\begin{array}{l} \text{WITH NO REWIND} \\ \text{FOR REMOVAL} \end{array} \right] \\ \text{WITH} \left[\begin{array}{l} \text{NO REWIND} \\ \text{LOCK} \end{array} \right] \end{array} \right] \dots \end{array} \right] \end{array}$$

- `REEL` et `UNIT` indiquent que le fichier est implanté physiquement sur plusieurs bobines magnétiques (`REEL`) ou unités (`UNIT`). A la fermeture du fichier, la bobine en cours est rembobinée. Avec la clause `FOR REMOVAL`, le système d'exploitation est informé du démontage de cette bobine.
- L'option `NO REWIND` a pour effet de laisser la bobine dans sa position courante (pas de rembobinage).
- L'option `LOCK` interdit toute nouvelle ouverture du fichier concerné durant l'exécution du même programme.

DELETE

Destruction logique d'un enregistrement d'un fichier indexé ou relatif.

DELETE nom-de-fichier RECORD

[INVALID KEY instruction-impérative-1]
[NOT INVALID KEY instruction-impérative-2]
[END - DELETE].

- L'instruction DELETE n'est pas autorisée avec les fichiers séquentiels.
- Le fichier doit être ouvert en mode 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 contrôle⁶.

INVALID KEY

Après toute instruction d'entrée-sortie 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.

- Cette clause ne nécessite pas la définition explicite du FILE STATUS.
- La clause INVALID KEY n'est pas autorisée avec les fichiers séquentiels, en revanche elle est fortement recommandée pour les fichiers indexés et relatifs.
- En COBOL 85, la clause NOT INVALID KEY permet de spécifier la ou les actions à effectuer si l'on ne détecte pas d'erreur dans l'exécution de l'instruction.

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

⁶ cf. page 73

OPEN

Ouverture /création de fichiers

```

OPEN {
  INPUT fich-1 [ { RESERVED
                  WITH NO REWIND
                } ] [ fich-2 [ { RESERVED
                              WITH NO REWIND
                            } ] ] ...
  OUTPUT fich-3 [WITH NO REWIND] [fich-4 [WITH NO REWIND]] ...
  I - O nom-de-fichier-5 [nom-de-fichier-6]...
  EXTEND nom-de-fichier-7 [nom-de-fichier-8]...
} ...
  
```

- 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 automatiquement, 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.
- L'option REVERSED ne peut être utilisée qu'avec un fichier séquentiel enregistré sur une seule bobine ou unité. Quand on l'indique, le fichier est positionné à sa fin lors de l'ouverture. Les articles sont ensuite délivrés par l'instruction READ dans l'ordre inverse, c'est-à-dire en commençant par le dernier article.
- WITH NO REWIND : De même, cette option ne s'applique qu'aux fichiers séquentiels placés sur une seule bobine ou unité. Le fichier doit être positionné à son début. Lors de l'exécution de l'instruction OPEN, il n'y a pas rembobinage du fichier.

READ

Lecture d'enregistrements.

```

READ nom-de-fichier [NEXT] RECORD [INTO identificateur]
  [AT END instruction-impérative-1]
  [NOT AT END instruction-impérative-2]
  [END - READ].
  } lecture séquentielle
  
```

```

READ nom-de-fichier RECORD [INTO identificateur]
  [KEY IS nom-de-donnée]
  [INVALID KEY instruction-impérative-1]
  [NOT INVALID KEY instruction-impérative-2]
  [END - READ].
  } uniquement fichiers
    indexés / relatifs
  
```

- Le fichier doit être ouvert en INPUT ou en I-O.
- La lecture séquentielle en mode d'accès DYNAMIC nécessite la locution NEXT.
- La clause INTO transfère les données vers un autre champ de données, en plus du champ décrit dans la FILE SECTION.
- KEY indique la clé (principale ou secondaire) utilisée pour la recherche des articles du fichier.
- AT END exécute automatiquement la phrase impérative si l'on se trouve à la fin du fichier. Les fichiers relatifs ou indexés peuvent utiliser soit AT END (lecture séquentielle) soit INVALID KEY (lecture par une clé).

REWRITE

Réécriture d'un enregistrement.

```
REWRITE nom-d'article [FROM identificateur]
  [INVALID KEY instruction-impérative-1]
  [NOT INVALID KEY instruction-impérative-2]
  [END - REWRITE].
```

- Pour la réécriture d'enregistrements de fichiers d'organisation quelconque en 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 dans le fichier.
- Si l'enregistrement est écrit avec succès (FILE STATUS = "00") le champ de données n'est plus accessible, dans le cas contraire c'est la phrase impérative après INVALID KEY qui est exécutée.

START

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

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

```
[INVALID KEY instruction-impérative-1]
[NOT INVALID KEY instruction-impérative-2] [END - START].
```

- 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.

WRITE

Ecriture d'enregistrements.

WRITE nom-d' article [FROM identificateur-1]

$$\left[\begin{array}{l} \left\{ \begin{array}{l} \text{AFTER} \\ \text{BEFORE} \end{array} \right\} \text{ADVANCING} \left\{ \begin{array}{l} \text{identificateur-2} \\ \text{entier} \end{array} \right\} \left[\begin{array}{l} \text{LINE} \\ \text{LINES} \end{array} \right] \\ \left\{ \begin{array}{l} \text{nom-mnémonique} \\ \text{PAGE} \end{array} \right\} \end{array} \right] \\ \left[\text{[NOT] AT} \left\{ \begin{array}{l} \text{END - OF - PAGE} \\ \text{EOP} \end{array} \right\} \text{instruction-impérative} \right] \text{[END - WRITE]}.$$

WRITE nom-d' article [FROM identificateur]

$$\left[\text{[NOT] INVALID KEY instruction-impérative} \right] \text{[END - WRITE]}.$$

- La clause FROM permet d'économiser une instruction MOVE en transférant directement le champ de données dans le fichier.
- Les options BEFORE / AFTER ADVANCING servent à créer des fichiers d'édition (préciser LINE SEQUENTIAL), elles sont donc interdites avec des fichiers indexés ou relatifs.
- Si l'enregistrement est écrit avec succès (FILE STATUS = "00") le champ de données n'est plus accessible, dans le cas contraire c'est la phrase impérative après INVALID KEY qui est exécutée.

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				
	DELETE				
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.

Exemple : Fichier mouvement

Enoncé du problème

Une société régionale de marketing téléphonique dispose d'un fichier CONTACTS dont chaque article contient :

TELCNT	PICTURE 9 (8)	numéro de téléphone
NOMCNT	PICTURE X (10)	nom du contact
SFCNT	PICTURE X	situation de famille (C, M, V, A)
RESSCNT	PICTURE X (5)	ressources mensuelles en F
PROFCNT	PICTURE XX	code profession INSEE

Le fichier CONTACTS est trié sur TELCNT, et mis à jour une fois par semaine, à partir du fichier MVTCNT, dont chaque article a même format que CONTACTS, plus un code mouvement et un code jour, placés en tête :

CODMVT	PICTURE X	code mouvement (A, S, M)
CODJR	PICTURE 9	code jour (1-5)

Le fichier MVTCNT est trié sur TELCNT, CODJR et CODMVT.

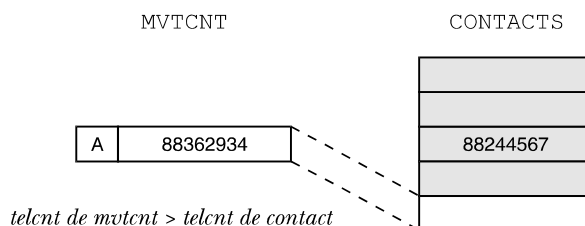
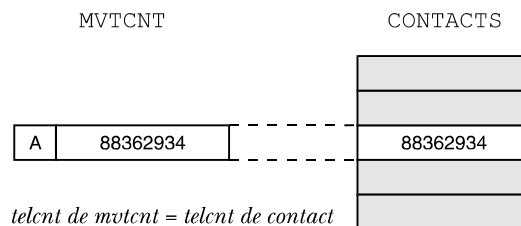
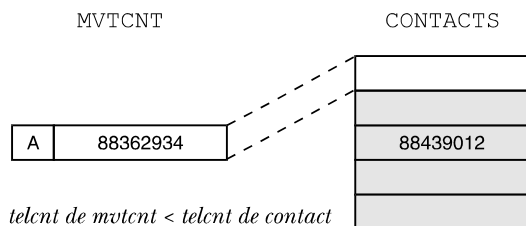
La procédure de mise-à-jour applique les règles suivantes :

1. Une adjonction nécessite la présence de toutes les rubriques.
2. Un article de modification comporte, en sus de TELCNT, uniquement les rubriques à modifier dans l'article de CONTACTS, les autres étant à blanc.
3. Il peut y avoir plusieurs articles de modification pour un même contact, mais au plus un par jour.
4. Toute erreur sur un mouvement implique :
 - a) le rejet de ce mouvement et de tous les mouvements suivants relatifs au même contact,
 - b) la production d'un message d'erreur sur liste, comportant :
 - l'image de l'enregistrement de base
 - l'image du mouvement erroné
 - un message d'erreur explicite

Analyse du problème

La lecture séquentielle ne permettant pas les retours en arrière, il faudra synchroniser l'avancement dans les fichiers MVTCNT et CONTACTS. La comparaison entre les articles des deux fichiers se fera sur le champ TELCNT que l'on peut considérer comme une clé primaire. On distingue alors trois cas :

Différentes configurations pouvant se présenter



Contenu des fichiers

CONTACTS		MVTCNT		NOUVEAU
88102345DUPONT	M1507032	A288050506VERTADET	C1250056	88050506VERTADET C1250056
88210554SCHMITT	M1200045	A388100500VIGNERON	C1250034	88100500VIGNERON C1250034
88211045HEINTZ	C2300078	A288101010MAIRE	C3400023	88101010MAIRE C3400023
88235643DURAND	V0890090	S188235643		88102345DUPONT M1507032
88243434WELSCH	M3450012	M188243434	C	88210554SCHMITT M1200045
88244567ROSEN	M5600045	M388243434	2450000	88211045HEINTZ C2300078
88322323HEREANDNOWC3420067		M588243434	A	88243434WELSCH A2450000
88323232SCHMIDT	V1290092	M288244567ROSY		88244567ROSY M5600045
88345678HABERBINS	C2450056	A188261210ESTANOCH	M1450034	88261210PESTANOCH M1450034
88345689BINSHABER	M1390045	M288261210PESTANOCH		88322323HEREANDNOWC3420067
88346723BERNARD	C2200023	A488345678BIERENBAL	C2300067	88323232SCHMIDT V1290092
88356723MULLER	V0940078	M188346723	M	88345678HABERBINS C2450056
88356734MUNCH	C3400024	M288346723BERNARDINI		88345689BINSHABER M1390045
88356823RIEFFEL	M1740045	S588358112		88346723BERNARDINIM2200023
88357823HAOUN	M4560010	S288361010		88356723MULLER V0940078
88358112MEYER	C1200023	A188362312HEINRICH	C1280045	88356734MUNCH C3400024
88358114HANS	M0760028	M188362312	1280054	88356823RIEFFEL M1740045
88361212BINTZER	A2390034	M188362312	M	88357823HAOUN M4560010
88362534HIRTZEL	V1900034	S288362312		88358114HANS M0760028
88362823FRANCOIS	M0890047	M488362823FRANTZ		88361212BINTZER A2390034
88362910BRUCHER	M1200047	A288364500SCHNITZEL	1306659	88362312HEINRICH C1280054
88362934BERANGER	C0980034	M388364500SCHNITZEL	C	88362534HIRTZEL V1900034
88363810DURAND	V2300037	A588782132TERNINO	A0990034	88362823FRANTZ M0890047
88364510PIERRE	C1490078			88362910BRUCHER M1200047
88364720BIBER	V0890067			88362934BERANGER C0980034
88374567MARTIN	C2310034			88363810DURAND V2300037
88392345ROVALET	C3490056			88364510PIERRE C1490078
88392378BERTRAND	C3410055			88364720BIBER V0890067
88400000TRICHOX	A2130047			88374567MARTIN C2310034
88401000BERARDOT	V1230067			88392345ROVALET C3490056
88410101AICHOUNET	C1230045			88392378BERTRAND C3410055
88439012ECHOUNETTEV1450048				88400000TRICHOX A2130047
				88401000BERARDOT V1230067
				88410101AICHOUNET C1230045
				88439012ECHOUNETTEV1450048
				88782132TERNINO A0990034

Programme source

```

IDENTIFICATION DIVISION.
PROGRAM-ID. MOUVMNT.

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

        SELECT F-CONTACT ASSIGN TO DISK "contacts"
            ORGANIZATION IS LINE SEQUENTIAL.

        SELECT F-MVTCNT ASSIGN TO DISK "mvtcnt"
            ORGANIZATION IS LINE SEQUENTIAL.

        SELECT F-NOUVEAU ASSIGN TO DISK "nouveau"
            ORGANIZATION IS LINE SEQUENTIAL.

        SELECT F-ERREURS ASSIGN TO DISK "erreurs"
            ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.

FD F-CONTACT.
01 cli-contact.
   COPY "regdef.txt".

FD F-MVTCNT.
01 cli-mvtcnt.
   05 codmvt          PICTURE X.
   05 codjr           PICTURE 9.
   COPY "regdef.txt".

FD F-NOUVEAU.
01 cli-nouveau.
   COPY "regdef.txt".

FD F-ERREURS.
01 sortie-erreurs.
   02 cli-base.
      COPY "regdef.txt".
   02 FILLER          PICTURE XX      VALUE
SPACE.
   02 cli-erreurs.
      05 codmvt          PICTURE X.
      05 codjr           PICTURE 9.
      COPY "regdef.txt".
   02 FILLER          PICTURE XX      VALUE
SPACE.
   02 message-err      PICTURE X(23)  VALUE
SPACE.

WORKING-STORAGE SECTION.

77 telrejet          PICTURE 9(8)      VALUE ZERO.

01 bool-fin-fichier  PICTURE 9          VALUE 0.
88 fin-contact       VALUE 1.
88 fin-mvtcnt        VALUE 2.
88 fin-contact-mvtcnt VALUE 3.

01 bool-ajout        PICTURE 9          VALUE 2.
88 en-attente        VALUE 1.
88 pas-en-attente    VALUE 2.

01 precedent.
   05 mvtprec         PICTURE X          VALUE
SPACE.
   05 jrprec          PICTURE 9          VALUE ZERO.
   05 telprec         PICTURE 9(8)      VALUE ZERO.
   05 FILLER          PICTURE X(18)     VALUE
SPACE.

```

Identification du programme.

Déclaration des fichiers :

LINE SEQUENTIAL indique que chaque enregistrement occupe une ligne : cela permet une meilleure visualisation du contenu des fichiers.

Pour être exact, il aurait fallu écrire :
ASSIGN TO PRINTER

Définition des fichiers :

On utilise COPY pour éviter de réécrire la structure du fichier à chaque fois et pour prévenir d'éventuelles erreurs de format. Le contenu du fichier *regdef.txt* est le suivant :

```

05 telcnt          PICTURE 9(8).
05 nomcnt          PICTURE X(10).
05 sfcnt           PICTURE X.
05 resscnt         PICTURE X(5).
05 profcnt         PICTURE XX.

```

Remarquez l'utilisation de FILLER pour insérer des espaces entre les images de l'enregistrement de base et du mouvement erroné. COPY assure l'équivalence des noms de champs qui est nécessaire à l'affectation intelligente du MOVE CORRESPONDING.

Définition des données :

telrejet est le numéro de téléphone du dernier mouvement rejeté.

Ici nous utilisons d'une manière peu commune les noms de condition. Il est possible, en effet, d'additionner un entier à *bool-fin-fichier*, la somme de *fin-contact* et de *fin-mvtcnt* sera dès lors *fin-contact-mvtcnt*.

bool-ajout indique si un ajout a été préparé, donc s'il est en attente d'être écrit définitivement.

```
PROCEDURE DIVISION.
Ouverture-des-Fichiers.
    OPEN INPUT F-CONTACT F-MVTCNT.
    OPEN OUTPUT F-NOUVEAU F-ERREURS.
    READ F-CONTACT AT END ADD 1 TO bool-fin-fichier.
    READ F-MVTCNT AT END ADD 2 TO bool-fin-fichier.

Principal.
    PERFORM Traitement UNTIL fin-contact-mvtcnt.
    PERFORM Ecrire-Nouveau-Si-Ajout.
    CLOSE F-CONTACT F-MVTCNT F-NOUVEAU F-ERREURS.
    STOP RUN.

Traitement.
    IF fin-mvtcnt
        PERFORM Ecrire-Nouveau-Si-Ajout
        WRITE cli-nouveau FROM cli-contact
        PERFORM Lecture-Contact-Suivant
    ELSE
        IF fin-contact
            MOVE 99999999 TO telcnt OF cli-contact
            PERFORM Traitement-Enregistrements
        ELSE
            PERFORM Traitement-Enregistrements.

Traitement-Enregistrements.
    IF telcnt OF cli-mvtcnt = telrejet
        MOVE "Mouvement rejeté !" TO message-err
        PERFORM Sortie-Fichier-Erreurs
    ELSE
        IF codmvt OF cli-mvtcnt = "A" PERFORM Ajout
        ELSE IF codmvt OF cli-mvtcnt = "M" PERFORM Modif
        ELSE IF codmvt OF cli-mvtcnt = "S" PERFORM Suppr
        ELSE
            MOVE "Opération inconnue !" TO message-err
            PERFORM Sortie-Fichier-Erreurs.

Ajout.
    IF telcnt OF cli-mvtcnt < telcnt OF cli-contact
        PERFORM Ecrire-Nouveau-Si-Ajout
        PERFORM Preparer-Ajout-Si-Valide
    ELSE
        IF telcnt OF cli-mvtcnt = telcnt OF cli-contact
            MOVE "Numéro existant !" TO message-err
            PERFORM Sortie-Fichier-Erreurs
        ELSE
            PERFORM Ecrire-Nouveau-Si-Ajout
            WRITE cli-nouveau FROM cli-contact
            PERFORM Lecture-Contact-Suivant.

Modif.
    IF telprec = telcnt OF cli-mvtcnt
        AND jrprec = codjr OF cli-mvtcnt AND mvtprec = "M"
            MOVE "2 modifs le même jour" TO message-err
            PERFORM Sortie-Fichier-Erreurs
    ELSE
        IF telcnt OF cli-mvtcnt < telcnt OF cli-contact
            IF en-attente
                AND telprec = telcnt OF cli-mvtcnt
                PERFORM Preparer-Modification-Nouveau
                PERFORM Lecture-Mouvement-Suivant
            ELSE
                MOVE "Article non trouvé !" TO message-err
                PERFORM Sortie-Fichier-Erreurs
            END-IF
        ELSE
            IF telcnt OF cli-mvtcnt = telcnt OF cli-contact
                PERFORM Ecrire-Nouveau-Si-Ajout
                PERFORM Preparer-Modification-Contact
                PERFORM Lecture-Mouvement-Suivant
```

Partie des traitements :

Ouverture des fichiers et initialisation des champs cli-contact et cli-mvtcnt par une première lecture.

On traite les enregistrements jusqu'à la fin des deux fichiers. Ensuite, s'il reste un ajout en attente d'écriture on l'écrit. Enfin, tous les fichiers sont fermés.

Si fin-mvtcnt est vrai, cela implique par la condition précédente (PERFORM Traitement UNTIL fin-contact-mvtcnt) qu'il reste encore des articles dans le fichier CONTACTS. On procède donc à un vidage du fichier.

De même, si fin-contact est vrai, il reste encore des mouvements à traiter. Le MOVE 99999999 TO telcnt est très important car il impose l'aiguillage correct, sachant que le fichier CONTACTS ne doit pas être relu.

Nous avons maintenant deux nouveaux articles face à face (cf. schéma) ; on commence par vérifier si le mouvement courant n'est pas à rejeter. Le traitement est ensuite réparti sur trois sous-procédures. Si le code mouvement ne correspond à aucun des cas (A, M, S), c'est que le fichier MVTCNT est erroné.

On distingue trois cas :

- < insérer mouvement avant contact.
- = ajout sur un article existant !
- > rechercher la clé immédiatement supérieure à telcnt de cli-mvtcnt.

Dans le premier et le troisième cas, il faudra penser à écrire d'éventuels ajouts en attente.

On vérifie qu'il n'y a pas eu le même jour une modification sur le même article. En effet, il serait impossible de fixer des priorités sur de tels mouvements.

On distingue à nouveau trois cas :

- < deux sous-cas :
 - a) la modification concerne un ajout ou une modification précédente en attente ⇒ modifier le champ en attente.
 - b) on est trop loin, l'article en question n'a pas été trouvé !

= modifier les champs non vides.

```

ELSE
    PERFORM Ecrire-Nouveau-Si-Ajout
    WRITE cli-nouveau FROM cli-contact
    PERFORM Lecture-Contact-Suivant.

Suppr.
IF telcnt OF cli-mvtcnt < telcnt OF cli-contact
    IF en-attente AND telprec = telcnt OF cli-mvtcnt
        SET pas-en-attente TO TRUE
        PERFORM Lecture-Mouvement-Suivant
    ELSE
        MOVE "Article non trouvé !" TO message-err
        PERFORM Sortie-Fichier-Erreurs
    END-IF
ELSE
    IF telcnt OF cli-mvtcnt = telcnt OF cli-contact
        PERFORM Ecrire-Nouveau-Si-Ajout
        PERFORM Lecture-Contact-Suivant
        PERFORM Lecture-Mouvement-Suivant
    ELSE
        PERFORM Ecrire-Nouveau-Si-Ajout
        WRITE cli-nouveau FROM cli-contact
        PERFORM Lecture-Contact-Suivant.

Ecrire-Nouveau-Si-Ajout.
    IF en-attente
        WRITE cli-nouveau
        SET pas-en-attente TO TRUE.

Preparer-Ajout-Si-Valide.
    IF resscnt OF cli-mvtcnt NOT NUMERIC
        OR profcnt OF cli-mvtcnt NOT NUMERIC
        OR (SPACE = nomcnt OF cli-mvtcnt OR sfcnt OF cli-mvtcnt)

        MOVE "Rubrique manquante !" TO message-err
        PERFORM Sortie-Fichier-Erreurs
    ELSE
        MOVE CORRESPONDING cli-mvtcnt TO cli-nouveau
        SET en-attente TO TRUE
        PERFORM Lecture-Mouvement-Suivant.

Sortie-Fichier-Erreurs.
    MOVE telcnt OF cli-mvtcnt TO telrejet.
    MOVE CORRESPONDING cli-contact TO cli-base.
    MOVE CORRESPONDING cli-mvtcnt TO cli-erreurs.
    WRITE sortie-erreurs.
    PERFORM Lecture-Mouvement-Suivant.

Lecture-Mouvement-Suivant.
    MOVE cli-mvtcnt TO precedent.
    READ F-MVTCNT AT END ADD 2 TO bool-fin-fichier.

Lecture-Contact-Suivant.
    READ F-CONTACT AT END ADD 1 TO bool-fin-fichier.

Preparer-Modification-Contact.
    IF nomcnt OF cli-mvtcnt NOT = SPACE
        MOVE nomcnt OF cli-mvtcnt TO nomcnt OF cli-contact.

    IF sfcnt OF cli-mvtcnt NOT = SPACE
        MOVE sfcnt OF cli-mvtcnt TO sfcnt OF cli-contact.

    IF resscnt OF cli-mvtcnt NUMERIC
        MOVE resscnt OF cli-mvtcnt TO resscnt OF cli-contact.

    IF profcnt OF cli-mvtcnt NUMERIC
        MOVE profcnt OF cli-mvtcnt TO profcnt OF cli-contact.

```

> rechercher la clé correspondant à *telcnt* de *cli-mvtcnt*.

La suppression se traduit ici par une non-écriture. Encore trois cas :

< deux sous-cas :

a) la modification concerne un ajout ou une modification précédente en attente ⇒ déclarer qu'il n'y a plus d'attente.

b) on est trop loin, l'article en question n'a pas été trouvé !

= ignorer le contact courant.

> rechercher la clé correspondant à *telcnt* de *cli-mvtcnt*.

Écriture des ajouts en attente à partir de *cli-nouveau* où ils sont stockés. Cette utilisation du SET ne marche qu'avec TRUE.

Les ajouts ne sont pas écrits tout de suite car il se peut que le mouvement suivant soit une modification sur l'article qui vient d'être ajouté.

Si l'ajout est valide on le copie vers *cli-nouveau*, mais il faudra alors penser à l'écrire ; *bool-ajout* permet de savoir à chaque instant si un ajout est en attente et donc d'agir en conséquence.

Le mouvement est erroné, donc on commence par enregistrer son numéro dans *telrejet*.

Le MOVE CORRESPONDING est pratique, mais il ne faut pas oublier de qualifier chaque nom de donnée.

Le traitement de la fin de fichier est transparent au reste du programme.

Ici on écrit directement dans les champs d'un fichier ouvert en lecture seule ! Mais n'essayez pas un WRITE ou REWRITE sur *cli-contact* ; la seule instruction valide est évidemment :

```
WRITE cli-nouveau FROM cli-contact
```

N'oublions pas cependant qu'un WRITE exécuté avec succès entraîne la perte des données du champ associé (dans notre cas F-NOUVEAU).

Preparer-Modification-Nouveau.

```
IF      nomcnt  OF cli-mvtcnt NOT = SPACE
MOVE nomcnt  OF cli-mvtcnt TO nomcnt OF cli-nouveau.

IF      sfcnt   OF cli-mvtcnt NOT = SPACE
MOVE sfcnt   OF cli-mvtcnt TO sfcnt OF cli-nouveau.

IF      resscnt OF cli-mvtcnt NUMERIC
MOVE resscnt OF cli-mvtcnt TO resscnt OF cli-nouveau.

IF      profcnt OF cli-mvtcnt NUMERIC
MOVE profcnt OF cli-mvtcnt TO profcnt OF cli-nouveau.
```

On se trouve dans le cas particulier où un enregistrement vient d'être ajouté (*en-attente* = TRUE). Celui-ci est en attente d'écriture dans *cli-nouveau*. L'enregistrement suivant a déjà été lu et se trouve actuellement dans *cli-contact*. Il ne faut pas surtout pas toucher à *cli-contact*, mais modifier directement le champ *cli-nouveau* qui sera écrit ultérieurement.

ERREURS

88345678HABERBINS	C2450056	A488345678BIERENBAL	C2300067	Numéro existant !
88361212BINTZER	A2390034	S288361010		Article non trouvé !
88362534HIRTZEL	V1900034	M188362312	M	2 modifs le même jour
88362534HIRTZEL	V1900034	S288362312		Mouvement rejeté !
88364510PIERRE	C1490078	A288364500SCHNITZEL	1306659	Rubrique manquante !
88364510PIERRE	C1490078	M388364500SCHNITZEL	C	Mouvement rejeté !

CONTACTS

MVTCNT

NOUVEAU

88102345DUPONT	M1507032	A288050506VERTADET	C1250056
88210554SCHMITT	M1200045	A388100500VIGNERON	C1250034
88211045HEINTZ	C2300078	A288101010MAIRE	C3400023
88235643DURAND	V0890090	S188235643	
88243434WELSCH	M3450012	M188243434	C
88244567ROSEN	M5600045	M388243434	2450000
88322323HEREANDNOWC	3420067	M588243434	A
88323232SCHMIDT	V1290092	M288244567ROSY	
88345678HABERBINS	C2450056	A188261210ESTANOCH	M1450034
88345689BINSHABER	M1390045	M288261210PESTANOCH	
88346723BERNARD	C2200023	A488345678BIERENBAL	C2300067
88356723MULLER	V0940078	M188346723	M
88356734MUNCH	C3400024	M288346723BERNARDINI	
88356823RIEFFEL	M1740045	S588358112	
88357823HAOUN	M4560010	S288361010	
88358112MEYER	C1200023	A188362312HEINRICH	C1280045
88358114HANS	M0760028	M188362312	1280054
88361212BINTZER	A2390034	M188362312	M
88362534HIRTZEL	V1900034	S288362312	
88362823FRANCOIS	M0890047	M488362823FRANTZ	
88362910BRUCHER	M1200047	A288364500SCHNITZEL	1306659
88362934BERANGER	C0980034	M388364500SCHNITZEL	C
88363810DURAND	V2300037	A588782132TERNINO	A0990034
88364510PIERRE	C1490078		
88364720BIBER	V0890067		
88374567MARTIN	C2310034		
88392345ROVALET	C3490056		
88392378BERTRAND	C3410055		
88400000TRICHOX	A2130047		
88401000BERARDOT	V1230067		
88410101AICHOUNET	C1230045		
88439012ECHOUNETTEV	1450048		

88050506VERTADET	C1250056
88100500VIGNERON	C1250034
88101010MAIRE	C3400023
88102345DUPONT	M1507032
88210554SCHMITT	M1200045
88211045HEINTZ	C2300078
88243434WELSCH	A2450000
88244567ROSY	M5600045
88261210PESTANOCH	M1450034
88322323HEREANDNOWC	3420067
88323232SCHMIDT	V1290092
88345678HABERBINS	C2450056
88345689BINSHABER	M1390045
88346723BERNARDINI	M2200023
88356723MULLER	V0940078
88356734MUNCH	C3400024
88356823RIEFFEL	M1740045
88357823HAOUN	M4560010
88358114HANS	M0760028
88361212BINTZER	A2390034
88362312HEINRICH	C1280054
88362534HIRTZEL	V1900034
88362823FRANTZ	M0890047
88362910BRUCHER	M1200047
88362934BERANGER	C0980034
88363810DURAND	V2300037
88364510PIERRE	C1490078
88364720BIBER	V0890067
88374567MARTIN	C2310034
88392345ROVALET	C3490056
88392378BERTRAND	C3410055
88400000TRICHOX	A2130047
88401000BERARDOT	V1230067
88410101AICHOUNET	C1230045
88439012ECHOUNETTEV	1450048
88782132TERNINO	A0990034

Tri-fusion

SORT

L'instruction SORT sert à trier des fichiers. Le COBOL permet l'introduction de séquences de programme en début et/ou en fin de tri, pour modifier ou d'éliminer des enregistrements avant et/ou après le tri.

Le format de l'instruction SORT semble a priori assez compliqué, nous allons donc classer ses fonctionnalités en quatre sous-catégories afin d'en simplifier l'approche.

1. Tri sans modification des enregistrements en entrée et sortie :

```
SORT nom-du-fichier-de-tri USING nom-de-fichier-2 [nom-de-fichier-3]...
                                GIVING nom-de-fichier-4.
```

Il n'est pas utile d'ouvrir, de lire, d'écrire ou de fermer les fichiers d'entrée et de sortie car le tri gère l'ensemble des fichiers.

2. Tri avec modification des enregistrements en entrée :

```
SORT nom-du-fichier-de-tri
    INPUT PROCEDURE IS nom-de-section-1 [ { THROUGH } nom-de-section-2 ]
    GIVING nom-de-fichier-4.
```

Dans la section appelée on écrira :

```
OPEN ..... fichier en entrée
READ ..... fichier en entrée
RELEASE..... enregistrement du fichier-de-tri décrit en SD
```

3. Tri avec modification des enregistrements en sortie :

```
SORT nom-du-fichier-de-tri
    USING nom-de-fichier-2 [nom-de-fichier-3]...
    OUTPUT PROCEDURE IS nom-de-section-3 [ { THROUGH } nom-de-section-4 ].
```

Dans la section appelée on écrira :

```
OPEN ..... fichier en sortie
RETURN..... nom du fichier-de-tri décrit en SD .....AT END phrase impérative
WRITE ..... enregistrement du fichier en sortie.
```

4. Tri avec modification des enregistrements en entrée et en sortie :

SORT nom-du-fichier-de-tri

$$\begin{aligned} & \text{INPUT PROCEDURE IS nom-de-section-1} \left[\begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right] \text{nom-de-section-2} \\ & \text{OUTPUT PROCEDURE IS nom-de-section-3} \left[\begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right] \text{nom-de-section-4} \end{aligned}$$

Dans la section appelée on écrira :

nom-de-section-1 SECTION.

READ.....fichier en entrée

RELEASEenregistrement du fichier-de-tri décrit en SD

nom-de-section-3 SECTION.

RETURNnom du fichier-de-tri décrit en SD.....AT END phrase impérative

WRITEenregistrement du fichier en sortie.

Format général

SORT nom-de-fichier-1 ON $\left\{ \begin{array}{c} \text{ASCENDING} \\ \text{DESCENDING} \end{array} \right\}$ KEY nom-de-donnée-1 [nom-de-donnée-2]...

$\left[\text{ON} \left\{ \begin{array}{c} \text{ASCENDING} \\ \text{DESCENDING} \end{array} \right\} \text{KEY nom-de-donnée-3 [nom-de-donnée-4]} \dots \right]$

$\left\{ \begin{array}{l} \text{INPUT PROCEDURE IS nom-de-section-1} \left[\begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right] \text{nom-de-section-2} \\ \text{USING nom-de-fichier-2 [nom-de-fichier-3]} \dots \end{array} \right\}$

$\left\{ \begin{array}{l} \text{OUTPUT PROCEDURE IS nom-de-section-3} \left[\begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right] \text{nom-de-section-4} \\ \text{GIVING nom-de-fichier-4} \end{array} \right\}.$

☒ Remarque :

La clause SELECT est obligatoire pour tous les fichiers et aura le format suivant :

SELECT fich-de-tri ASSIGN TO nom-fich-syst [SORT STATUS IS identificateur].

`SORT STATUS` est un indicateur de format `XX` qui doit être défini en `WORKING-STORAGE SECTION`. Cet indicateur joue un rôle semblable au `FILE STATUS`, c'est-à-dire qu'il prend les valeurs suivantes :

<i>Valeur</i>	<i>Signification</i>
00	Tri correctement effectué
10	Mémoire insuffisante pour le tri
11	Tentative de trier un fichier déjà en cours de tri par une autre procédure
71	Erreur à l'ouverture du fichier (OPEN)
72	Erreur à la fermeture du fichier (CLOSE)
73	Erreur en écriture (WRITE)
74	Erreur en lecture (READ)
76	Erreur en suppression (DELETE)
80	Erreur interne au tri
81	
82	

- Une description `SD` (`SORT DESCRIPTION`) est obligatoire pour chaque fichier à trier et cette description suit les mêmes règles que les descriptions de fichier `FD` (`FILE DESCRIPTION`) mises à part les clauses `BLOCK CONTAINS` et `LABEL RECORD` qui n'ont ici aucune utilité.
- `ASCENDING` indique que le tri est croissant, `DESCENDING` qu'il est décroissant ; `KEY` désigne l'argument de tri.
- Pour un fichier, il ne peut y avoir plus de 12 arguments de tri et la somme des longueurs de ces arguments ne peut excéder 255 caractères.
- Les clés ne peuvent appartenir à une zone `OCCURS`.

RELEASE

`RELEASE` nom-d'article [`FROM` identificateur].

L'ordre `RELEASE` ne peut être utilisé que dans une section `INPUT` et est destiné à fournir au module de tri les enregistrements du fichier source, éventuellement à partir d'une autre zone (`FROM`).

RETURN

`RETURN` nom-de-fichier `RECORD` [`INTO` identificateur]
[`AT` `END` instruction-impérative].

L'ordre `RETURN` ne peut être utilisé que dans une section `OUTPUT` et est destiné à recevoir du module de tri les enregistrements du fichier source qui vient d'être trié et les délivrer éventuellement dans une zone particulière (`INTO`).

MERGE

Il est possible de fusionner des fichiers déjà triés sur des arguments identiques, en utilisant le verbe `MERGE`. La programmation est identique à celle du tri `SORT`, à ceci près qu'il n'est pas admis de modification des enregistrements en entrée (`INPUT PROCEDURE`).

```
MERGE nom-de-fichier-1 ON { ASCENDING
                           DESCENDING } KEY nom-de-donnée-1
                               [nom-de-donnée-2] ...
[ ON { ASCENDING
      DESCENDING } KEY nom-de-donnée-3 [nom-de-donnée-4] ... ]
USING nom-de-fichier-2 nom-de-fichier-3 [nom-de-fichier-4] ...

{ OUTPUT PROCEDURE IS nom-de-section-1 [ { THROUGH
      THRU } nom-de-section-2 ] }
GIVING nom-de-fichier-5
```

Exemple

Afin d'illustrer le fonctionnement des instructions `SORT` et `MERGE`, nous vous proposons une autre manière d'effectuer la mise à jour des fichiers présentée à la page 83. Toutefois, les fichiers d'entrée ne sont pas nécessairement triés : pour ce faire, nous utiliserons l'instruction `SORT`. Quant à la synchronisation des articles, c'est l'instruction `MERGE` qui s'en charge - il n'y a donc plus lieu de faire une étude de cas qui est, certes, assez compliquée.

Remarquez également que les enregistrements de base du fichier *erreurs* ont changé :

88345678HABERBINS	C2450056	A488345678BIERENBAL	C2300067	Numéro Existant !
88358114HANS	M0760028	S288361010		Article non trouvé !
88362312HEINRICH	C1280054	M188362312	M	2 modifs le même jour
88362312HEINRICH	C1280054	S288362312		Mouvement rejeté !
88363810DURAND	V2300037	A288364500SCHNITZEL	1306659	Rubrique manquante !
88363810DURAND	V2300037	M388364500SCHNITZEL	C	Mouvement rejeté !

IDENTIFICATION DIVISION.
PROGRAM-ID. EXSORT2.

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

```
SELECT F-CONTACT ASSIGN TO DISK      "contact"
ORGANIZATION IS LINE SEQUENTIAL.
SELECT F-CONTACT-2 ASSIGN TO DISK    "contact2".
```

Identification du programme.

`LINE SEQUENTIAL` indique que chaque enregistrement occupe une ligne : cela permet une meilleure visualisation du contenu des fichiers ; c'est donc une option inutile pour le fichier intermédiaire `CONTACT-2`.


```

SELECT F-MVTCNT ASSIGN TO DISK      "mvtcnt"
ORGANIZATION IS LINE SEQUENTIAL.
SELECT F-MVTCNT-2 ASSIGN TO DISK    "mvtcnt2".

SELECT F-NOUVEAU ASSIGN TO DISK     "nouveau"
ORGANIZATION IS LINE SEQUENTIAL.
SELECT F-ERREURS ASSIGN TO DISK     "erreurs"
ORGANIZATION IS LINE SEQUENTIAL.

SELECT F-SORT-1 ASSIGN TO DISK      "tri-1".
SELECT F-SORT-2 ASSIGN TO DISK      "tri-2".
SELECT F-MERGE ASSIGN TO DISK       "fusion".

DATA DIVISION.
FILE SECTION.
FD F-CONTACT.
01 cli-contact.
   COPY "regdef.txt".
FD F-CONTACT-2.
01 cli-contact-2.
   COPY "regdef.txt".
   05 FILLER                                PICTURE XX.
FD F-MVTCNT.
01 cli-mvtcnt.
   05 codmvt                                PICTURE X.
   05 codjr                                PICTURE 9.
   COPY "regdef.txt".
FD F-MVTCNT-2.
01 cli-mvtcnt-2.
   COPY "regdef.txt".
   05 codmvt                                PICTURE X.
   05 codjr                                PICTURE 9.
FD F-NOUVEAU.
01 cli-nouveau.
   COPY "regdef.txt".
FD F-ERREURS.
01 sortie-erreurs.
   02 cli-base.
      COPY "regdef.txt".
   02 FILLER                                PICTURE XX    VALUE SPACE.
   02 cli-erreurs.
      05 codmvt                                PICTURE X.
      05 codjr                                PICTURE 9.
      COPY "regdef.txt".
   02 FILLER                                PICTURE XX    VALUE SPACE.
   02 message-err                            PICTURE X(23) VALUE SPACE.

SD F-SORT-1.
01 cli-sort-1.
   COPY "regdef.txt".
   05 FILLER                                PICTURE XX.
SD F-SORT-2.
01 cli-sort-2.
   05 codmvt                                PICTURE X.
   05 codjr                                PICTURE 9.
   COPY "regdef.txt".
SD F-MERGE.
01 cli-merge.
   COPY "regdef.txt".
   05 codmvt                                PICTURE X.
   05 codjr                                PICTURE 9.

WORKING-STORAGE SECTION.
77 telrejet                                PICTURE 9(8)    VALUE ZERO.

01 bool-ajout                                PICTURE 9      VALUE 2.
   88 en-attente                            VALUE 1.
   88 pas-en-attente                        VALUE 2.

01 precedent.
   05 mvtprec                                PICTURE X      VALUE SPACE.
   05 jrprec                                PICTURE 9      VALUE ZERO.

```

On aurait aussi pu écrire :
ASSIGN TO PRINTER

Déclaration des fichiers tri (en fait on réserve juste des enregistrements)

Ici, rien ne change par rapport à l'ancienne méthode, si ce n'est que l'on ajoute des espaces en fin de rubrique pour harmoniser la structure des fichiers pour le MERGE.

MERGE impose que les clés des fichiers d'entrée soient situées à la même position. Il sera donc nécessaire de déplacer *codmvt* et *codjr* vers la fin de la rubrique du fichier MVTCNT-2.

F-ERREURS n'a pas changé...

Définition des structures de tri.

F-SORT-1 servira à trier CONTACTS, F-SORT-2 à trier MVTCNT, et enfin F-MERGE sera utile au traitement de la mise à jour.

Notez tout particulièrement la position du code mouvement et du code jour.

telrejet est le numéro du dernier mouvement erroné.

precedent s'est allégé de quelques champs, puisqu'on ira chercher ces informations dans *cli-nouveau*.

```
PROCEDURE DIVISION.
PRINCIPAL SECTION.

*--- tri avec modification des enregistrements en entrée

OPEN INPUT F-CONTACT.
SORT F-SORT-1
  ON ASCENDING KEY telcnt OF cli-sort-1
  INPUT PROCEDURE IS Tri-Contact THROUGH Fin-Tri-Contact
  GIVING F-CONTACT-2.
CLOSE F-CONTACT.

*--- tri avec modification des enregistrements en sortie

OPEN OUTPUT F-MVTCNT-2.
SORT F-SORT-2
  ON ASCENDING KEY telcnt OF cli-sort-2
                  codjr  OF cli-sort-2
                  codmvt OF cli-sort-2
  USING F-MVTCNT
  OUTPUT PROCEDURE IS Tri-Mvtcnt THROUGH Fin-Tri-Mvtcnt.
CLOSE F-MVTCNT-2.

*--- fusion avec modification des enregistrements en sortie

OPEN OUTPUT F-NOUVEAU F-ERREURS.
MERGE F-MERGE
  ON ASCENDING KEY telcnt OF cli-merge
                  codjr  OF cli-merge
                  codmvt OF cli-merge

  USING F-CONTACT-2 F-MVTCNT-2
  OUTPUT PROCEDURE IS Fusion THROUGH Fin-Fusion.

IF en-attente
  WRITE cli-nouveau.
CLOSE F-NOUVEAU F-ERREURS.
STOP RUN.

CONTACT SECTION.
Tri-Contact.
  READ F-CONTACT AT END GO TO Fin-Tri-Contact.
  MOVE CORRESPONDING cli-contact TO cli-sort-1.
  RELEASE cli-sort-1.
  GO TO TRI-CONTACT.

Fin-Tri-Contact.
  EXIT.

MVTCNT SECTION.
Tri-Mvtcnt.
  RETURN F-SORT-2 AT END GO TO Fin-Tri-Mvtcnt.
  MOVE CORRESPONDING cli-sort-2 TO cli-mvtcnt-2.
  WRITE cli-mvtcnt-2.
  GO TO Tri-Mvtcnt.

Fin-Tri-Mvtcnt.
  EXIT.

TRAITEMENT-CONTACT-MVTCNT SECTION.
Fusion.
  RETURN F-MERGE AT END GO TO Fin-Fusion.
  IF telcnt OF cli-merge = telrejet
    MOVE "Mouvement rejeté !" TO message-err
    PERFORM Sortie-Fichier-Erreurs
```

Le premier SORT est un SORT INPUT, le programmeur ne n'a donc que le fichier d'entrée à gérer.

On trie les contacts par numéro de téléphone, le fichier résultant est CONTACT-2.

Le second SORT est un SORT OUTPUT, ici le programmeur le fichier de sortie MVTCNT-2. On trie sur trois clés : le numéro de téléphone, le jour et le code mouvement (A, M, S).

Enfin, la fusion : on s'occupe de NOUVEAU et d'ERREURS, MERGE se charge de CONTACT-2 et de MVTCNT-2.

Du tri sur le code mouvement il résulte que les articles de MVTCNT-2 sont placés après les articles de CONTACT-2 (dont le code mouvement est un espace). Ceci est très important, car c'est la seule façon de gérer correctement les modifications et suppressions sur les articles de CONTACT-2.

La CONTACT SECTION regroupe tout ce qui concerne le fichier CONTACT. Ici le traitement consiste à ajouter des espaces à la fin des rubriques. RELEASE transmet l'enregistrement modifié à la fonction SORT.

La MVTCNT SECTION regroupe tout ce qui concerne le fichier MVTCNT. RETURN met à notre disposition l'enregistrement courant trié. Le MOVE CORRESPONDING à pour effet de déplacer *codmvt* et *codjr* à la fin des rubriques.

La fusion serait en fait plutôt destinée à des ajouts, mais il est tout-à-fait possible d'indiquer un traitement complexe dans l'OUTPUT PROCEDURE.

```

ELSE
    EVALUATE codmvt OF cli-merge
        WHEN SPACE PERFORM Ajout
        WHEN "A" PERFORM Ajout
        WHEN "M" PERFORM Modif
        WHEN "S" PERFORM Suppr
        WHEN OTHER
            MOVE "Opération Inconnue !" TO message-err
            PERFORM Sortie-Fichier-Erreurs
    END-EVALUATE
END-IF.
MOVE codmvt OF cli-merge TO mvtprec.
MOVE codjr OF cli-merge TO jrprec.
GO TO Fusion.

Ajout.
IF resscnt OF cli-merge NOT NUMERIC
OR profcnt OF cli-merge NOT NUMERIC
OR ( SPACE = nomcnt OF cli-merge OR sfcnt OF cli-merge )

    MOVE "Rubrique manquante !" TO message-err
    PERFORM Sortie-Fichier-Erreurs
ELSE
    IF telcnt OF cli-merge = telcnt OF cli-nouveau
        MOVE "Numéro Existant !" TO message-err
        PERFORM Sortie-Fichier-Erreurs
    ELSE
        IF en-attente
            WRITE cli-nouveau
        END-IF
        MOVE CORRESPONDING cli-merge TO cli-nouveau
        SET en-attente TO TRUE.

Modif.
IF telcnt OF cli-merge = telcnt OF cli-nouveau
    IF jrprec = codjr OF cli-merge AND mvtprec = "M"
        MOVE "2 modifs le même jour" TO message-err
        PERFORM Sortie-Fichier-Erreurs
    ELSE
        PERFORM Preparer-Modification-Nouveau
    ELSE
        MOVE "Article non trouvé" TO message-err
        PERFORM Sortie-Fichier-Erreurs.

Suppr.
IF en-attente AND
telcnt OF cli-merge = telcnt OF cli-nouveau
    SET pas-en-attente TO TRUE
ELSE
    MOVE "Article non trouvé !" TO message-err
    PERFORM Sortie-Fichier-Erreurs.

Sortie-Fichier-Erreurs.
MOVE telcnt OF cli-merge TO telrejet.
MOVE CORRESPONDING cli-nouveau TO cli-base.
MOVE CORRESPONDING cli-merge TO cli-erreurs.
WRITE sortie-erreurs.

Preparer-Modification-Nouveau.
IF nomcnt OF cli-merge NOT = SPACE
    MOVE nomcnt OF cli-merge TO nomcnt OF cli-nouveau.
IF sfcnt OF cli-merge NOT = SPACE
    MOVE sfcnt OF cli-merge TO sfcnt OF cli-nouveau.
IF resscnt OF cli-merge NUMERIC
    MOVE resscnt OF cli-merge TO resscnt OF cli-nouveau.
IF profcnt OF cli-merge NUMERIC
    MOVE profcnt OF cli-merge TO profcnt OF cli-nouveau.

Fin-Fusion.
EXIT.

```

EVALUATE nous dispense ici de l'écriture de structures IF imbriquées. En effet, à partir de cinq conditions exclusives il est plus intéressant d'utiliser EVALUATE.

Sauvegarde des informations importantes concernant l'enregistrement précédent.

Les procédures suivantes ressemblent de très près à celles de la méthode classique. On remarquera que nous gardons le système des enregistrements en attente, sachant toutefois que leur gestion se trouve grandement simplifiée par le MERGE.

On intègre ici les procédures Ecrire-Nouveau-Si-Ajout et Préparer-Ajout-Si-Valide de la méthode précédente.

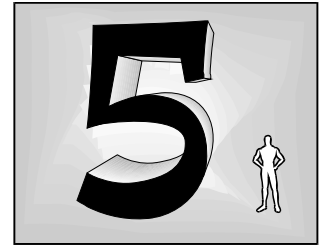
Plus besoin de *telprec*, *telcnt OF cli-nouveau* fait très bien l'affaire puisque si l'erreur a lieu, c'est qu'un mouvement a déjà été effectué sur cli-nouveau le même jour.

Une fois de plus nous pouvons utiliser *cli-nouveau* au lieu de *précédent*. En fait, cette gestion est beaucoup plus simple puisque l'on écrit toujours à partir de *cli-nouveau*. Dans la méthode classique, il fallait tenir compte des enregistrements lus en avance et les répartir sur *cli-contact* et *cli-nouveau*. Ensuite l'écriture se faisait à partir de *cli-contact*, ou de *cli-nouveau*, suivant le cas...

Traitement des erreurs

Aide à la mise au point	99
WITH DEBUGGING MODE	99
READY / RESET TRACE	99
DECLARATIVES - USE	100
Illegal character in numeric field	102

Traitement des erreurs



Aide à la mise au point

WITH DEBUGGING MODE

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

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE – COMPUTER. nom-de-calculateur [WITH DEBUGGING MODE].

- 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.

READY / RESET TRACE

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.

PROCEDURE DIVISION.

...

<u>READY TRACE.</u>	} <i>partie du programme qui sera tracée.</i>
...	
<u>RESET TRACE.</u>	

...

- Le traçage n'est possible que si l'on précise à la compilation : `COBOL nom-fichier TRACE`

DECLARATIVES - USE

Il s'agit d'une section spéciale dans la PROCEDURE DIVISION que l'on utilise pour le traitement des incidents d'entrée-sortie et pour la mise au point du programme.

PROCEDURE DIVISION.

DECLARATIVES.

nom - de - section SECTION. USE ...

nom - de - paragraphe.

...

END DECLARATIVES.

USE FOR DEBUGGING ON

USE est une instruction permettant de définir les éléments qui seront suivis durant l'exécution du programme.

nom-de-section SECTION [numéro-de-section].

<u>USE FOR DEBUGGING ON</u>	$\left\{ \begin{array}{l} \text{nom-CD-1} \\ [\text{ALL REFERENCES OF}] \text{identificateur-1} \\ \text{nom-de-fichier-1} \\ \text{nom-de-procédure-1} \\ \text{ALL PROCEDURES} \end{array} \right\} \dots$
-----------------------------	--

- De façon générale la section sera exécutée à chaque référence de l'objet spécifié.
- Les noms symboliques invoqués ne peuvent appartenir à une autre section déclaratives.
- Dans une section de déclaratives, on ne peut pas faire référence à une autre procédure définie hors de la section.
- 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.
 - **un nom de procédure**, la section est exécutée avant de passer dans cette procédure, ou à la suite de l'exécution d'une instruction ALTER citant ce nom.
 - **un nom de donnée**, les instructions de la procédure USE FOR DEBUGGING seront exécutées avant chaque WRITE ou REWRITE citant explicitement identificateur-1, avant chaque initialisation ou modification de la valeur d'identificateur-1 s'il est cité, dans une instruction PERFORM, après VARYING, AFTER ou UNTIL, et immédiatement après toute instruction COBOL citant explicitement identificateur-1 et modifiant son contenu.

- Si l'on a indiqué la locution `ALL REFERENCES OF`, aux effets explicités ci-dessus s'ajoute l'exécution de la procédure `USE` pour chaque `GO TO DEPENDING ON` citant identificateur-1, avant que ne s'opère la rupture de séquence.
- **Attention** : certains compilateurs nécessitent le positionnement d'une variable d'environnement (Cobol Micro Focus : `COBSW=+D`).

DEBUG-ITEM

En liaison avec les sections déclaratives `USE FOR DEBUGGING`, COBOL fournit un registre spécial qui permet le stockage automatique d'informations utiles à la mise au point des programmes. Le format implicite de ce registre est le suivant :

```
DEBUG-ITEM.  
02  DEBUG-LINE      PICTURE IS X(6).  
02  FILLER          PICTURE IS X VALUE SPACE.  
02  DEBUG-NAME      PICTURE IS X(30).  
02  FILLER          PICTURE IS X VALUE SPACE.  
02  DEBUG-SUB-1     PICTURE IS S9(4) SIGN IS LEADING SEPARATE CHARACTER.  
02  FILLER          PICTURE IS X VALUE SPACE.  
02  DEBUG-SUB-2     PICTURE IS S9(4) SIGN IS LEADING SEPARATE CHARACTER.  
02  FILLER          PICTURE IS X VALUE SPACE.  
02  DEBUG-SUB-3     PICTURE IS S9(4) SIGN IS LEADING SEPARATE CHARACTER.  
02  FILLER          PICTURE IS X VALUE SPACE.  
02  DEBUG-CONTENTS  PICTURE IS X(n).
```

- `DEBUG-LINE` : contient le numéro de ligne de l'instruction qui a appelé la section déclaratives.
- `DEBUG-NAME` : contient le nom-de-donnée qui a provoqué l'appel de la section déclaratives.
- `DEBUG-SUB-1, 2, 3` : contiennent la valeur de chacun des indices ou index, à supposer que `DEBUG-NAME` soit une table.
- `DEBUG-CONTENTS` : de manière générale, cette donnée indique le contenu d'identificateur-1 si c'est le changement de sa valeur qui a provoqué l'exécution de la procédure `USE` ; dans les autres cas, `DEBUG-CONTENTS` peut contenir :
 - "START PROGRAM" au début de l'exécution du programme.
 - "PERFORM LOOP" si la cause est une instruction `PERFORM`.
 - "FALL THROUGH" en cas de passage en séquence dans une autre procédure du programme.
 - "USE PROCEDURE" en cas d'exécution d'une procédure `USE`.
 - "SORT INPUT", "SORT OUTPUT" ou "MERGE OUTPUT" si l'exécution de la procédure est causée par une procédure d'entrée de tri, de sortie de tri ou de fusion.

☑ Remarque :

USE est en fait une instruction très puissante qui possède aussi un format de sections déclaratives spécifique aux incidents d'entrée-sortie.

```
USE AFTER STANDARD { EXCEPTION  
                   ERROR  
PROCEDURE ON { nom-de-fichier-1 [nom-de-fichier-2] ...  
              INPUT  
              OUTPUT  
              I - O  
              EXTEND  
              }
```

■ Le branchement à la section déclarative se fait essentiellement dans deux cas :

- erreur lors de l'ouverture des fichiers (INPUT, OUTPUT, I-O et EXTEND)
- fin de fichier (nom-de-fichier-1), si l'option AT END n'a pas été spécifiée pour le READ.

Illegal character in numeric field

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

La solution la plus simple consisterait à trouver une option du compilateur qui invalide la vérification de classe sur les champs numériques. Cependant les puristes vous conseilleront de procéder dans l'ordre suivant :

1. **Initialisez** toutes les variables numériques dans la WORKING-STORAGE SECTION (clause VALUE) ou dans la PROCEDURE DIVISION.
3. **Testez la numéricité** sur tous les ACCEPT de champs destinés à un traitement numérique.
4. **Décomposez** les instructions READ INTO en READ + MOVE TO si les champs concernés sont numériques.
5. **Vérifiez** les affectations et pointez les zones de format édité.
6. **Remplacez les 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 (Cobol Micro Focus : **COBSW=-F**).

Annexes

Mots réservés	105
Tableau récapitulatif des formats	109
Codes d'erreurs des entrées-sorties	111

Annexes



Mots réservés

mot réservé	sens / domaine	page
+	addition	41
-	soustraction	41
*	multiplication	41
/	division	41
**	exponentielle	41
>	supérieur	41
<	inférieur	41
=	égale	41

A

ACCEPT	entrée de données	28
ACCESS	mode d'accès fichiers	74
ADD	addition	37
ADVANCING	états impr. / DISPLAY	29
AFTER	édition / tables	49
ALL	+ lit. = constante fig.	1
ALPHABET	cf. SPECIAL NAMES	8
ALPHABETIC	test de classe	36 ; 44
ALPHANUMERIC		
ALPHANUMERIC-EDITED		
ALPHANUMERIC-LOWER		
ALPHANUMERIC-UPPER		
ALSO	EVALUATE	42
ALTER	aiguillages	47
ALTERNATE	clé secondaire fich. idx	75
AND	et logique	45
ANY	n'importe lequel	42
ARE	mot facultatif	/
AREA	zone tampon fichiers	74
AREAS	zones tampons fichiers	74
ASCENDING	clé ascendante	60 ; 89
ASSIGN	+ nom syst. de fichiers	74
AT	de AT END	80
AUTHOR	+ commentaire	6

B

BEFORE	de TEST BEFORE	49
BINARY	format binaire	23
BLANK	espace	19
BLOCK	définition de fichiers	74
BOTTOM	définition états impr.	76
BY	par	39

C

CALL	appel de sous-prog.	52
CANCEL	libère mémoire ss-prog.	54
CD	communication	/
CF	= CONTROL FOOTING	/
CH	= CONTROL HEADING	/
CHARACTER	opération sur chaînes	32
CHARACTERS	opération sur chaînes	32
CLASS		/
CLOCK-UNITS	cf. RERUN	/
CLOSE	fermeture fichier	78
COBOL		/
CODE	définition états impr.	11
CODE-SET	définition états impr.	76
COLLATING	SEQUENCE	/
COLUMN	module d'impression	/
COMMA	virgule	8
COMMON	programme contenu	56
COMMUNICATION		/
COMP	abrv. COMPUTATIONAL	23
COMPUTATIONAL	format binaire	23
COMPUTE	opér. arithmétique	41
CONFIGURATION	SECTION	7
CONTAINS	définition de fichiers	77
CONTENT		/
CONTINUE	équival NEXT SENTENCE	43
CONTROL	définition états impr.	11
CONTROLS	définition états impr.	11
CONVERTING	option de INSPECT	32
COPY	⇔ #include du C	27
CORR	abrv. CORRESPONDING	31
CORRESPONDING	affectation - MOVE	31
COUNT	opération sur chaînes	32
CURRENCY	définit signe F ou \$	8

D

DATA	DIVISION	10
DATE	date système	28
DATE-COMPILED	date de compilation	6
DATE-WRITTEN	+ commentaire	6
DAY	jour système	28
DAY-OF-WEEK	no. du jour ds semaine	28
DE	= DETAIL	/
DEBUG-CONTENTS	élém. de DEBUG-ITEM	101

DEBUG-ITEM	structure débogage	101
DEBUG-LINE		
DEBUG-NAME		
DEBUG-SUB-1	éléms. de DEBUG-ITEM	101
DEBUG-SUB-2		
DEBUG-SUB-3		
DEBUGGING	active D en 7ème col.	99
DECIMAL-POINT	séparateur décimal	8
DECLARATIVES	débogage - USE	100
DELETE	suppr. enreg. du fichier	79
DELIMITED	opération sur chaînes	34
DELIMITER	opération sur chaînes	35
DEPENDING	sauts cond./tailles var.	46 ; 59
DESCENDING	clé descendante	60 ; 91
DESTINATION	descr. communication	/
DETAIL	module d'impression	/
DISABLE	communication	/
DISPLAY	affichage de données	29
DIVIDE	division	39
DIVISION	les 4 divisions du Cobol	6
DOWN	tables - index / SET	63
DUPLICATES	clés multiples - fichiers	75
DYNAMIC	accès dynamique	75

E

EGI	communication	/
ELSE	sinon - IF	44
ENABLE	communication	/
END	de AT END	80
END-ADD		
END-CALL		
END-COMPUTE	terminateurs des	
END-DELETE	fonctions	
END-DIVIDE	associées	
END-EVALUATE		
END-IF		
END-MULTIPLY		
END-OF-PAGE	fichiers d'édition	/
END-PERFORM		
END-READ		
END-RECEIVE		
END-RETURN		
END-REWRITE	terminateurs des	
END-SEARCH	fonctions	
END-START	associées	
END-STRING		
END-SUBTRACT		
END-UNSTRING		
END-WRITE		
ENTER	appel langage différent	/
ENVIRONMENT	division	7
EOP	abrég. END-OF-PAGE	/
EQUAL	⇔ signe égale	41
ERROR	gest. des erreurs - USE	100
ESI	communication	/
EVERY	op. chaînes/trait. err.	100
EXCEPTION	erreur / exception	102
EXIT	paragraphe vide	51
EXTEND	mode ajout - fichiers	80
EXTERNAL	déf. fichiers en global	55

F

FALSE	EVALUATE	42
FD	File-Description	76
FILE	de FILE STATUS	111
FILE-CONTROL	déclarat. fich. - SELECT	8
FILLER	description données	19
FINAL	définition états impr.	11
FIRST	opération sur chaînes	32
FOOTING	fichiers d'édition	11 ; 76
FOR	USE	100
FROM	à partir de - WRITE	82

G

GENERATE		/
GIVING	affectations	37
GLOBAL	programmes contenus	56
GO	sauts inconditionnels	46
GREATER	plus grand	41
GROUP		/

H

HEADING	description états impr.	11
HIGH-VALUE	tous les bits à 1	1
HIGH-VALUES	tous les bits à 1	1

I

I-O	INPUT-OUTPUT	80
I-O-CONTROL	→ instruction RERUN	9
IDENTIFICATION	première division	6
IF	condition	44
IN	qualification	16
INDEX	déf. tables - index	62
INDEXED	déf. tables - index	62
INDICATE	module d'impression	/
INITIAL	type autom. prog. cont.	56
INITIALIZE	initialise zones données	36
INITIATE	initialise traitement état	/
INPUT	lecture uniquement	80
INPUT-OUTPUT	lecture - écriture	80
INSPECT	opération sur chaînes	32
INSTALLATION	+ commentaire	6
INTO	affectation - division	39
INVALID	KEY / clé incorrecte	79
IS	mot facultatif	/

J

JUST	abrég. de JUSTIFIED	20
JUSTIFIED	représ. interne donnée	20

K

KEY	clé - fichiers	79
-----	----------------	----

L

LABEL	déf. fichiers - RECORD	77
LAST	de LAST DETAIL	11
LEADING	opération sur chaînes	21; 32
LEFT	voir JUSTIFIED	20
LENGTH		/
LESS	moins (que)	41
LIMIT	définition d'état impr.	11
LIMITS	définition d'état impr.	11
LINAGE		/
LINAGE-COUNTER	fichiers d'édition	76
LINE		11
LINE-COUNTER		/
LINES	édition	11
LINKAGE	section	53
LOCK	verrouillage fichier	78
LOW-VALUE	tous les bits à 0	1
LOW-VALUES	tous les bits à 0	1

M

MEMORY	ds. OBJECT COMPUTER	7
MERGE	fusion de fichiers triés	92
MESSAGE	communication	/
MODE	DEBUGGING	99
MODULES	ds. OBJECT COMPUTER	7
MOVE	affectation	30
MULTIPLE	cf. RERUN	9
MULTIPLY	multiplication	39

N

NATIVE	ALPHABET IS	7
NEGATIVE	test de négativité	44
NEXT	lect. prochain enreg.	80
NO	ADVANCING	29
NOT	non logique	44
NUMBER	COLUMN NUMBER	/
NUMERIC-EDITED	test de classe	44
NUMERIC	test de classe	44

O

OBJECT-COMPUTER	+ commentaire	6
OCCURS	définition de tables	59
OF	qualifications	16
OFF	option de SET	/
OMITTED	LABEL RECORD IS	77
ON	DEPENDING - USE	...
OPEN	ouverture fichiers	80
OPTIONAL	fich. peut ne pas exister	74
OR	ou logique	44
ORDER		/
ORGANIZATION	organisation fichiers	74
OTHER	EVALUATE - autre cas	42
OUTPUT	écriture uniquement	80
OVERFLOW	dépassement de zone	...

P

PACKED-DECIMAL	DCB \Leftrightarrow COMP-3	23
PADDING	remplissage enreg. vide	75
PAGE	module d'impression	/
PAGE-COUNTER		/
PERFORM	appel de procédures	48-52
PF	abrég. PAGE FOOTING	/
PH	abrég. PAGE HEADING	/
PIC	abrég. de PICTURE	14
PICTURE	définition de données	14
PLUS		/
POINTER	opération sur chaînes	34
POSITION		/
POSITIVE	test de positivité	44
PRINTING		/
PROCEDURE	USE	100
PROCEDURES		
PROCEED	ALTER	47
PROGRAM	EXIT	55
PROGRAM-ID	identification du prog.	6
PURGE		/

Q

QUEUE		/
QUOTE	guillemet	1
QUOTES	guillemets	1

R

RANDOM	accès direct fich. idx rel	75
RD	Report Description	11
READ	lecture enreg. fichier	80
RECEIVE	communication	/
RECORD	LABEL	75; 77
RECORDS	définition de fichiers	75; 77
REDEFINES	redéfinition données	17
REEL	bande magnétique	78
REFERENCE	BY REFERENCE	52
REFERENCES	débogage avec USE	100
RELATIVE	organisation relative	75
RELEASE	tri-fusion	91
REMAINDER	reste de la division	40
REMOVAL	CLOSE FOR (pr suppr)	78
RENAMES	définition données	17
REPLACE		27
REPLACING	opération sur chaînes	33; 36
REPORT	SECTION / RH / RF	11
REPORTING		/
REPORTS		/
RERUN	déf. pts. de reprise fich.	9
RESERVE	déf. nbre buffers E/S	75
RESET	module d'impression	/
RETURN	lect. fich. interm. de tri	91
REVERSED	lect. arrière des bandes	80
REWIND	rembobinage des bandes	78
REWRITE	réécriture d'un enreg.	81
RF	REPORT FOOTING	/
RH	REPORT HEADING	/

RIGHT	JUSTIFIED	20
ROUNDED	pour les arrondis	37
RUN	de STOP RUN	47

S

SAME	AREA, zone tampon	9
SD	Sortfile-Definition	91
SEARCH	recherche dans tables	64
SECTION		6
SECURITY	+ commentaire (obs.)	6
SEGMENT	gestion des overlays	/
SEGMENT-LIMIT	gestion des overlays	/
SELECT	définition de fichiers	74
SEND	communication	/
SENTENCE	de NEXT sentence	44
SEPARATE	déf. position signe	21
SEQUENCE	COLLATING	7
SEQUENTIAL	mode séquentiel	74
SET	déf. val. pour index	18 ; 63
SIGN	définition signe	21
SIZE	opér. chaînes/gest. err.	...
SORT	tri Cobol	89
SORT-MERGE	tri-fusion (?)	/
SOURCE	module d'impression	/
SOURCE-COMPUTER	+ commentaire	6
SPACE	const. fig. espace	1
SPACES	const. fig. espaces	1
SPECIAL-NAMES	déf. noms utilisateur	8
STANDARD	LABEL RECORD IS	77
STANDARD-1	ALPHABET IS	7
STANDARD-2		
START	positionnement fich.	81
STATUS	état d'un fichier	111
STOP	fin du programme	47
STRING	concaténation chaîne	34
SUB-QUEUE-1		/
SUB-QUEUE-2		/
SUB-QUEUE-3		/
SUBTRACT	soustraction	38
SUM	module d'impression	/
SUPPRESS		/
SYMBOLIC		/
SYNC	abrv.SYNCHRONISED	22
SYNCHRONIZED	repr. interne donnée	22

T

TABLE		/
-------	--	---

TALLY	champ op. chaînes	/
TALLYING	opération sur chaînes	32
TAPE		/
TERMINAL		/
TERMINATE	termine un état impr.	/
TEST	AFTER / BEFORE	49
TEXT		/
THAN	GREATER / LESS	41
THEN	mot optionnel avec IF	44
THROUGH	déf. plage de valeurs	18 ; 48
THRU	ou procédures	
TIMES	boucles avec PERFORM	48
TO	vers, par ex. MOVE TO	30
TOP	fichiers d'édition	/
TRAILING	déf. position signe	21
TRUE	EVALUATE	42
TYPE	module d'impression	/

U

UNIT	CLOSE	60
UNSTRING	éclatement chaînes	34
UNTIL	ici : tant que	49
UP	incrémenter idx - SET	63
UPON	DISPLAY	29
USAGE	représ. interne données	23
USE	gestion des erreurs	100
USING	transmission données	53

V

VALUE	initialisation champ	24
VALUES	initialisation champs	24
VARYING	boucles avec PERFORM	49

W

WHEN	EVALUATE / SEARCH	42 ; 64
WITH	attribut	28
WORDS		/
WORKING-STORAGE	SECTION	12
WRITE	écriture enreg. fich.	82

Z

ZERO	un zéro - const. fig.	1
ZEROES	beaucoup de zéros	1
ZEROS	idem.	1

Tableau récapitulatif des formats

Caractère	Élément d'origine		Élément 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	\$□□□□□□□□

Caractère	Élément d'origine		Élément d'édition	
	PICTURE	CONTENU	PICTURE	CONTENU
+ - 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	\$\$\$99	□□\$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

Le tableau suivant donne la valeur du FILE-STATUS qui, rappelons-le, est au format XX.

X	X	Seq	Idx	Rel	Signification
0	0				Entrée-Sortie réussie.
0	2				Entrée-Sortie réussie, mais le système a détecté une clé dupliquée.
0	4				Lecture faite, mais l'enregistrement n'a pas la longueur déclarée dans le programme.
0	5				Tentative d'OPEN d'un fichier OPTIONAL qui n'est pas présent. En mode I-O ou EXTEND le fichier est créé.
0	7				A l'occasion d'une instruction CLOSE ou OPEN avec une des clauses NO REWIND, REEL/UNIT ou REMOVAL, on constate que le fichier n'est pas sur bande magnétique.
1	0				Fin de fichier en lecture.
1	4				Pour les fichiers relatifs, le nombre d'enregistrements du fichier dépasse la capacité de la clé relative.
2	1				Problème de séquence en traitement séquentiel. Par exemple, on a lu un enregistrement et on veut le réécrire avec une clé différente.
2	2				On veut écrire un enregistrement alors qu'il en existe déjà un avec une clé identique.
2	3				Tentative d'accès à un enregistrement qui n'existe pas.
2	4				Tentative de WRITE hors des limites d'un fichier relatif ou indexé.
3	0				Erreur permanente.
3	5				OPEN sur un fichier, non OPTIONAL, qui n'est pas présent.
3	7				OPEN sur un fichier supposé en accès direct qui, en fait, ne l'est pas.
3	8				OPEN sur un fichier au préalable fermé par CLOSE WITH LOCK.
3	9				OPEN impossible suite à un conflit entre la description Cobol et la réalité.
4	1				Tentative d'OPEN sur un fichier déjà ouvert.
4	2				Tentative de CLOSE sur un fichier non ouvert.
4	3				Tentative de DELETE ou REWRITE sur un enregistrement qui n'a pu être lu par un READ précédent.
4	4				Tentative de WRITE ou de REWRITE d'un enregistrement dont la dimension ne correspond pas à la description de fichier.
4	6				Tentative de lecture séquentielle d'un enregistrement alors que la lecture précédente n'a pas réussi.
4	7				Tentative de READ ou START (fichiers relatifs ou indexés) sur un fichier non ouvert en INPUT ou en I-O.
4	8				WRITE sur un fichier non ouvert en OUTPUT, en I-O ou en EXTEND.
4	9				Tentative de DELETE ou REWRITE sur un fichier non ouvert en I-O.
9	1				Fichier endommagé.
9	X				Erreur irréparable - fichier endommagé.

Index alphabétique

A

ACCEPT, 28
ACCESS, 74
ADD, 37
AFTER, 49
Aide
 à l'écriture, 27
 à la mise au point, 99
ALL, 1
ALSO, 42
ALTER, 47
ALTERNATE, 75
ASCENDING
 OCCURS, 60
 SORT, 91
AT END, 78

B

BINARY, 23
BLANK WHEN ZERO, 19
BLOCK, 77
Branchements
 conditionnels, 46
 inconditionnels, 46
 modifiés (ALTER), 47

C

Calculs, 37
CALL, 52
CANCEL, 54
Caractères
 autorisés, 1
 de formatage, 14
CLOSE, 78
COMMON, 56
COMPUTATIONAL, 23
COMPUTE, 41
Concaténation de chaînes, 34
CONFIGURATION, 7
Constantes
 alphanumériques, 3
 figuratives, 1
 numériques, 2
CONTAINS, 77
CONTINUE, 43
Conventions
 de codage, 3
 typographiques, i
CONVERTING, 33
COPY, 27

D

DATA DIVISION, 10
Débogage, 99
DEBUG-ITEM, 101
DEBUGGING, 99; 100
DECLARATIVES, 100
DELETE, 79
DELIMITED
 STRING, 34
 UNSTRING, 35
DEPENDING
 GO TO, 46
 OCCURS, 59
DESCENDING
 OCCURS, 60
 SORT, 91
DISPLAY, 29
DIVIDE, 39
Données
 affectation, 16; 30
 classes, 15
 description, 12
 qualification, 16
DUPLICATES, 75
DYNAMIC, 75

E

Enregistrements
 écriture, 82
 positionnement, 81
 réécriture, 81
 structure (FD), 76
 suppression, 79
ENTRY, 55
ENVIRONMENT DIVISION, 7
Erreurs
 codes, 111
 gestion des, 99-102
EVALUATE, 42
EXIT, 51
EXIT PROGRAM, 55
EXTERNAL, 55

F

FD, 76
Fichiers
 accès, 69; 82
 description, 74
 fermeture, 78
 lecture, 80
 méthode du hashing, 71
 modifications, 72
 mouvement, 83

organisation, 69
ouverture, 82
réorganisation, 73
FILE STATUS, 111
FILLER, 19
Formats
internes, 23
tableau récapitulatif, 109

G

GIVING, 37
GLOBAL, 56
GO TO, 46
GOBACK, 55

H

Hash-code, 71
Historique, ii

I

IDENTIFICATION DIVISION, 6
IF, 44
Illegal character in numeric field, 102
IN, 16
Index, 62
Indices, 61
INITIAL, 56
INITIALIZE, 36
INSPECT, 32
INVALID KEY, 79

J

JUSTIFIED-RIGHT, 20

L

LABEL, 77
LEADING
REPLACING, 33
SIGN, 21
TALLYING, 32
Ligne (poursuite de), 4
LINKAGE SECTION, 53
Littéraux
alphanumériques, 3
numériques, 2
LOCK, 78

M

MERGE, 92
Modèles d'écriture, 5
Modules

appel, 52
sortie, 55

Mots

réservés, 1
réservés (liste des), 105-8
utilisateur, 2

MOVE, 30

MOVE CORRESPONDING, 31

MULTIPLY, 39

N

Noms-de-condition, 18
Notation par référence, 28

O

OCCURS, 59
OF, 16
OMITTED, 77
OPEN, 80
Opérateurs
arithmétiques, 41
de comparaison, 44

P

PACKED-DECIMAL, 23
PADDING, 75
PERFORM, 48-52
PICTURE, 14
POINTER
STRING, 34
UNSTRING, 35
PROCEDURE DIVISION, 12
Procédures
appel, 48-52
sortie, 51
Programmes contenus, 56
PSEUDO-TEXTE, 27

Q

Qualification, 16
QUOTE, 3

R

RANDOM, 75
READ, 80
Recherches
dans un fichier, 81
dans un tableau, 64
RECORD, 75; 77
REDEFINES, 17
REEL, 78

Référence

- notation par, 28
- transmission par, 52

RELATIVE, 75

RELEASE, 91

REMAINDER, 40

REMOVAL, 78

RENAMES, 17

REPLACE, 27

REPLACING

- INITIALIZE, 36

- INSPECT, 33

RERUN, 9

RESERVE, 75

RETURN, 91

REVERSED, 80

REWIND, 78

REWRITE, 81

ROUNDED, 37

S

Sauts

- conditionnels, 46
- inconditionnels, 46

SD, 91

SEARCH, 64

SELECT, 74

SEPARATE, 21

SET, 18; 63

SIGN, 21

SORT, 89

SPACE, 3

SPECIAL NAMES, 8

STANDARD, 77

START, 81

STOP, 47

STRING, 34

Structure

- d'un programme, 6
- de fichiers, 76

SUBTRACT, 38

Sujets implicites, 45

SYNCHRONIZED, 22

T

Tableaux, 59-66

TALLYING

- INSPECT, 32

- UNSTRING, 35

TEST, 49

TIMES, 48

TRACE, 99

TRAILING, 21

Tri-fusion, 89

U

UNIT, 78

UNSTRING, 34

UNTIL, 49

USAGE, 23

USE, 100

USING, 53

V

VALUE, 24

VARYING, 49

W

WORKING-STORAGE SECTION, 12

WRITE, 82

Z

ZERO, 1

Bibliographie

Nous vous recommandons tout particulièrement ces excellents ouvrages :

COBOL A.N.S. 85 avec exercices et corrigés, Christian Bonnin ; ed. Eyrolles

COBOL Perfectionnement et Pratique, M.Koutchouk ; ed. Masson