COBOL

2007

J-M. MESKENS

1. INTRODUCTION

COBOL, *'Common Business Oriented Language '* est un langage orienté vers les affaires. Ce langage a été mis au point en 1959 à la demande du gouvernement américain dans le but d'harmoniser les systèmes des différentes administrations.

Ce langage est encore fort répandu car à peu près 70% des applications commerciales sont écrites en COBOL. Cette grande utilisation peut être expliquée par la grande portabilité de ce langage et ses différentes remises à niveau. Des différences existent entre chaque compilateur COBOL. Nous verrons principalement le compilateur Cobol IBM-AS400 et celui de Microsoft.

1. Principes de base

Un programme Cobol s'écrit sur une ligne de 80 colonnes.
Les colonnes 1 à 6 représentent la numérotation des lignes et des pages.
La colonne 7 est utilisée pour indiquer une ligne de continuation (code -) ou pour indiquer un commentaire (code *).
Les colonnes 8 à 72 sont utilisées pour l'écriture du programme.
Les colonnes 73 à 80 sont utilisées pour l'identification mais non contrôlées.

Les noms de Divisions, Sections, Paragraphes sont écrits à partir de la marge A (Colonne 8), les instructions élémentaires sont écrites à partir de la marge B (colonne 12).

2. Structure du programme

Un programme Cobol est toujours composé des 4 même parties appelées **Divisions.** Ces divisions doivent se succéder dans un ordre bien précis.

IDENTIFICATION DIVISION Cette partie permet de spécifier le nom du programme et celui du programmeur

ENVIRONMENT DIVISION décrit le matériel utilisé et les périphériques supportant-les fichiers

DATA DIVISION décrit toutes les données indépendantes ou structurées

PROCEDURE DIVISION contient les instructions exécutables

3. Conventions d'écriture

Le programme peut être écrit aussi bien en minuscule qu'en majuscule, mais dans ce cas la recherche d'un nom via l'éditeur sera incomplète.
Les caractères utiles sont : 0 à 9 de A à Z ou a – z
L'espace ou Blanc
Les signes + / * - < > = $, . ' ( )

Tout élément du programme sera précédé et suivi de un ou plusieurs espaces, sauf pour les (qui ne sont pas suivies de blancs.
Les caractères de ponctuation . et , ne sont pas précédés d'espace, de même que la).

Les noms symboliques

–1 à 30 caractères
–lettres, chiffres et -
–pas de - au début ou à la fin
–aucun espace inclus
–premier caractère alphabétique
–ne peut être identique à un mot réservé

Ils comportent un maximum de 30 caractères choisis parmi A à Z et 0 à 9 et le trait d'union.
Un nom doit commencer par une lettre, ne pas comporter un caractère blanc et être suivi et précédé d'un espace.

- Qualification : il peut arriver qu'un même nom soit utilisé pour désigner deux zones différentes. On distingue ces 2 zones en qualifiant le nom par IN ou OF.
- Exemple : NOM OF FCLIENT et NOM OF FOURNISSEUR

- Nom de paragraphe : il s'agit d'un nom désignant l'adresse d'une instruction du programme. Ce nom s'écrit en marge A et est suivi d'un point.

- Nom-condition : il désigne un état d'une variable associée à une instruction IF.

Les Littéraux

Un littéral est une constante non identifiée par un nom. Il en existe de 2 types, les numériques et les alphanumériques.

- Les numériques : il s'agit d'un nombre de **18 chiffres** maximum. Il s'écrit seul ou avec un point décimal si nécessaire et est précédé ou non d'un + ou – (+ par défaut).
- Exemple : -125.66 +0.55 555 258746

- Les alphanumériques : il s'agit d'une combinaison de caractères inclus entre apostrophes.
- Exemple : ' *** Ceci est un texte de 120 ou 255 caractères…../++ '

Si un tel littéral est trop long pour être contenu sur une seule ligne, on peut le continuer sur la ligne suivante en plaçant un trait d'union en colonne 7
une ' en marge B
Attention aux espaces se trouvant sur la première ligne ils sont compris dans l'ensemble du littéral.

Les constantes Figuratives

Ce sont des constantes désignées par des Noms Réservés et ayant une fonction précise.

ZERO (S)
SPACE SPACES
HIGH-VALUE LOW-VALUE
ALL

2. IDENTIFICATION DIVISION

Cette division sert à identifier le nom du programme et fournit des informations relatives au nom du programmeur, de la date d'écriture du programme et aux remarques éventuelles.

Forme Générale

```
A  B
IDENTIFICATION              DIVISION.

PROGRAM-ID. Program4.
AUTHOR. jean-marie meskens.
DATE-WRITTEN. dimanche 20 août 2006 9:46:52.
REMARKS.
```

Seules les 2 premières instructions sont obligatoires.

PROGRAM-ID. Program4. indique le nom du programme, cette clause est toujours présente car elle identifie le programme. Ce nom est limité à 8 caractères alphanumériques dont le premier doit être alphabétique.

AUTHOR. MOI.	Désigne le nom du programeur.
DATE-WRITTEN	permet d'identifier les différentes versions compilées de votre Programme.

3. ENVIRONMENT DIVISION

Cette division permet de définir le type d'ordinateur pour lequel est écrit le programme, ainsi que les périphériques nécessaires pour supporter les fichiers.

```
A  B
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-AS400.
                 Ou        IBM-PC.
OBJECT-COMPUTER.  IBM-AS400.
                  Ou       IBM-PC.
SPECIAL-NAMES. Liste des noms symboliques spéciaux.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
       Options
I-O-CONTROL.
        Options
```

Les Special-names

CURRENCY-SIGN IS caractère par défaut le $ mais un autre caractère peut être utilisé

DECIMAL-POINT IS COMMA la Virgule indique la séparation des unités et des décimales.

Nom-réservé IS nom-mnémonique cette clause s'adresse aux instructions d'entrée-sortie DISPLAY et ACCEPT.

L'INPUT-OUTPUT SECTION

Fait le lien entre les fichiers du programme et les fichiers sur les périphériques (Disque, bande, imprimante). C'est ici qu'on indique si le fichier est séquentiel ou en accès direct.

En FILE-CONTROL, on établit le lien entre chacun des noms de fichiers désignés dans la DATA DIVISION et les noms de fichiers externes.

Forme générale

```
A  B
FILE-CONTROL.
   SELECT FOURNISSEUR ASSIGN TO …….
```

FOURNISSEUR est le nom indiqué en FILE SECTION dans la clause FD Nom-de-fichier

Les options

FILE-CONTROL.
SELECT [optional] nom-du-fichier assign to ……
ORGANIZATION IS ……
ACCES MODE IS…….
Clause KEY…
FILE STATUS IS ……..

1. ORGANIZATION IS LINE SEQUENTIAL
SEQUENTIAL
INDEXED
RELATIVE

2. ACCESS MODE IS SEQUENTIAL
RANDOM
DYNAMIC

3. KEY
RELATIVE KEY IS nom-de-donnée (indexé relative)

RECORD KEY IS nom-de-donnée1 (indexé séquentiel)
ALTERNATE RECORD KEY IS nom-de-donnée2
[WITH DUPLICATES]

4.FILE-STATUS IS SKF1

Remarques

Au niveau de l'organisation :
Line-sequential pour les fichiers crées par EDITEUR.
Relative pour les fichiers à accès directe au moyen d'une clé numérique qui indique la position de L'enregistrement dans le fichier.
Indexed, les enregistrements sont les uns derrières les autres et les clés d'accès se trouvent dans un Second fichier.

Au niveau de l'accès :
Sequential le fichier sera lu enregistrement par enregistrement du début à la fin.
Random le fichier est lu directement au moyen d'une clé, l'enregistrement recherché est donné Immédiatement.
Dynamic le fichier peut être lu comme le sequential ou comme le random durant toute la durée du programme.

Au niveau des clés :
La définition de clés n'est valable que pour les fichiers NON séquentiels.

Le *FILE STATUS* permet de récupérer sous forme de code numérique le résultat de l'opération effectuée sur ce fichier. Pour cela optional doit être indiqué.

Utilisation d'un fichier de type texte

```
INPUT-OUTPUT SECTION.
      FILE-CONTROL.
         SELECT OPTIONAL fdisk1
               ASSIGN TO disk "c:\temp\monfichier.txt"
                Organization is Line Sequential
                FILE STATUS IS status-variable.
     *
          Select impr assign to print "-p spooler".

     ****************************************************************
      DATA DIVISION.
      FILE SECTION.
      FD  fdisk1.
      01  maligne pic x(80).
```

(LEN : Donne la longueur d'une chaîne)

*** si vous ne pouvez pas indiquer directement le nom du fichier vous pouvez utiliser cette méthode

```
      FD  fdisk1  VALUE OF FILE-ID  LENOM_DE_MONFICHIER.
      01  maligne pic x(80).

      FD  Impr.
      01  ligne pic x(80).

01 LENOM_DE_MONFICHIER PIC X(100).
```

Pour un fichier de données

```
ENVIRONMENT DIVISION.
      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
         SELECT optional FOUT
               ASSIGN TO "c:\temp\fout.dta"
               FILE STATUS is sk.

     ****************************************************************
      DATA DIVISION.
      FILE SECTION.
      FD  Fout.
      01  EOUT pic x(20).
```

4. DATA DIVISION

La DATA DIVISION est organisée en 4 sections.

La FILE SECTION qui définit les fichiers en regroupant les instructions de gestion des entrées sorties et les descriptions des zones de mémoire où sont transmis les enregistrements des fichiers.

La WORKING-STORAGE SECTION où sera décrit chaque variable de travail.

La LINKAGE SECTION qui permet de déclarer les zones de mémoire utilisées par plusieurs programmes.

La SCREEN SECTION pour la gestion du GUI

4.1 FILE SECTION

La FILE SECTION contient une description de fichier ainsi qu'une ou plusieurs descriptions d'enregistrement pour chaque fichier utilisé par le programme.

DECLARATION DU FICHIER

FD file-name BLOCK CONTAINS XXX RECORDS (N° de records à écrire au même temps)

RECORD CONTAINS NNN CHARACTERS (calcul automatique)
(N° d'octets total du record)

LABEL RECORD IS {STANDARD} (par défaut /// Ne faut pas les préciser)
{OMITTED }

VALUE OF { FILE-ID } IS id-name
DATA { RECORD IS } {record-name} ...
{ RECORDS ARE }

Le nom est celui que l'on trouve dans SELECT
XXX indique le nombre d'enregistrements contenu dans un bloc
NNN donne la longueur en octet d'un enregistrement
Label indique s'il s'agit d'enregistrements sur imprimante (omitted) ou autre
Value permet de donner le nom du fichier sur disque
Data Record est utilisé pour indiquer le nom de l'enregistrement qui décrira le niveau 01, comme il peut y avoir des enregistrements de modèles différents on peut les renseigner ici.

Description de l'enregistrement

Cette partie donne :

–le type et la longueur des champs qui le composent
–l'ordre de ces champs

```
01 nom-enregistrement.
        05 nom-élément-simple ….
        05 nom-item-groupe.
                10 nom-élément-simple ….
                10 nom-élément-simple ….
        05 nom-élément-simple ….
```

Dans la description de l'enregistrement :
–les données sont définies de façon hiérarchique (par niveau).
–Le nom de l'enregistrement porte le numéro 01
–les numéros 02 à 49 peuvent être utilisés pour identifier les items dans l'enregistrement.

–Les items qui composent un item de groupe doivent porter un numéro plus grand que ce dernier.
–Tous les items de même niveau doivent porter le même numéro.

Le format général d'une description de zone de données est ：01 débute dans la marge A, les autres numéros dans la marge B.

4.2 WORKING-STORAGE SECTION

Cette section décrit les variables de travail. Les variables COBOL peuvent être classées en 2 catégories : les données élementaires et les groupes de données.

Les variables élémenantaires

Niveau nom-de-donnée { PICTURE / PIC } description Usage [VALUE IS littéral]

Niveau : les données élémentaires isolées ont soit le niveau 77 soit 01 (marge A), en soi il n'y a aucune différence, seulement les 77 doivent être déclarées en tête de la WSS. Il n'y a pas de 77 en File Section.

Nom de la donnée : c'est le nom par lequel le programmeur pourra référencer la variable. Ils doivent être différents des mots réservés et univoques.

PICTURE

identifie le type de donnée
indique la dimension (grandeur) de l'item
le type peut être alphanumérique (X) numérique (9) alphabétique (A) numérique édité (9,Z,+, -, …)

01 CODE-PROVINCE PIC XXX.
05 PRENOM PIC XXXXXXXXXX.
10 SALAIRE PIC 9999999.
15 INITIALE PIC A.
Pour éviter les répétitions, on peut utiliser la forme suivante

05 PRENOM PIC X(10).
15 SALAIRE PIC 9(7).

La clause PICTURE :
– est présente pour les éléments SIMPLES seulement.
– Peut débuter n'importe où dans la marge B.
– la chaîne de caractère décrivant l'item doit être précédée d'au moins un espace.
– Il ne doit pas y avoir d'espace dans la chaîne de caractères.

Les items numériques (PICTURE 9) sont les seuls qui peuvent être utilisés pour les opérations arithmétiques.
Lorsqu'une valeur décimale est lue dans un fichier d'entrée, le point décimal n'apparaît pas mais sa présence est implicite.

Ex.
12500 représente 125.00
Ceci est indiqué en plaçant le symbole V à l'endroit où doit être le point.
Ex.
05 MONTANT PIC 999V99

USAGE

identifie le type de représentation interne qui est utilisé pour enregistrer l'information. Les principaux types sont :

DISPLAY
BINARY
PACKED-DECIMAL

VALUE

Il est possible d'initialiser la donnée, le littéral est une valeur constante. Cette initialisation ne se fait qu'une fois lors du lancement du programme. Cette valeur doit être conforme à la descriptions qui a été faite.

Si Value est omis, la valeur initiale est **INDETERMINEE.**

77 TAUX PIC 99 VALUE 0.
01 CAPITAL PIC 999999 VALUE ZERO.
01 REPONSE PIC XXX VALUE 'NON'.

Dans le cas des PIC X ou A attention à la longueur, la chaîne est toujours mise entre " " , le mot ALL peut être utilisé et dans ce cas il y aura n occurrence du caractère dans la chaîne.

77 CHAINE PIC XXXXXXXX VALUE ALL 'N'.
sans ALL un seul N suivi de blancs serait placé dans chaine.

4.3 Les TYPES DE DONNEES

`A pour Alphabétique`

Ne peut contenir que des chaînes de caractères issues des lettres "A" à "Z" ou "a" à "z" et le caractère blanc.
Un caractère occupe 1 octet en mémoire.

`X pour Alphanumérique`

Une chaîne de ce type peut contenir n'importe quel caractère. Un caractère occupe 1 octet en mémoire.

Dans les 2 cas :

si le texte du value est < à la taille du Pic un complémént à blanc est effectué.
si le texte du value est > à la taille du Pic une erreur sera donnée à la compilation
si All est utilisé le caractère sera transféré dans toutes les positions de la variable
si Space est utilisé toute la donnée sera mise à blanc
les données sont cadrées à gauche

`9 pour Numérique`

DISPLAY
BINARY
PACKED-DECIMAL

Les données numériques sont représentées par les symboles **9 S** et **V** et selon l'usage utilisé elles seront DISPLAY BINARY ou PACKED-DECIMAL.
Les données sont cadrées à droite sur la virgulle virtuelle. Si la virgule n'est pas présente elles est supposée être à l'extrême droite. Lors d'un MOVE la donnée sera tronquée à gauche si la zone réceptrice est plus petite que la zone émettrice.
Le transfert de 1000 dans une PIC 999 donne comme résultat 000. Si la réceptrice est plus grande la donnée est complétée par des zéros à gauche

La donnée aura :

Autant de chiffres que la description contiendra de 9.
Un signe suivant la présence ou non du S en début de la description
Un point décimal à l'emplacement défini par V

En mode **DISPLAY**

La donnée occupera autant d'octects qu'il y a de 9 et avec un maximum de 18.

Le signe est superposé au dernier chiffre, dans les 4 bits de gauche de l'octet le plus à droite.
Le point décimal V est implicite et n'occupe aucune place en mémoire.

En mode **BINARY**

Le signe se trouve dans le bit de gauche. Comme les variables sont binaires elles sont normalisées à 2, 4, 8 octets.

Pour Pic 9 à Pic 9999 on réserve en mémoire 2 octets PIC 9999 BINARY
Pour Pic 9(5) à Pic 9(9) on réserve en mémoire 4 octets
Pour Pic 9(10) à Pic 9(18) on réserve en mémoire 8 octets

En mode **PACKED-DECIMAL**

Ce mode caractérise les numériques en décimal condensé avec 2 chiffres par octet. Le signe se trouve dans les 4 bits de droite de l'octet le plus à droite. Ce mode est particulièrement adapté à l'écriture sur les supports externes.
La longueur en octet est calculée de la manière suivante : nombre de 9 / 2 + 1

Soit VARA PIC 9(7) PACKED-DECIMAL => 7/2 + 1 = 4 octets

4.4 PICTURES D'ÉDITION

Lorsque l'information est entreposée sur un support magnétique les caractères d'édition ne sont pas inclus.
Certaines fonctions d'édition doivent donc être accomplies quand ces données sont imprimées.
Ceci est accompli en utilisant des caractères d'édition dans la PICTURE des items qui doivent être imprimés

Les fonctions d'édition les plus courantes sont :

- Suppression des zéros non-significatifs
- Impression du point décimal
- Impression des virgules
- Impression du signe de dollar
- Impression d'astérisques en guise de protection
- Impression d'un signe (+ ou -)

Le symbole Z est utilisé pour supprimer l'impression des zéros non-significatifs. Ce symbole peut apparaître plus d'une fois dans un PICTURE.
Le Z a le même effet que le 9, sauf qu'une valeur de 0 qui n'est pas significative est remplacée par un espace.

```
01 E-ITEM     PIC 9999.
01 S-ITEM     PIC ZZZ9.
MOVE E-ITEM TO S-ITEM
                E-ITEM          S-ITEM
                0014            □□14
                0564            □564
```

0000 □□□0

Le symbole **.** est utilisé pour indiquer un point décimal. Un seul point peut apparaître dans un nombre.

```
01 E-ITEM    PIC 999V99.
01 S-ITEM    PIC 999.99.
MOVE E-ITEM TO S-ITEM
             E-ITEM          S-ITEM
             12345           123.45
             00825           008.25
             00005           000.05
```

```
01 E-ITEM    PIC 999V99.
01 S-ITEM    PIC ZZZ.99.
MOVE E-ITEM TO S-ITEM
             E-ITEM          S-ITEM
             12345           123.45
             00825           □□8.25
             00005           □□□.05
```

L'impression d'une virgule est réalisée en plaçant le symbole **,** à l'endroit désiré dans le PICTURE. Il peut y avoir plus d'une virgule.

```
01 E-ITEM    PIC 9(8)V99.
01 S-ITEM    PIC 99,999,999.99.
MOVE E-ITEM TO S-ITEM
             E-ITEM          S-ITEM
             1289344506      12,893,445.06
             0000000835      00,000,008.35
```

Si le caractère d'édition Z est utilisé avec la virgule, il supprime l'impression des virgules qui sont suivies d'un zéro non-significatif.

```
01 E-ITEM    PIC 9(8)V99.
01 S-ITEM    PIC ZZ,ZZZ,ZZ9.99.
MOVE E-ITEM TO S-ITEM
             E-ITEM          S-ITEM
        1289344506           12,893,445.06
        0000000835           □□□□□□□□□8.35
```

Le symbole ***** est utilisé de la même façon que le Z, sauf que les zéros non-significatifs sont remplacés par des astérisques.

Pour que COBOL distingue entre un nombre positif et un nombre négatif, le programmeur doit indiquer la présence d'un signe en utilisant le symbole S.

```
01 E-ITEM    PIC 9999.
01 S-ITEM    PIC S999.
MOVE -37 TO ITEM1, ITEM2.
             ITEM1      ITEM2
             0037       -037
```

L'utilisation du S indique la présence d'un signe mais celui-ci ne sera pas automatiquement imprimé. Il faut pour cela utiliser le symbole +.

```
01 E-ITEM     PIC S999.
01 S-ITEM     PIC +999.
MOVE E-ITEM TO S-ITEM.
```

E-ITEM	S-ITEM
+123	+123
-014	-014

Le symbole + peut être utilisé au début ou à la fin du PICTURE. Il peut être employé avec les autres caractères d'édition.

```
01 E-ITEM     PIC S9(5)V99.
01 S-ITEM     PIC ZZ,ZZZ.99+.
MOVE E-ITEM TO S-ITEM.
```

E-ITEM	S-ITEM
+3490500	34,905.00+
- 0150000	1,500.00 –

Si on désire que le signe ne soit imprimé que si le nombre est négatif, on peut utiliser le symbole - .

```
01 E-ITEM     PIC S9(5)V99.
01 S-ITEM     PIC -ZZ,ZZZ.99.
MOVE E-ITEM TO S-ITEM.
```

E-ITEM	S-ITEM
+3490500	34,905.00
- 0150000	- 1,500.00

4.5 SYMBOLES FLOTTANTS

Pour créer un PICTURE qui contient un symbole flottant, on remplace chaque **Z** par le symbole qui doit précéder le nombre (ceux-ci peuvent être * + - Z)

```
01 E-ITEM     PIC 9(8)V99.
01 S-ITEM     PIC +++,+++,++9.99.
MOVE E-ITEM TO S-ITEM.
```

E-ITEM	S-ITEM
1289344506	+12,893,445.06
0000000835	+8.35
0002565700	+25,657.00

Il est à noter que le premier + - ne prend pas la place d'un caractère numérique. Donc il faut compter une position de plus afin de garder l'intégrité de la donnée.
Pour éviter qu'un signe soit imprimé sans chiffre, la clause BLANK WHEN ZERO est utilisée.

```
01 E-ITEM     PIC S999.
01 S-ITEM     PIC ++++ BLANK WHEN ZERO.
MOVE E-ITEM TO S-ITEM.
```

E-ITEM	S-ITEM
- 023	□-23
+131	+131
000	□□□□

La clause **BLANK WHEN ZERO** est utilisée avec les items numériques. Si la valeur de l'item est nulle, l'item ne contient que des espaces.

La clause **REDEFINES** permet de définir des structures de données différentes pour une même zone de mémoire.

La clause **JUSTIFIED** est utilisée pour modifier la justification (cadrage) des items alphanumériques ou alphabétiques.

La clause **SIGN**, utilisée pour les items numériques de types DISPLAY contenant un PICTURE S, indique le mode de représentation du signe.

La clause **OCCURS** sert à définir un tableau.

Le mot réservé **FILLER** est utilisé :

- Pour indiquer la présence dans un fichier d'entrée d'un ou plusieurs champs auxquels on ne fera pas référence.
- Le mot FILLER peut apparaître plus d'une fois dans un programme.
- le mot FILLER est optionnel.

4.6 LES STRUCTURES

Pour les travaux de gestion, il est nécessaire d'attribuer un nom collectif à tout ensemble de données puis des noms collectifs à des sous-ensembles,….
En COBOL nous disposons de 2 sortes d'ensembles de données, les structures et les tables.

Une structure est un système hierarchique de noms qui renvoie à une zone de mémoire centrale regroupant un ensemble de données.

Au premier niveau 01, se trouve le nom de la structure principale qui désigne l'ensemble des éléments de données mémorisées dans la totalité de la zone attribuée à cette structure. Ce nom ne pourra plus être utilisé pour définir une variable simple ou une autre structure ou un quelconque autre nom dans ce programme.

Au niveau 02, il est attribué de nouveaux noms à certaines portions de la zone. Ce processus se poursuit jusqu'à ce qu'au dernier niveau un seul nom désigne un seul et même élément de données appelé ***élément simple***.

Un élément collectif de structure est appelé un groupe et est toujours de type alphanumérique.

Remarque :

Toute ligne de description d'un niveau se termine par un point, un niveau par ligne.

Le niveau 01 doit être en marge A.

Les niveau suivant de 02 à 49 sont écrits à partir de la marge B, avec indentation.

Les éléments simples sont les seuls à disposer d'une PICTURE, le mode DISPLAY étant l'option par défaut.

Si pour certaines raisons, le programme dispose de zones décrites dans une ou plusieurs structures et qui ne doivent pas être utilisées, nous pouvons utiliser l'option FILLER.

```
01  MADATE.
           02 ANNEE     PIC 9999.
           02 MOIS        PIC 99.
           02 JOUR        PIC 99.

01  EECOls.
           02 ARTECOls    PIC X(38).
           02 TOTECOls    PIC S9(7)V9(4).
           02 TOTDEPECOls PIC S9(7)V99.
           02 FOURNECOls  PIC X(20).
           02 NOFACECOls  PIC X(20).
           02 IMPRECOls   PIC 9 VALUE 0.
           02 MSECOls     PIC 999v99.
```

4.7 LES TABLES

1. Principes généraux :

Une table est un groupe de données composé d'éléments simples ayant tous des attributs identiques. Les tables sont utilisées pour mettre en mémoire un certain nombre d'informations dans le but de pouvoir les retrouver plus facilement.
Par exemple : trouver le prix de vente d'un article connaissant son code, trouver le nom d'un employé en fonction de son matricule.
La taille d'un tableau n'est pas illimitée, les contraintes viennent du type d'ordinateur utilisé sur PC OCCURS 1300 et il faut encore tenir compte des longueurs des variables utilisées. (PRUDENCE).
La recherche d'un élément en table se fait en affectant un numéro d'ordre à chaque élément de la table. Ce numéro d'ordre est appelé **INDICE.**

Exemple : 01 TABLE.
02 QUANTITE OCCURS 5 PIC 999.

Ce qui représente une table à 1 dimension de 5 éléments de 3 caractères numériques.

Quantité 1	Quantité 2	Quantité 3	Quantité 4	Quantité 5
1	**2**	**3**	**4**	**5**

La quantité 4 de la table sera désignée par l'expression QUANTITE(4), et si nous désirons ajouter un nombre QACHAT à la quantité 2, nous devons écrire : ADD QACHAT TO QUANTITE (2).

```
01 PAYS.
  02 Province OCCURS 10 TIMES.
    03  SalesValue   PIC 9(8)V99.
    03  NumSold      PIC 9(7).
```

2. Utilisation des indices.

L'indice peut être utilisé sous forme d'un nombre ou d'une variable.

Sous forme de nombre : on écrit QUANTITE (3) pour atteindre à chaque fois l'élément 3.

Sous forme de variable : on lira par exemple l'indice N qui peut avoir les valeurs 1 à 5 puis la quantité est atteinte en écrivant QUANTITE (N).

Comme toute variable, l'indice doit être déclaré en DATA-DIVISION.
Il peut être tout simplement une zone de fichier entrée qui sert occasionnellement d'indice.

Remarque : la parenthèse gauche doit toujours être précédée par un espace à gauche, mais pas la parenthèse droite.

3. Tables à plusieurs indices.

   Cobol autorise des tables à 3 indices. Dans l'exemple suivant, on recherche les points d'un examen de math en deuxième année d'un étudiant d'info.

```
01 TABLE-ETUDIANT.
    02 LANNEE   OCCURS  3.
        03 CODE-ETUDIANT OCCURS 55.
            04  LABRANCHE  OCCURS 15  PIC 99.
```

L'ensemble des points par étudiant se retrouve dans TABLE-ETUDIANT, il s'agit d'une suite continue de caractères numériques. Il n'autorise pas l'usage d'un indice.

LANNEE, spécifie si les points recherchés sont ceux d'un étudiant de première, seconde ou troisième année. L'utilisation de LANNEE (2) nous donne une longue série de chiffres qui correspond à 55 x 15 càd les 15 résultats des 55 étudiants de seconde.
Cette zone autorise l'usage d'un indice et d'un seul.

CODE-ETUDIANT, donne pour un étudiant d'une année l'ensemble des points obtenus pour cette année la. L'utilisation de CODE-ETUDIANT (2, 17) nous donne une série de 15 chiffres de 2 caractères, cette série représente pour l'étudiant n° 17 de seconde année l'ensemble des points obtenus. Cette zone autorise l'usage de 2 indices et de toujours 2 indices.

LABRANCHE, donne pour un étudiant d'une année les points obtenus dans une branche. L'utilisation de LABRANCHE (2, 17, 4) nous donne les points de la quatrième matière de l'étudiant n° 17 de 2ème année , il s'agit d'un nombre de 2 positions en PIC 99. Cette zone autorise l'usage de 3 indices et de toujours 3 indices.

Les points du premier examen du deuxième étudiant de troisième année s'écrira :

```
LABRANCHE (3, 2, 1)
```

Pour déplacer l'ensemble des points de l'étudiant 5 de première année on écrira

```
CODE-ETUDIANT (1, 5)
```

Pour déplacer l'ensemble des points des étudiants de première année on écrira

```
LANNEE (1)
```

4. La clause OCCURS

   OCCURS *n* TIMES, spécifie qu'un groupe ou un élément se reproduit *n* fois. Dans notre exemple nous avions considéré 3 années de 55 étudiants ayant au maximum 15 banches. Cette clause ne peut pas s'écrire à un niveau 1. Pour définir une table à plusieurs niveaux, chaque OCCURS doit être de niveau inférieur sans quoi on obtient plusieurs tables à 1 niveau.
   Le mot TIMES est facultatif.

   Une clause VALUE spécifiée pour un OCCURS initialise chaque position à cette valeur.

Pour initialiser une table à des valeurs différentes on doit utiliser la clause REDEFINES sous la forme suivante.

```
01  ENSEMBLE.
    02  FILLER  PIC X(9)  VALUE "JANVIER".
    02  FILLER  PIC X(9)  VALUE "FEVRIER".
    02  FILLER  PIC X(9)  VALUE "MARS".
    02  FILLER  PIC X(9)  VALUE "AVRIL".
    02  FILLER  PIC X(9)  VALUE "MAI".
    02  FILLER  PIC X(9)  VALUE "JUIN".
    02  FILLER  PIC X(9)  VALUE "JUILLET".
    02  FILLER  PIC X(9)  VALUE "AOUT".
    02  FILLER  PIC X(9)  VALUE "SEPTEMBRE".
    02  FILLER  PIC X(9)  VALUE "OCTOBRE".
    02  FILLER  PIC X(9)  VALUE "NOVEMBRE".
    02  FILLER  PIC X(9)  VALUE "DECEMBRE".

01  TABLE-MOIS REDEFINES ENSEMBLE.
     02   MOIS OCCURS 12 PIC X(9).
```

4.8 INDEXATION

L'utilisation des tables est inchangée, mais l'accès mémoire est plus rapide, certaines instructions de recherche peuvent être utilisées ce qui facilite la programmation.
Dans ce cas les indices utilisés sont des données spéciales appelées INDEX. Ceux-ci doivent être définis par la clause INDEXED BY des clauses OCCURS.

Description d'une table :

```
01 TABLE.
    02  INFO   OCCURS 10 INDEXED BY J   PIC X(33).
```

Utilisation :

```
MOVE "Ceci est un test " TO INFO (J).

MOVE "Ceci est un test " TO INFO (J + 3).
```

Remarque :

Les INDEX ne peuvent être modifiés ou initialisés que par PERFORM ou par l'instruction SET. L'index qui définit un OCCURS doit toujours être utilisé avec le nom de son OCCURS dans notre cas J et INFO.
Une opération + ou – dans la parenthèse peut être réalisée, ce qui donne une indexation relative. L'entier utilisé doit être > 0 et non signé, le signe doit être précédé et suivi d'un blanc.
On ne peut pas mélanger les INDICES et les INDEX.

Exemple :

```
01  TABLE.
    02   PCODE   OCCURS  3  TIMES INDEXED  BY PAR.
      03   ACODE    OCCURS   3 TIMES   INDEXED BY AGE.
        04 MF    OCCURS   2 TIMES   INDEXED BY F  PIC 9999 USAGE DISPLAY.
```

La table est de dimension (3, 3, 2) et référencée par MF (PAR, AGE + 2, F – 1)

5. La clause OCCURS dans le cas des INDEX
6.

Format 1 : OCCURS n TIMES { ASCENDING / DESCENDING } KEY is nom-de-donnée-a, …

INDEXED BY nom-index.

Format 2: OCCURS n TO m TIMES [DEPENDING ON nom-de-donnée1]

{ ASCENDING / DESCENDING } KEY is nom-de-donnée2, nom-de-donnée3,…

INDEXED BY nom-index.

La clause ASCENDING ou DESCENDING spécifie l'argument de classement de la table de manière à préparer les recherches futures par l'instruction SEARCH.

```
01  TABLE.
    02  ELEMENT  OCCURS  100 ASCENDING KEY  COD.
      04  COD            PIC    XX.
      04  VALEUR       PIC  S9999 BINARY.
```

Ce qui signifie que le programmeur rangera les différentes informations de la table dans l'ordre croissant du COD. Il peut y avoir un maximum de 12 arguments de classement. Il est bien entendu que la valeur du classement ne peut dépendre que d'une variable incluse au tableau.

Exemples

Mise à zéro d'une table à 2 dimensions.

Soit une table de 30 lignes et 30 colonnes. Chaque élément est numérique entier de 4 de long en display. On demande d'initialiser le tableau à 1.

```
WORKING-STORAGE SECTION.
01  TABLE.
    02   LIGNES   OCCURS 30.
         05   COLON     OCCURS 30.
              10  ELEMENT   PIC 9999.
01  I      PIC 99 VALUE 0.
01  J      PIC 99  VALUE 0.


PROCEDURE DIVISION.
DEB.
        PERFORM VARYING I FROM 1 BY 1 UNTIL I > 30
                        AFTER  J FROM 1 BY 1 UNTIL  J > 30

                         MOVE 1 TO  ELEMENT (I,  J)
        END-PERFORM.
```

Soit une table de 30 lignes et 30 colonnes.
Chaque élément du tableau est formé d'une zone numérique entière de 4 de long en display et d'une zone alphanumérique de 20 caractères.
On demande d'initialiser les numériques à 5 et les alphanumériques à blancs.

```
WORKING-STORAGE SECTION.
02  TABLE.
    02   LIGNES   OCCURS 30.
         05   COLON     OCCURS 30.
              10  ELEMENT         PIC 9999.
              10   CARACTER      PIC  X(20).
01  I      PIC 99 VALUE 0.
01  J      PIC 99  VALUE 0.


PROCEDURE DIVISION.
DEB.
        PERFORM VARYING I FROM 1 BY 1 UNTIL I > 30
                      AFTER  J FROM 1 BY 1 UNTIL  J > 30
                        MOVE SPACE TO CARACTER(I,  J)
                        MOVE 5 TO  ELEMENT (I,  J)
        END-PERFORM.
```

Soit le vecteur 01 VECT.

```
03 A   PIC XXX   OCCURS 4.
03 B   PIC XXX   OCCURS 4.
```

Et le vecteur 01 VECT2.

```
03 A OCCURS 4.
      05 B OCCURS 4   PIC XXX.
```

Le premier vecteur représente 2 petits tableaux séparés à 1 dimension de 4 éléments chacun.
Le second correspond à 1 tableau à 2 dimensions de 4 lignes de 4 colonnes chacune.

5. LES NIVEAUX SPECIAUX

En dehors des niveaux 1 à 49 il existe encore les niveaux 66, 77, 88.

Le niveau 66.

Ce niveau s'écrit en marge A comme le 01, il permet de donner un nouveau nom symbolique à une zone élémentaire ou à un groupe. (Equivalant à un COPY BLOC AFTER)

```
 01  DEBUT.
     02  LECODE.
         03  LENUM   PIC 999.
         03  LECAR    PIC XXX.
     02  LEDEPOT.
          03  NOM      PIC XXXXX.
          03  DEBIT    PIC 9999999.
01  COMPTES RENAMES LECODE.
```

La structure COMPTES regroupe les variables LENUM et LECAR, ce qui permet l'utilisation des CORRESPONDING et l'usage de LECAR of LECODE ou de LENUM in COMPTES.

Le niveau 77

Ce niveau n'est utilisable qu'en Working-storage Section, et sert à définir directement une zone élémentaire indépendante d'une structure. *Il doit être immédiatement après la clause* WORKING-STORAGE SECTION.

```
77  Nom-de-donnée   PIC …

77  I     PIC 99 VALUE 0.
```

Le niveau 88

Le niveau 88 permet de définir des NOMS-CONDITIONS.

Supposons, par exemple, que nous devions représenter la situation matrimoniale des individus par 1 pour célibataire, 2 pour les mariés et 3 pour les veufs.
Dans ce cas la description de la variable MATRIMONIAL serait :

```
01  MATRIMONIAL    PIC   9.
```

Dans la procédure division nous devons écrire
```
IF  MATRIMONIAL = 1  PERFORM  CELIB
IF  MATRIMONIAL = 2  PERFORM  MARIES
IF  MATRIMONIAL = 3  PERFORM  VEUFS
```

Pour éviter cette écriture longue et fastidieuse nous pouvons compléter la description de la variable comme ceci :

```
01  MATRIMONIAL    PIC   9.
    88  CELIBAT     VALUE 1.
    88  MARIE       VALUE 2.
    88  VEUF        VALUE 3.
```

Et l'utiliser de la manière suivante : IF CELIBAT PERFORM CELIB
ELSE IF MARIE PERFORM MARIES
ELSE IF VEUF PERFORM VEUFS

Ce qui aura exactement le même résultat mais sera beaucoup plus parlant.

Ce niveau peut inclure plusieurs valeurs pour un même NOM-CONDITION,
par exemple : 88 Nom-condition VALUE { littèral-1 THRU littèral-2 }

{littèral-3 THRU littéral-4}

```
02  AGE    PIC 999.
 88  ENFANT    VALUE   1 THRU  17.
 88  ADO        VALUE  18 THRU 25.
 88  ADULTE     VALUE  26 THRU 65.
 88  AGES       VALUE  66 THRU 135.
```

L'utilisation reste inchangée, en procédure il suffit d'écrire IF ADO PERFORM PARAG-ADO

Une association de valeurs peut aussi être créée sous la forme suivante :

```
02  JOUR  PIC 99.
88  PAIR          VALUE 0 2 4 6 8 10 12.
88 IMPAIR         VALUE 1 3 5 7 9 11 13.
```

6. REDEFINES

Pour les besoins de certains traitements, il est parfois important de pouvoir spécifier des groupages différents dans une structure ou de définir des structures différentes au sein même d'un fichier.

Il faudra faire très attention aux USAGE aux tailles et aux VALUE utilisés dans les différentes descriptions de données.

S'il s'agit d'un fichier dans lequel une redéfinition de la structure générale est nécessaire l'option DATA RECORD **ARE** ENREG1 ENREG2, sert de **REDEFINES** et ce mot réservé ne peut être utilisé au niveau 01.

Lorsqu'il s'agit d'un REDEFINES de niveau 01 les longueurs peuvent être différentes la structure la plus longue devant toujours se trouver en première position. S'il s'agit d'un autre niveau les longueurs doivent toujours être identiques.

```
01 Rates.
  02 Rate1                PIC 99V999.
  02 Rate2    REDEFINES Rate1  PIC 999V99.
  02 Rate3    REDEFINES Rate1  PIC 9999V9.

01 HoldDate.
  02 EuroDate.
     03 EuroDay    PIC 99.
     03 EuroMonth PIC 99.
     03 EuroYear   PIC 9(4).
  02 USDate REDEFINES EuroDate.
     03 USMonth    PIC 99.
     03 USDay      PIC 99.
     03 USYear     PIC 9(4).


01 LetterTable.
  02 TableValues.
    03 FILLER PIC X(13)     VALUE "ABCDEFGHIJKLM".
    03 FILLER PIC X(13)     VALUE "NOPQRSTUVWXYZ".
  02 FILLER REDEFINES TableValues.
    03 Letter PIC X OCCURS 26 TIMES.


01 BonusTable.
  02 BonusValues.
    03 FILLER PIC X(24)        VALUE "507590758595354365406085".
  02 FILLER REDEFINES BonusValues.
    03 Province OCCURS 4 TIMES.
       04 Bonus OCCURS 3 TIMES PIC 99.
```

7. SCREEN SECTION

Cette section sert (selon la version du compilateur utilisé) à créer un dessin d'écran.

Sa forme générale est :

nombre niveau nom-écran AUTO SECURE REQUIRED FULL.

Le nombre niveau doit être un entier compris entre 01 et 49.

Le format au niveau d'un écran élémentaire est,

Nombre-niveau

BLANK SCREEN	vide l'écran
LINE NUMBER IS entier-1	
COLUMN NUMBER IS entier-2	
BELL	sonnerie
UNDERLINE	
REVERSE-VIDEO	
HIGHLIGHT	
FOREGROUND-COLOR entier-3	
BACKGROUND-COLOR entier-4	
VALUE is litéral	affiche le litéral
PIC IS description FROM TO USING } Identifier	
JUSTIFIED	
AUTO	
SECURE	
REQUIRED	
FULL.	

Remarques.

La longueur maximum pour un élément de l'écran est de 80 caractères.

AUTO : passe automatiquement au champs suivant lorsque celui en cours est rempli.

SECURE : met des * dans le champs lors de l'encodage, c'est une sécurité pour le mot de passe.

REQUIRED : une réponse non vide est exigée.

FULL : la zone doit être remplie pour la lecture, son usage permet l'utilisation des tabulations.

PICTURE : oblige l'usage d'un USING, TO ou FROM.

BLANK SCREEN : met tout l'écran à blanc et le curseur est mis en position 1.

Les codes couleurs sont :
0 noir
1 bleu
2 vert
3 cyan
4 rouge
5 magenta
6 brun
7 blanc
8 gris
9 bleu clair
10 vert clair

VALUE is litéral spécifie une chaîne de caractères qui doit être écrite lors du DISPLAY de cet écran. Ce litéral doit être entre quotes et n'est pas imprimé lors d'un ACCEPT de l'écran.
VALUE est incompatible avec PICTURE.

PICTURE spécifie le format de présentation de la donnée. Durant un DISPLAY le contenu du FROM ou du USING est transféré dans une variable temporaire. (*JAMAIS DE VARIABLES INDICEES*)

LINE et COLUMN donnent la position du curseur.

FOREGROUND et BACKGROUND-COLOR donnent la couleur des caractères et du fond.

Exemple

```
SCREEN SECTION.
01  ECRAN1.
    02  BLANK SCREEN BACKGROUND-COLOR   1.
    02  LINE 5 COL 15 PIC X(10) TO PSW SECURE BELL AUTO.
    02  LINE 10 COL 10 VALUE "VOTRE MOT DE PASSE".
    02  LINE 15 COL 20 PIC X(25) TO NOM.
    02  LINE 20 COL 11 VALUE "VOTRE NOM".


PROCEDURE DIVISION.
DEBUT.
        DISPLAY ECRAN1.
        ACCEPT ECRAN1.
```

Pour envoyer une image écran on utilise le DISPLAY, pour prendre les informations encodées sur l'écran on utilise l'ACCEPT.

Affichage : DISPLAY
Saisie : ACCEPT
FROM : On affiche à partir d ’une zone de données.
TO : On reçoit pour une zone de données.
USING : On affiche et reçoit en utilisant une zone de données

```
IDENTIFICATION          DIVISION.

 PROGRAM-ID. Program1.
 AUTHOR. jean-marie meskens.
 DATE-WRITTEN. vendredi 1 septembre 2006 8:18:52.
 REMARKS.

 ENVIRONMENT             DIVISION.
 CONFIGURATION           SECTION.
 SPECIAL-NAMES.

    DECIMAL-POINT IS COMMA.

 INPUT-OUTPUT            SECTION.
 FILE-CONTROL.

 DATA DIVISION.
  WORKING-STORAGE SECTION.
  77 ch1             PIC 99.
  77 ch2             PIC 99.
  77 somme                 PIC 999.
  SCREEN SECTION.
  01 ecran-entree.
     05 BLANK SCREEN.
     05 LINE-1.
        10 LINE 1 COLUMN 10 VALUE 'Entrez un chiffre : '.
        10 LINE 1 COLUMN 40 PIC 99 TO ch1.
     05 LINE-2.
        10 LINE 2 COLUMN 10 VALUE 'Un autre chiffre SVP : '.
        10 LINE 2 COLUMN 40 PIC 99 TO ch2.
  01 ecran-sortie.
     05 LINE-4.
           10 LINE 4 COLUMN 10 VALUE  'Voici la somme : '.
           10 LINE 4 COLUMN 40 PIC ZZZ using somme.
  PROCEDURE DIVISION.
  DEB.
        MOVE ZEROS TO ch1, ch2, somme.
        DISPLAY ecran-entree.
        ACCEPT ecran-entree.
        ADD ch1 to ch2 GIVING somme.
        DISPLAY ecran-sortie.
        accept Ecran-Sortie.
        STOP RUN.
```

8. La PROCEDURE DIVISION

8.1 LA GESTION DES FICHIERS - Les Entrées-Sorties

Un fichier est une collection de données répétitives analogues portant sur des individus ou des objets différents. Un fichier est un ensemble d'enregistrements différents mais contenant le même genre de structure.
Un tel ensemble pourrait très bien se trouver en mémoire centrale mais nous ne parlerons de fichier que lorsque l'ensemble des informations se trouve sur un périphérique externe disque ou bande.

On utilise des fichiers pour plusieurs raisons, tout d'abord par opposition avec la mémoire ou dés la fermeture de l'ordinateur toutes informations en mémoire est perdue, sur disque nous pouvons les conserver le temps nécessaire.
Les mémoires centrales sont limitées et la masse d'information à traiter peut être supérieure à l'espace disponible.
Les données contenues dans un fichier peuvent être utilisées par différents programmes et peuvent même être échangées entre ordinateurs.

Le COBOL permet *3 types d'organisations* de fichier,

l'organisation séquentielle
l'organisation séquentiel indexée
l'organisation relative.

Par organisation, il faut entendre la façon dont le fichier est organisé physiquement sur le support et l'accès à un enregistrement est la manière que nous utilisons pour atteindre cet enregistrement.

Un **accès** est dit **séquentiel** lorsque pour lire le 25ème record du fichier, il est obligatoire de lire les 24 précédents. C'est ce type d'accès qui est utilisé sur bande magnétique.

Un **accès** est dit **sélectif** lorsqu'un record bien précis du fichier peut être atteint sans devoir lire tous ceux qui le précédent. Ce type d'accès oblige l'utilisation d'une **clé** permettant de retrouver la position de l'enregistrement dans le fichier. Ce type d'accès ne s'utilise que pour les périphériques disques.
Tous les ordres d'entrées-sorties ne concernent toujours qu'un seul enregistrement à la fois.
Cobol oblige le programmeur à ouvrir chaque fichier par un ordre **OPEN** avant de pouvoir lire ou écrire sur un fichier. De même, chaque fichier sera fermé par une instruction **CLOSE** en fin de travail.

OUVERTURE DES FICHIERS OPEN

Un ordre d'ouverture de fichiers **OPEN** doit être donné pour tous les fichiers utilisés dans le programme en séparant les fichiers ENTREE de lecture **INPUT,** des fichiers SORTIE en écriture **OUTPUT** et **EXTEND.** Cette instruction OPEN doit toujours précéder les instructions de lecture ou d'écriture.

Format : **OPEN** { **INPUT** nom-de-fichier-1 / **OUTPUT** nom-de-fichier-2 / **I-O** nom-de-fichier-3 / **EXTEND** nom-de-fichier-4 }

Un fichier ne peut être cité qu'une seule fois dans une clause OPEN.
Il n'est pas obligatoire d'ouvrir tous les fichiers en même temps.

Un fichier ne doit être ouvert qu'une fois, un second passage dans la même instruction OPEN sans être passé par une instruction CLOSE pour ce fichier entraîne une erreur dans le programme.
L'ouverture d'un fichier positionne le pointeur d'enregistrement au début du fichier sauf pour l'ouverture en EXTEND.

La clause **INPUT** déclare que le *fichier est ouvert uniquement pour être lu*. Donc ce fichier doit déjà exister.

La clause **OUTPUT** déclare que le fichier est ouvert uniquement pour qu'on y écrive. Si le fichier existe déjà avec ses enregistrements, un OPEN OUTPUT les efface, le fichier est alors vidé de son contenu, et un nouveau fichier est créé.

La clause **I-O** déclare un fichier comme étant ouvert en même temps en lecture et en écriture.

La clause **EXTEND** déclare le fichier comme étant ouvert en écriture, le fichier n'est pas vidé de son contenu, le pointeur d'enregistrement se positionne à la fin du fichier. Seul un fichier à **organisation séquentiel** peut être ouvert en extend.

Un essai d'ouverture en lecture d'un fichier non existant génère une erreur d'exécution.

Exemple : OPEN INPUT FICH1 FICH2
OUTPUT IMPR FPCLIENT.

OPEN INPUT FICH1.
OPEN INPUT FICH2.
OPEN OUTPUT IMPR.
OPEN OUTPUT FPCLIENT.

Nous obtenons le même résultat dans les 2 cas mais le premier est plus rapide.

FERMETURE DES FICHIERS CLOSE.

Format : **CLOSE** nom-de-fichier1 nom-de-fichier2 ….

Un ordre de fermeture doit être donné pour tous les fichiers utilisés, donc ouverts et ceci avant l'instruction STOP RUN. Dans cette instruction on ne fait pas de distinction entre les fichiers INPUT ou OUTPUT, tous les fichiers peuvent être fermés en même temps.
Exemple : CLOSE FICH1 FICH2 IMPR FPCLIENT.

LECTURE DE FICHIERS SEQUENTIELS READ

La description du fichier et celle de son enregistrement ont déjà été réalisées dans la DATA DIVISION, de ce fait pour le programme tout se déroule comme si le fichier n'était composé que d'enregistrements logiques qui lui sont délivrés les uns après les autres à chaque ordre de lecture.

Format : **READ** *nom-de-fichier* ⎛**INTO** nom-donnée⎞ **AT END** ordre impératif.

Exemple : READ FPCLIEN INTO STRWSS AT END MOVE 1 TOSWFIN.

Remarque : si le fichier est composé d'enregistrements de différentes structures, ils seront délivrés dans la même zone déclarée en FILE SECTION. (pour rappel : redéfinition implicite)

READ InternalFilename [NEXT] RECORD
[INTO Identifier]
AT END StatementBlock
END-READ

AT END ordre impératif est toujours obligatoire et indique l'opération ou la série d'opération qui doivent être exécutées lorsque la fin du fichier est atteinte.

INTO nom-donnée signifie que le programmeur désire travailler dans une autre zone que celle définie sous le FD du fichier. Soit directement dans une autre structure de fichier soit dans une zone de la Working-storage section. **INTO** est équivalent à un **MOVE** de structure à structure.

```
PERFORM UNTIL StudentRecord = HIGH-VALUES
   READ StudentRecords    AT END MOVE HIGH-VALUES TO StudentRecord
END-PERFORM.
```

ECRITURE DE FICHIERS SEQUENTIELS WRITE

L'instruction WRITE transfère le contenu de la mémoire dans le fichier à la position où l'on se trouve dans le fichier.

Format : WRITE *nom-enregistrement* (FROM nom-de-donnée-1) **.**

FROM est équivalent à un MOVE d'une donnée vers la structure d'enregistrement.

Il faut prendre garde au fait que l'on lit un fichier mais que l'on écrit un enregistrement.

FICHIER IMPRIMANTE WRITE

Format : **WRITE** nom-enregistrement(**FROM** nom-donnée-1)

({ **BEFORE** / **AFTER** } ADVANCING { nom-de-donnée-2 LINES / nombre entier LINES / **PAGE** })

BEFORE signifie que l'écriture de la ligne sera faite avant le saut de ligne.

AFTER l'écriture sera faite après le saut de ligne.

Le nombre de lignes qu'il faut sauter est donné par la variable nom-de-donnée-2 ou par le nombre entier.

```
WRITE L80 FROM LIGNE1 AFTER 4.
WRITE L80 FROM LIGNE1 AFTER XLIG.
```

Nom-de-donnée-2 doit être une zone élémentaire < à 100. La position d'impression est alors avancée de la valeur donnée. Il est conseillé de ne pas mélanger les after et les before.

PAGE signifie que la ligne à imprimer, doit l'être sur la première ligne de la page suivante.

```
WRITE L80 FROM TITRE AFTER PAGE.
```

Une bonne organisation du programme consiste à déclarer et définir toutes les lignes d'impression en WORKING-STORAGE SECTION sous la forme suivante :

```
WORKING-STORAGE SECTION.
01  TITRE    PIC X(80) VALUE "CECI EST LE TITRE DE LA PAGE".
01  STITRE PIC X(80) VALUE ALL "_".
01  LIGNE1.
    02  FILLER PIC X(25) VALUE " DATE          LIBELLE  ".
    02  FILLER PIC X(30) VALUE "      DESCRIPTION ".
    02  FILLER PIC XXX  VALUE "LE".
    02  JJ         PIC 99.
    02  FILLER PIC X       VALUE "/".
    02  MM       PIC 99.
    02  FILLER PIC X       VALUE "/".
    02  ANNEE PIC 9999.
PROCEDURE DIVISION.
DEB.
        ACCEPT MADATE FROM DATE.
        OPEN OUTPUT IMPR.
        WRITE LIMPR FROM TITRE.
        WRITE LIMPR FROM  STITRE.
        MOVE  JOUR  TO JJ
        MOVE MOIS TO MM
        MOVE AN TO ANNEE
        WRITE LIMPR FROM  LIGNE1.
        CLOSE IMPR.
        STOP RUN.
```

Dans le cas d'un fichier de données

```
OPEN OUTPUT StudentFile.
MOVE "9334567Frank Curtain  LM051" TO StudentDetails.
WRITE StudentDetails.
 MOVE "9383715Thomas Healy   LM068" TO StudentDetails.
 WRITE StudentDetails.
 CLOSE StudentFile.
 STOP RUN.
```

STATUS KEY

Il est possible de contrôler le bon achèvement d'une instruction d'entrèe-sortie grâce à un indicateur comportant 2 caractères numériques.
Tout d'abord la clause FILE STATUS is nom-de-donnée dans l'ENVIRONMENT DIVISION au niveau de la clause SELECT doit être définie.
Nom-de-donnée doit être déclaré en WORKING-STORAGE SECTION sous la forme d'un PIC XX. Ensuite l'ordinateur fait le reste, à chaque OPEN – READ – WRITE – CLOSE - …. Il vérifie le bon fonctionnement de l'instruction utilisée et dans ce cas la variable nom-de-donnée est remplie d'un "00" ce qui signifie que l'instruction c'est bien déroulée. Dans le cas contraire il existe dans le manuel du compilateur utilisé une série de code 10 30 34 90 92 96 …. Qui indique le mauvais fonctionnement de l'opération, soit que le fichier n'est pas trouvé, que le EOF est rencontré ou une erreur sur la longueur ou sur la clé etc..

Exemple :
```
SELECT OPTIONAL FPCLIEN ASSIGN TO DISK
        FILE STATUS IS SKCLIEN.

WORKING-STORAGE SECTION
01  SKCLIEN  PIC XX VALUE "00".

PROCEDURE DIVISION.
DEB.
        OPEN INPUT FPCLIEN.
        IF SKCLIEN NOT = "00" PERFORM ERREUROPEN.
```

L'INSTRUCTION ACCEPT

Le verbe ACCEPT est un verbe particulier de lecture de données à partir de l'écran. On lira toujours des blocs de 80 caractères, même si vous ne devez en lire que 16 par exemple.

Format : **ACCEPT** nom-de-donnée [FROM nom-symbolique]

Le nom-de-donnée représente une zone de donnée en WORKING-STORAGE SECTION dans laquelle seront transmis les octets transmis par le périphérique.

Le nom-symbolique permet de définir un système autre que l'écran pour l'introduction des données.

⚠ Attention : cette option est différente selon le compilateur ou l'ordinateur utilisé, référencez-vous auprès de votre manuel utilisateur du Labo

Format spécial de l'instruction ACCEPT

ACCEPT nom-de-donnée **FROM** { **DATE** / **DAY** / **TIME** }

Ce format permet de transférer dans la zone nom-de-donnée le contenu de l'un de 3 compteurs spéciaux DATE DAY TIME.

DATE est un PIC 9(6) pour AA MM JJ sous la forme 990526

DAY est un PIC 9(5) pour AA JJJ sous la forme 99236

TIME est un PIC 9(8) pour HH MM SS CS

Exemple :

```
01 DATE-DU-JOUR.
     02  AN          PIC  99.
     02  MOIS        PIC 99.
     02  JOUR        PIC 99.

PROCEDURE DIVISION.

     ACCEPT DATE-DU-JOUR FROM DATE.
```

En version 2000 ACCEPT DATE-DU-JOUR FROM CENTURY-DATE

```
01 DATE-DU-JOUR.
     02  AN          PIC  9999.
     02  MOIS        PIC 99.
     02  JOUR        PIC 99.
```

L'INSTRUCTION DISPLAY

Ce verbe correspond à l'émission de données vers l'imprimante ou l'écran.

Format : **DISPLAY** { nom-de-donnée-1 / littéral- 1 } { nom-de-donné-2 / littéral-2 } …….

UPON nom-symbolique.

Exemple :

```
DISPLAY "FIN DU PROGRAMM".
DISPLAY MESSAGE1.
```

Le nom-symbolique **PRINTER** indique l'écriture sur l'imprimante.

Le nom-de-donnée peut être une structure.

8.2 LES INSTRUCTIONS ARITHMETIQUES

Ces instructions sont de 2 types,

- les opérations simples : **ADD**
  SUBTRACT
  MULTIPLY
  DIVIDE

- les opérations complexes : **COMPUTE** et ses opérateurs **+ - * / ****

Ces instructions disposent d'options communes :

- **GIVING** nom-de-donnée qui indique que le résultat de l'opération sera mémorisé dans nom-de-donnée.
  ADD VAR1 VAR2 GIVING RESULTAT

- **ROUNDED** après une opération le résultat tronque les décimales au-delà de ce qui a été déclaré dans la PIC. Ainsi la quantité 234,478 sera de 234,4 si la PIC de la variable résultat est de 999V9.
  Avec ROUNDED, COBOL ajoute 5 au premier chiffre non significatif de droite pour réaliser l'arrondi à l'unité supérieure du chiffre le moins significatif si le premier chiffre non significatif de droite est >= 5.
  Avec **rounded** notre exemple devient 234,5.

ON SIZE ERROR ordre impératif

Cette option permet de prévenir des incidents de calcul comme

- la division par zéro
- les dépassements de capacité (pour rappel un numérique est limité à 18 chiffres)

exemple : DIVIDE A BY B ON SIZE ERROR MOVE ZERO TO B

Les calculs ne pourront se faire naturellement qu'avec des zones élémentaires numériques, il en ira de même pour les constantes.

L'ADDITION ADD

Format n°1 :

ADD { Nom-de-donnée-1 / littéral-numérique-1 } [nom-de-donnée-2 / littéral-numérique-2]

TO nom-de-donnée [**ROUNDED**]

nom-de-donnée-n [**ROUNDED**]

[**ON SIZE ERROR** ordre-impératif]

On ajoute nom-de-donnée-1 ou littéral-numérique-1 et nom-de-donnée-2 ou littéral-numérique-2 à la variable nom-de-donnée et le résultat peut être arrondi.

Exemple : ADD 1000 QTE1 TO TOTAL ROUNDED ON SIZE ERROR MOVE 0 TO TOTAL

Ce qui signifie ajouter 1000 + QTE1 + TOTAL arrondir, transférer le résultat de l'opération dans TOTAL. En cas d'erreur mettre TOTAL à 0.

Format n° 2

ADD { nom-donnée-1 / littéral-1 } { nom-donnée-2 / littéral-2 } …. { nom-donnée-n / littéral-n }

GIVING nom-de-donnée [**ROUNDED**] [**ON SIZE ERROR** ordre-imp.]

Dans ce cas on ajoute nom-donnée-1 à nom-donnée-2 à….. nom-donnée-n le résultat est placé dans nom-de-donnée, éventuellement arrondi.

Exemple : ADD TVA TOTHTVA GIVING TOTTVAC

Format n°3

ADD CORRESPONDING nom-de-groupe-1 **TO** nom-de-groupe2

[**ROUNDED**] [**ON SIZE ERROR** ordre-imp.]

Avec ce format, les zones élémentaires du groupe-1 sont additionnées aux données élémentaires du groupe-2 ayants des noms-données identiques, et les résultats sont placés dans les zones élémentaires du groupe-2.
L'abréviation COBOL de **CORRESPONDING** est **CORR.** Les clauses OCCURS et REDEFINES sont mal supportées.

Prenons par exemple un fichier paie mensuel d'enregistrement ENRMENS et le fichier ENRTOT. De description :

```
01   ENRMENS.
     02  CODE-EMPLOYEE     PIC 999.
     02  HEURE-P           PIC 999.
     02  SAL-BRUT          PIC 9(6).
01  ENRTOT.
     02  COD-EMPL          PIC 999.
     02  HEURE-P           PIC 999.
     02  SAL-BRUT          PIC 9(6).
     02  SAL-NET           PIC 9(6).
```

Le fait d'écrire ADD CORR ENRMENS TO ENRTOT, engendre les additions suivantes ,

HEURE-P OF ENRTOT = HEURE-P OF ENRTOT + HEURE-P OF ENRMENS
SAL-BRUT OF ENRTOT = SAL-BRUT OF ENRTOT + SAL-BRUT OF ENRMENS

ADD identifier-1 TO identifier-2 ROUNDED ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre impératif-2
END-ADD

ADD identifier-1 TO identifier-2 GIVING identifier-3 ROUNDED
ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-ADD

Identifier-1 peut être une constante ou une variable numérique ou un ensemble des deux.
Les ordres impératifs 1 et 2 peuvent être constitués d'une ou de plusieurs instructions.
Avec l'option GIVING identifier-2 peut également être une constante numérique puisque le résultat est placé dans identifier-3.

LA SOUSTRACTION SUBTRACT

Format 1 :

SUBTRACT (Identifier-1 / Literal-1) **FROM** identifier-2 **ROUNDED**

ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-SUBTRACT

Format 2 :

SUBTRACT (Identifier-1 / Litéral-1) **FROM** (identifier-2 / litéral-2) **GIVING** identifier-3
ROUNDED ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-SUBTRACT

Format 3 :

SUBTRACT CORR identifier-1 **FROM** identifier-2 **ROUNDED**
ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-SUBTRACT

Remarques: identifier-1 peu être une série de variables ou de constantes numériques.
Les options ROUNDED et ON SIZE ERROR ne sont pas obligatoires.
END-SUBTRACT ne s'applique que si les options SIZE ERROR sont utilisées.

Exemple : SUBTRACT V1 V2 12 FROM TOTAL. (TOTAL = TOTAL – (V1 + V2 + 12))
SUBTRACT V1 V2 12 FROM TOTAL GIVING SOUSTOT.
(SOUSTOT = TOTAL – (V1 + V2 + 12))

Dans le 1er cas TOTAL doit être un numérique de travail, dans le 2ème cas SOUSTOT peut être un numérique d'édition car il n'intervient pas dans le calcul.

LA MULTIPLICATION MULTIPLY

Format 1 : **MULTIPLY** (Identifier-1 / Litéral-1) **BY** identifier-2 **ROUNDED**
ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-MULTIPLY

Format 2 : **MULTIPLY** (Identifier-1 / Litéral-1) **BY** (identifier-2 / litéral-2) GIVING identifier-3 **ROUNDED**
ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-MULTIPLY

Exemple : MULTIPLY 12 BY V1. (V1 = V1 * 12)
MULTIPLY V1 BY V2 GIVING V3. (V3 = V1 * V2)

LA DIVISION DIVIDE

Format 1 : **DIVIDE** { identifier-1 / litéral-1 } **INTO** identifier-2 **ROUNDED**
ON SIZE ERROR ordre-impératif-1

NOT ON SIZE ERROR ordre-impératif-2

END-DIVIDE

Format 2 : **DIVIDE** { identifier-1 / litéral-1 } { **INTO** / **BY** } { identifier-2 / litéral-2 } **GIVING** identifier-3 **ROUNDED**
ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-DIVIDE

Format 3 : **DIVIDE** { identifier-1 / litéral-1 } { **INTO** / **BY** } { identifier-2 / litéral-2 } **GIVING** identifier-3 **ROUNDED**

REMAINDER identifier-4
ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-DIVIDE

Exemple : DIVIDE 10 INTO V1. (V1 = V1 / 10)
DIVIDE V1 BY V2 GIVING V3. (V3 = V1 / V2)
DIVIDE V1 BY V2 GIVING V3 REMAINDER RESTE.
(V3 = V1 / V2) et RESTE = V1 – (V2 * V3) ***le tout en entier sans quoi il n'y a pas de reste***

L'INSTRUCTION COMPUTE

COMPUTE nom-item-1 = { expression-arithmétique / constante / nom-item }

Les symboles arithmétiques valides sont :

+ pour l'addition
- pour la soustraction
* pour la multiplication
/ pour la division
** pour l'exposant Un symbole arithmétique doit être précédé et suivi par au moins un espace

Format : **COMPUTE** identifier-1 **ROUNDED** = expression
ON SIZE ERROR ordre-impératif-1
NOT ON SIZE ERROR ordre-impératif-2
END-COMPUTE

L'expression : doit être composée de constantes et de variables numériques séparées par des opérateurs arithmétiques.

Les *opérateurs* sont **: + - * / ** ( )**

Avec + pour addition, - pour la soustraction, * pour la multiplication le / pour la division
et ** pour un exposant.
Exemple : COMPUTE RESULTAT = A + (B – 3) * (G / 4) + (E ** 2).

L'expression est analysée de la gauche vers la droite en tenant compte des priorités suivantes : d'abord les exposants **, ensuite les * et / puis les + et -.
Il faut autant de (gauches que de) droites

8.3 LES MOUVEMENTS DE DONNEES MOVE

Quels que soient les PICTURE et USAGE, il n'existe qu'un seul ordre de mouvement de données en mémoire centrale, l'ordre MOVE.

Format 1 : MOVE (nom-de-donnée-1 / littéral) TO nom-de-donnée-2 (nom-de-donnée-3).....

Exemple : MOVE TVA TO TVAED. (TVAED = TVA)
MOVE "EXERCICE N° 1 " TO TITRE. (TITRE = 'EXERCICE N° 1)
MOVE ZERO TO V1 V2 V3 V4 V5.
MOVE STRUCTURE1 TO STRUCTURE2.

Remarque : le fait d'avoir un seul verbe MOVE pour tous les types de données implique de veiller à ne pas mouvementer n'importe quel type de donnée sur n'importe quel type de zone de données.
Dans le cas des alphabétiques **A :** ne pourront être mouvementés que sur des zones élémentaires **A** ou **X** et sur des zones groupes (les structures sont toujours considérées comme des PIC X).

Les alphanumériques **X :** ne pourront en principe être mouvementés que sur des zones PIC X ou sur des structures. Toutefois, le mouvement sur une zone numérique est autorisé, mais le programmeur doit être certain que la zone ne contient que des données numériques sans quoi il provoquera un arrêt du programme (MCH1202).

Les numériques, ne peuvent être mouvementés que vers des numériques ou des numériques d'édition. L'instruction MOVE convertit immédiatement n'importe quel usage en n'importe quel autre usage, pour passer d'un DISPLAY à un BINARY ou d'un PACKED-DECIMAL vers un DISPLAY aucune information ou instruction complémentaire n'est demandée au programmeur, la conversion est immédiate.

Principes de cadrage :

Les règles de cadrage sont toujours dictées par la zone réceptrice.

Dans le cas des alphabétiques et alphanumériques :
La donnée est cadrée à gauche avec troncature à droite si la zone émettrice est trop grande ou complétée avec des caractères blancs à droite si la zone est plus courte.
MOVE " " TO L120 avec 01 L120 PIC X(120). MOVE 120 blancs dans L120

Dans le cas des numériques :
Les nombres sont alignés sur la partie décimale, si celle-ci est absente de la zone réceptrice elle est considérée comme étant à l'extrême droite.
Il a ensuite troncature éventuel ou complément de zéros à droite et à gauche si le nombre de chiffres désignés par la PIC de la zone réceptrice l'exige.
Si la zone réceptrice n'a pas de signe alors c'est la valeur absolue qui est transférée.
Si la zone émettrice n'est pas purement numérique le résultat est alors imprévisible.

Un INDEX est formellement interdit dans un MOVE.

Le contenu de la zone émettrice reste exactement le même après l'instruction MOVE.

MOVE et REDEFINES :

Dans le cas d'une redéfinition de zone 02 B……
02 C REDEFINES B…

le fait d'exécuter un MOVE B TO C ou MOVE C TO B alors qu'il s'agit de la même zone mémoire entraîne un résultat inconnu!!

LE MOVE CORRESPONDING

Format : MOVE CORR nom-de-groupe-1 TO nom-de-groupe-2

Le MOVE CORRESPONDING suit le même principe que l'option CORR des instructions arithmétiques. Les données de la zone groupe-1 sont mouvementées sur les zones de même nom de la zone groupe-2.

Toutefois, avec MOVE, il n'est pas exigé que les zones de même nom soient toutes deux élémentaires , mais qu'au moins l'une des deux zones le soit.

Mais, toute zone élémentaire décrite avec une clause OCCURS ou REDEFINES est ignorée.

Utilisation des constantes figuratives.

Les constantes figuratives SPACE(S) et ZERO(S) peuvent être utilisées dans un MOVE.

SPACE : pour mettre une zone PIC X ou PIC A à blanc, toute la zone est remplie de blancs, il peut également être question d'une structure.

Exemple : MOVE SPACE TO VAR. La variable VAR PIC XX ou PIC AA est remplie de blancs

MOVE SPACE TO STRUCTURE.

Ici toutes les variables contenues dans le groupe STRUCTURE sont mises à blancs. Il s'agit donc d'être très prudent, car si dans la description de structure nous avons une variable en PIC 9 son contenu sera un BLANC.

ZERO : permet de mettre n'importe quel variable PIC 9 usage…… à ZERO. Il en ira exactement de même avec un groupe et dans ce cas toutes les variables du groupe seront initialisées à 0.

Exemple : MOVE ZERO TO VAR Dans ce cas VAR PIC 99.
MOVE ZERO TO STRUCTURE. Et dans ce cas 01 STRUCTURE.
02 A PIC 9999 BINARY.
02 B PIC 9(5).

A et B sont mises à 0.

Remarque : comme en Pascal l'usage d'un caractère de la chaîne est possible par l'option
MOVE CHAINE(i:j) TO A

Ou de l'hexadécimal : MOVE X"09" TO A

8.4 L'INSTRUCTION INITIALIZE

Format : **INITIALIZE** identifier-1 [**REPLACING**]

({ **ALPHABETIC** / **ALPHANUMERIC** / **NUMERIC** / **ALPHANUMERIC-EDITED** / **NUMERIC-EDITED** } **BY** [identifier-2 / litéral-1])

Identifier-1 est la variable ou la structure qui reçoit l'initialisation. Elle peut être tout ou partie de table mais ne peut jamais contenir les options DEPENDING ON POUR UN OCCURS. Les index sont interdits.

Identifier-2 ou litéral-1 représente le contenu d'initialisation, *par défaut* les numériques et les numériques édités sont initialisés à 0 et les alphabétiques les alphanumériques et les alphanumériques édités à blancs.

Si REPLACING est utilisé, identifier-2 ou litéral-1 doivent être compatible avec la catégorie indiquée (en respectant les règles du MOVE).

Lorsqu'une catégorie est indiquée après un REPACING, alors seulement les variables appartenant à identifier-1 et reprisent dans cette catégorie sont initialisées.

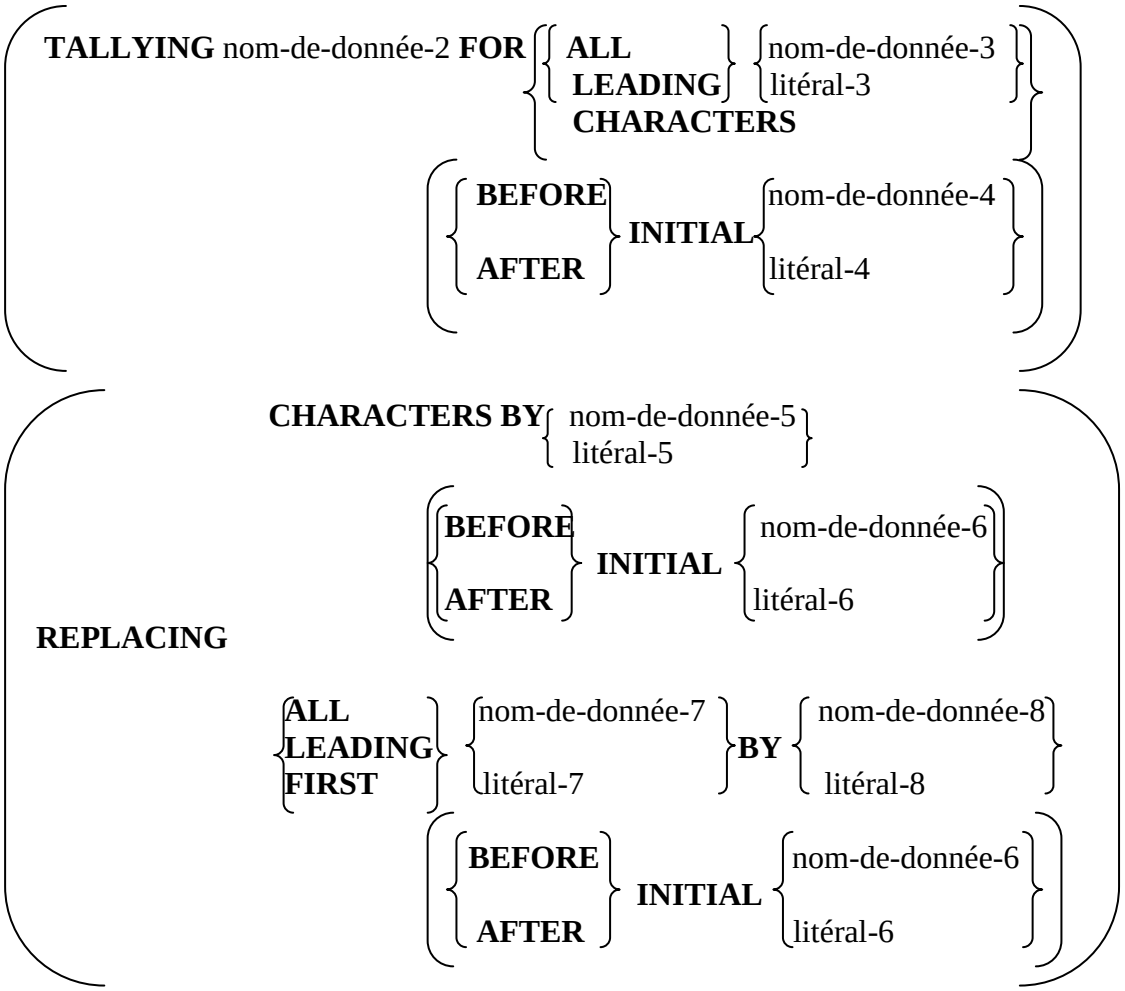
Exemple : INITIALIZE VAR1.

```
INITIALIZE STRUCTURE1.

INITIALIZE  VAR2  BY "*".

INITIALIZE  STRUCTURE2 REPLACING NUMERIC BY VAR3.
```

8.5 L'INSTRUCTION INSPECT

Cette instruction permet de compter le nombre de fois qu'une configuration particulière de caractères apparaît dans une zone de données et/ou de remplacer ces caractères par d'autres. Cette instruction ne s'applique pas aux variables avec index même à un niveau 01.

Format 1 : **INSPECT** nom-de-donnée-1



Nom-de-donnée-1 est la zone que l'on désire inspecter. Elle doit être en USAGE DISPLAY.

TALLYING signifie que l'on désire compter le nombre de fois qu'une configuration de caractères désignée par nom-de-donnée-3 ou litéral-3 apparaît dans nom-de-donnée-1 et que le résultat du comptage soit placé dans nom-de-donnée-2.

ALL signifie que toutes les occurrences doivent être comptées. Les caractères déjà utilisés dans un comptage ne sont jamais réutilisés.
Exemple: INSPECT VARA TALLYING COMPTEUR FOR ALL "NN".
Si VARA contient "NNNNN+NNN" COMPTEUR contiendra après l'opération de comptage 3, car le 5ème et le 9ème N ne peuvent pas être regroupés avec ceux déjà utilisés.

LEADING signifie que seules les premières apparitions contiguës doivent être comptées.
Exemple: INSPECT VARB TALLYING COMPTEUR FOR LEADING "N".
Si VARB contient "NN+NNN+NNN" COMPTEUR contiendra 2 puisqu'il y a 2 N en tête.
l'opération VARE contiendra "00000000".

CHARACTERS compte le nombre de caractères contenu dans nom-de-donnée-1.
Exemple: INSPECT VARC TALLYING COMPTEUR FOR CHARACTERS.
Si VARC contient "N&N+MOI+123" COMPTEUR contiendra 11.
Pour rendre l'usage de cette instruction plus partiel nous disposons des options BEFORE et AFTER.

BEFORE INITIAL : (nom-de-donnée-4 ou litéral-4) signifie que le comptage doit s'arrêter dès que la première apparition du ou des caractères désignés est rencontrée.

Exemple: INSPECT VARD TALLYING COMPTEUR FOR CHARACTERS
BEFORE INITIAL "+".
Si VARD contient "N&N+MOI++123" COMPTEUR contiendra 3.

AFTER INITIAL est l'inverse du BEFORE, on compte les caractères qui se trouvent après nom-de-donnée-4 ou litéral-4.

Exemple: INSPECT VARDAF TALLYING COMPTEUR FOR CHARACTERS
AFTER INITIAL "+".
Si VARD contient "N&N+MOI++123" COMPTEUR contiendra 8.

REPLACING : indique que nous voulons remplacer certains caractères par d'autres.

CHARACTERS, tous les caractères de nom-de-donnée-1 doivent être remplacés par le caractère désigné par nom-de-donnée-5 ou litéral-5.

Exemple: INSPECT VARE REPLACING CHARACTERS BY "0".
Si VARE contient "N&N+MOI3" après
ALL indique que toutes les occurrences désignées par nom-de-donnée-7 ou litéral-7 doivent être remplacées par nom-de-donnée-8 ou litéral-8. Il n'y a jamais réutilisation de caractères déjà utilisés dans une occurrence préalable.

Exemple: INSPECT VARF REPLACING ALL "N" BY "0".
Si VARF contient "NN+NT23N" après l'opération VARF contiendra "00+0T230".

LEADING indique que le remplacement s'arrêtera dés qu'un autre caractère que nom-de-donnée-7 est rencontré.

FIRST indique que le remplacement ne doit être effectué que pour la première apparition de nom-de-donnée-7.

Exemple: INSPECT VARG REPLACING FIRST "N" BY "0".
Si VARG contient "NN+NT23N" après l'opération VARG contiendra "0N+NT23N".

Les options BEFORE et AFTER sont utilisées de la même manière que dans TALLYING.

Les clauses **TALLYING** et **REPLACING** peuvent être utilisées en même temps,
Exemple: INSPECT VARTOT TALLYING COMPTE FOR CHARACTERS BEFORE "+"
REPLACING ALL "N" BY "*".
Si VARTOT contient "NN+NT23N" après l'opération VARTOT contiendra "**+*T23*" et COMPTE contiendra 2.

Le programmeur doit initialiser nom-de-donnée-2 à zéro car TALLYING additionne sans remettre à zéro.

Format 2: INSPECT nom-de-donnée-1 CONVERTING {nom-de-donnée-9 / litéral-9} TO {nom-de-donnée-10 / litéral-10}

{BEFORE / AFTER} nom-de-donnée-11.

Exemple: INSPECT VAR11 CONVERTING "ABCD" TO "WXYZ" AFTER QUOTE BEFORE "=".

Si VAR11 contient (AC"AEBDFBCD=AB"D) alors le résultat est (AC"WEXZFXYZ=AB"D)

INSPECT CHAINE CONVERTING MIN TO MAX
Avec MIN pic x(26) value 'abc…yz'.
Et Max pic x(26) value 'A Z'.

Remarque

Inspect ZONE TALLYING CPT1 FOR ALL 'A'
CPT2 FOR ALL 'E'
CPT3 FOR ALL 'I'

8.6 L'INSTRUCTION STRING

STRING { nom-de-donnée-1 / litéral-1 } [nom-de-donnée-2 / litéral-2] **…… DELIMITED BY** { **SIZE** / litéral-3 / nom-de-donnée-3 }

{ nom-de-donnée-4 / litéral-4 } [nom-de-donnée-5 / litéral-5] ……. DELIMITED BY { SIZE / litéral-6 / nom-de-donnée-6 }

INTO nom-de-donnée-résultat

[**WITH POINTER** nom-de-pointeur]

[**ON OVERFLOW** ordre impératif]

Les zones de données émettrices et réceptrices doivent être alphanumériques et les transferts de données suivent les règles correspondantes. Si la zone réceptrice est plus grande il n'y a pas remplissage par des blancs à droite. De même nom-de-donnée-résultat n'est jamais réinitialisée à blanc, c'est le travail du programmeur.

DELIMITED BY permet de spécifier une limite de transfert de données, soit en fonction de la taille "SIZE", soit en fonction du ou des caractères de nom-de-donée-3 ou litéral-3. Ces caractères là ne sont pas transmis.

POINTER permet de préciser la position de gauche du début du transfert dans la zone résultat. La variable nom-de-pointeur est un numérique. Normalement cette valeur est incrémentée de 1 par caractère transféré.

OVERFLOW en option avec POINTER indique l'ordre impératif à exécuter.

Exemple : STRING "CECI EST LE TITRE " GROUPE BLANC "1999" DELIMITED BY SIZE
INTO L80.

```
STRING INFO DELIMITED BY SIZE INTO L80 WITH POINTER PT ON OVERFLOW
        PERFORM PAR-ERREUR.
```

String A delimited by size B delimited by space C delimited by '*' into …

8.7 L'INSTRUCTION UNSTRING

Le but de cette instruction est d'éclater une variable DISPLAY en plusieurs morceaux. La règle d'éclatement est donnée par DELIMITED BY.

Format : **UNSTRING** nom-de-donnée-1

(**DELIMITED BY** **ALL** { nom-de-donnée-12 / litéral-12 }
(**OR** **ALL** { nom-de-donnée-13 / litéral-13 }))
[**WITH POINTER** nom-de-pointeur]
[**TALLYING IN** nom-de-compteur]
[**ON OVERFLOW** ordre impératif]
INTO nom-de-donnée-i [**DELIMITER IN** nom-de-donnée-i1] [**COUNT IN** nom-de-donnée-i2]
(nom-de-donnée-j **DELIMITER IN** nom-de-donnée-j1 **COUNT IN** nom-de-donnée-j2
……)

Les mêmes règles que pour le STRING sont d'application. ALL est destiné à éliminer les redondances d'un caractère délimiteur. Par exemple ALL "AB" signifie que AB ou même ABABAB sont délimiteurs.

NOM-DE-POINTEUR , numérique DISPLAY, que le programmeur doit initialiser à 0 permet de compter le nombre de caractères examinés dans la zone nom-de-donnée-1.

NOM-DE-COMPTEUR, numérique DISPLAY, compte le nombre de zones réceptrices créées.

OVERFLOW, arrête le déroulement de l'instruction si le contenu du pointeur est négatif ou > à la taille de la zone nom-de-donnée-1, ou lorsque la zone nom-de-donnée-1 n'est pas entièrement examinée mais que l'instruction UNSTRING ne dispose plus de zones réceptrices i,j,…

Si un OVERFLOW est déclenché sans que l'option ne soit spécifiée, l'instruction UNSTRING est arrêtée et le programme passe à l'instruction suivante.

Les zones nom-de-donnée-i etc. sont les zones réceptrices.

DELIMITER IN spécifie la variable dans laquelle on désire transférer le délimiteur qui a été utilisé pour cet éclatement.

COUNT IN indique le nombre de caractères transférés par cet éclatement.

Exemple :

```
01  ZONE  PIC X(12).
01  NOM   PIC X(5).
01  AN      PIC XXXX.

UNSTRING ZONE DELIMITED BY SPACE INTO NOM AN.
```

Si ZONE = "ANNEE 1999 " alors le résultat est NOM = "ANNEE" et
AN = "1999".

UNSTRING A delimited by '.' or '/' ….

8.9 LES INSTRUCTIONS CONTIONNELLES

Il existe différents types de conditions en COBOL. Une condition peut-être :

- de relation
- de classe
- de signe

Une condition de relation a la structure suivante :

nom-item-1 constante-1 expr-arith-1	IS [NOT]	GREATER THAN LESS THAN EQUAL TO > ou < ou = ou <= ou >= ou <>	nom-item-2 constante-2 expr-arith-2

IF MONTANT IS EQUAL TO 25.00 ….
IF TOTAL > MONTANT ….
IF SALAIRE-BRUT NOT < LIMITE ….

Une condition de classe a la structure suivante:

nom-item IS [NOT] { NUMERIC / ALPHABETIC }

IF CODE-PROV IS NOT ALPHABETIC ….
IF MONTANT IS NUMERIC ….

Une condition de signe a la structure suivante:

{expr-arith / nom-item} IS [NOT] { POSITIVE / NEGATIVE / ZERO }

L ’item doit être numérique (PIC 9)

IF TOTAL-CREDITS IS NEGATIVE ….
IF MONTANT IS POSITIVE ….
IF MONTANT-EN-CAISSE - DEBOURSES IS NOT ZERO ….

Les conditions peuvent être combinées pour former des conditions complexes.

Avec AND et OR

IF MOYENNE = 0 AND NBR-ETUDIANT NOT EQUAL TO ZERO ….
IF A = B OR C > D ….

La structure de sélection est traduite en COBOL par l ’énoncé IF.

IF condition
THEN
        Instruction impérative ….
[ELSE   Instruction impérative ….]
[END-IF]

```
IF HRES-TRAV > 40 THEN
        COMPUTE HRES-SUPP = HRES-TRAV - 40
        COMPUTE PAYE-SUPP = HRES-SUPP * TAUX-HOR
        MOVE 40 TO HRES-REG
ELSE
        MOVE ZERO TO PAYE-SUPP
        MOVE HRES-TRAV TO HRES-REG.
COMPUTE PAYE-REG = HRES-REG * TAUX-HOR.
COMPUTE PAYE-TOTALE = PAYE-REG + PAYE-SUPP
END-IF.
```

Il est possible de donner un nom à une condition et d ’utiliser celui-ci à sa place.
Un nom de condition est défini à l ’aide du niveau 88.

88 nom-condition VALUE const-1 [{THRU} const-2 ….]

La description de niveau 88 doit suivre immédiatement celle de l ’item élémentaire auquel on fait référence. La constante doit être du même type.

```
05 note      PIC X.
88 echec     VALUE ‘E’.
```

Quand le nom de condition est utilisé, la valeur de l ’item et la valeur indiquée dans la description de niveau 88 sont comparées.

```
IF echec THEN
        PERFORM traiter-echec
ELSE PERFORM traiter-reusssite
END-IF.
```

EVALUATE

est utilisé pour représenter une structure de sélection de type `CASE ’.

```
EVALUATE CHOIX
        WHEN ‘A’ PERFORM TRAITER-AJOUT
        WHEN ‘M’ PERFORM TRAITER-MODIFICATION
        WHEN ‘S’ PERFORM TRAITER-SUPPRESSION
        WHEN OTHER PERFORM TRAITER-ERREUR
END-EVALUATE.
```

9. LES BRANCHEMENTS

Les instructions **GO TO et ALTER**

Format : **GO TO** nom-de-paragraphe

ALTER nom-de-paragraphe TP PROCEED TO nom-de-paragraphe2

Le GO TO indique un branchement immédiat vers le nom de paragraphe spécifié, il rompt ainsi les liens que la programmation structurée a créés.

L'ALTER permet de modifier le nom du paragraphe vers lequel un GO TO renvoi. Ce GO TO doit être la seule instruction contenue dans le paragraphe nom-de-paragraphe et nom-de-paragraphe-2 est le nom de paragraphe auquel le GO TO doit brancher le programme.

```
P1.  GO TO  TRF.

P2. ……
    ……..
     ALTER P1 TO PROCEED TO P3.
P3.
```

Tout transfert vers P1 signifie GO TO P3. Il est possible de ne pas écrire TRF dans le GO TO de P1 mais alors le programmeur doit commencer par affecter un ALTER à ce paragraphe.

GO TO DEPENDING

Format : **GO TO** paragraphe1 paragraphe2 paragraphe3 paragraphe4 paragraphe-n
DEPENDING ON nom-de-donnée.

Nom-de-donnée représente une zone de donnée numérique entière, quand il contient la valeur 1, le branchement est fait au premier paragraphe. Quand la valeur est 2 c'est le 2ème paragraphe qui est utilisé et ainsi de suite.

Exemple : GO TO P1 P2 P3 P4 P5 DEPENDING ON VARTEST.

Si VARTEST contient une valeur incompatible avec le GO TO alors cette instruction est ignorée.

10. L'INSTRUCTION PERFORM

Il arrive fréquemment qu'une même séquence d'instructions intervienne à différents endroits dans le cours du programme. Il vient alors à l'esprit de n'écrire cette séquence d'instructions qu'une seule fois et de s'y référer chaque fois que cela s'avère nécessaire.

Si une telle séquence est extérieure au programme et donc compilée séparément on l'appellera un sous-programme et son utilisation sera réalisée au moyen d'une instruction CALL.
Si au contraire cette séquence d'instructions fait partie du programme, il s'agira alors d'une séquence indépendante sur laquelle on se branche par une instruction **PERFORM**.

Format 1 : PERFORM paragraphe-1 [THRU parapgraphe-2]

Exemple : PERFORM PAR1.

Cette instruction PERFORM provoque le branchement au paragraphe portant le nom de PAR1, lorsque le traitement de la dernière instruction de PAR1 est terminé le contrôle du programme repasse à l'instruction qui suit immédiatement le PERFORM.

Exemple : PERFORM P1 THRU P3.

Dans ce cas, il y a branchement vers le paragraphe P1 et toutes les instructions comprises entre P1 et la dernière instruction du paragraphe P3 sont exécutées. Entre P1 et P3 il peut y avoir d'autres paragraphes qui seront traités de la même manière. Dans ce cas une instruction de type GO TO limitée à ces paragraphes ne rompt pas les liens de la programmation structurée.
Il arrive aussi que dans ce cas la seule et unique instruction du dernier paragraphe soit une instruction EXIT, qui ne génère rien mais permet le transfert en fin de procédure.

```
Exemple:          PERFORM P1  THRU P5.
          P1.

          P3.

               GO TO P5.
          P4.

          P5.  EXIT.
```

Un format particulier du PERFORM existe sous la forme **PERFORM**
instructions Cobol
END-PERFORM

Ce qui est équivalent à un bloc dont le début est le mot clé PERFORM et la fin END-PERFORM.

```
Exemple : IF A = B PERFORM
                    Instructions Cobol
                   END-PERFORM
```

Format 2 : **PERFORM** paragraphe-1 [**THRU** parapgraphe-2] { nom-de-donnée / Entier } **TIMES**

Nom-de-donnée (ou entier) représente le nombre de fois que le ou les paragraphes désignés par le PERFORM doivent être exécutés successivement avant de revenir à l'instruction suivant le PERFORM.

```
PERFORM   X   TIMES
              Instructions COB
END-PERFORM
```

Avec X comme variable ou constante numérique entière.

Exemple: PERFORM P3 5 TIMES.

Format 3 : **PERFORM** paragraphe-1 [**THRU** parapgraphe-2] **UNTIL** condition

PERFORM WITH TEST { **BEFORE** / **AFTER** } **UNTIL** condition
Instructions COBOL

END-PERFORM

Exemple : PERFORM CALCUL UNTIL ECART < 0.01

```
CALCUL.
        MOVE X TO X0.
        COMPUTE X =  (X0 + (A / X0)) / 2
        SUBTRACT X0 FROM X GIVING ECART.
```

OU

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

Le paragraphe ou les instructions seront exécutées jusqu'à ce que la condition soit satisfaite, en option on peut choisir d'effectuer le test de la condition avant de pénétrer dans la boucle ou en sortie de la boucle.

Format 4 : **PERFORM** paragraphe-1 [**THRU** paragraphe-n]

VARYING nom-donnée-1 **FROM** i **BY** j **UNTIL** condition-1

AFTER nom-donnée-2 **FROM** m **BY** n **UNTIL** condition-2

AFTER nom-donnée-3 **FROM** p **BY** q **UNTIL** condition-3

PERFORM WITH TEST {**BEFORE** / **AFTER**}

VARYING nom-donnée-1 **FROM** i **BY** j **UNTIL** condition-1

AFTER nom-donnée-2 **FROM** m **BY** n **UNTIL** condition-2

AFTER nom-donnée-3 **FROM** p **BY** q **UNTIL** condition-3

Instructions COBOL

END-PERFORM

Exemple : PERFORM TOUR WITH TEST AFTER VARYING J FROM 1 BY 1 UNTIL J > 10.

Dès que la condition J > 10 est satisfaite, le traitement passe à l'instruction qui suit le PERFORM, il en va de même pour le PERFORM sans paragraphe.

PERFORM VARYING I FROM –100 TO 100 BY 5 UNTIL I > 100

PERFORM RECHERCHE VARYING I FROM 1 BY 1 UNTIL I > 25
AFTER K FROM 1 BY 1 UNTIL K > 25
AFTER M FROM 1 BY 1 UNTIL M > 25.

Le Cobol autorise les indices négatifs.

Avec AFTER on considère qu'il y a 3 boucles imbriquées dont la dernière tourne le plus rapidement, lorsque sa condition est remplie, on remonte d'un niveau et la boucle inférieure recommence et ainsi de suite jusqu'à ce que toutes les boucles atteignent leur condition d'arrêt.

Pour mettre un tableau à 2 dimensions à ZERO.

```
01  TABLEAU.
    02  LIGNE  OCCURS 20.
        03  COLL OCCURS 20 PIC 99.

PROCEDURE DIVISION.
DEB.
        PERFORM VARYING J FROM 1 BY 1 UNTIL J >20
                    AFTER   K FROM 1 BY 1 UNTIL K > 20
                      MOVE 0 TO COLL( J,   K)
        END-PERFORM.
```

Dans ce genre de PERFORM et selon les besoins du programmeur, on peut utiliser soit 1, soit 2 soit 3 indices de boucles.

Cette instruction peut également utiliser des valeurs INDEX plutôt que des variables numériques, ce qui en fait une instruction de choix pour la manipulation des tableaux avec indices ou index.

Exemple:

```
01 TABLE.
    02  NOM  OCCURS 200  PIC X(25) INDEXED BY J.

PROCEDURE DIVISION.
DEBUT.
   PERFORM VARYING J FROM 1 BY 1 UNTIL J > 200
                MOVE SPACES TO NOM(J)
    END-PERFORM.
```

11. L'INSTRUCTION EXIT

Format : Nom-de-paragraphe. EXIT.

Cette instruction sert à fournir un point unique de fin d'une série de paragraphes appelés par PERFORM. Ce paragraphe sert de délimiteur de fin et peut donc être utilisé par un GO TO éventuel à l'intérieur de ce bloc.

Le nom-de-paragraphe sera donc placé après l'option THRU du PERFORM.

Exemple :

```
PERFORM P1 THRU P2.

P1.
    IF A > B GO TO P2.
P2.  EXIT.
```

Si la logique du programme demande de sortir de P1 avant la fin de l'ensemble des instructions, le fait de passer via P2 ne rompt pas les liens de notre PERFORM.

12. L'INSTRUCTION STOP

Format 1 : **STOP RUN**

Le format 1 indique la fin du traitement dans un programme puis à rendre le contrôle au superviseur. Il peut y avoir plusieurs STOP RUN dans un même programme mais en programmation structurée un seul est suffisant.

13. L'INSTRUCTION SET

Format 1 :

SET index-name-1 [index-name-2] …. **TO** { index-name-3 / nombre entier }

Un nombre entier est un entier numérique > 0 et non signé.
Les index-names sont définis par des clauses INDEXED BY.

Cette instruction signifie

METTRE nom-index-1 A LA VALEUR CORRESPONDANT A { index-name-3 / nombre entier }

Nombre entier représente un numéro d'occurrence de l'élément qu'on désire repérer.

Exemple : SET INDEX1 TO 5, signifie que nous voulons utiliser le 5ème élément de la table. Ce n'est pas la valeur 5 que nous retrouverons dans INDEX1 mais l'adresse correspondant au déplacement de l'élément par rapport au début de la table.
Lorsque index-name-3 est utilisé, il a ajustement de index-name-1 index-name-2 … sur index-name-3 et COBOL traduit le contenu de index-name-3 en occurrence avant de calculer les valeurs de index-1 index-2.

```
01  TABLE.
    02  T1  OCCURS 20 INDEXED BY IND1.
        03 T2  OCCURS 30 INDEXED BY IND2.
              04  T3 OCCURS  40 INDEXED BY IND3 PIC 9.
```

Pour utiliser T3 (5,6,5),
```
SET IND3 TO 5
SET IND1 TO IND3
SET  IND2 TO 6
```

Format 2 : **SET** index-name-1 [index-name-2] … { **UP BY** / **DOWN BY** } { nombre entier / nom-index-3 }

Cette instruction permet avec UP BY d'augmenter ou de diminuer DOWN BY la valeur de index-name-1 et de index-name-2 d'une valeur de nombre ou de index-3. INDEX-3 ne doit pas appartenir à la table de index-1 ou index-2 car le COBOL traduit sa valeur en nombre d'occurrences.

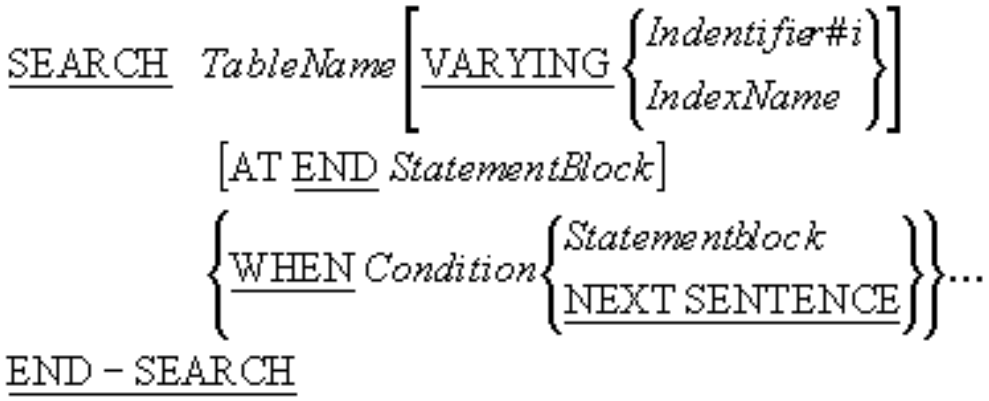
Il est à prévoir que durant l'exécution du programme nous devions retenir une valeur particulière d'index. Dans ce cas nous serons amenés à déclarer une zone particulière d'usage réservé aux INDEX.

01 INDEX-DE-RESERVE USAGE IS INDEX.

Cette clause est utilisée à n'importe quel niveau en DATA DIVISION. Il ne sera pas fait usage de la clause PICTURE puisque cette donnée contiendra une adresse pour notre index.
Lorsque cette clause est spécifiée pour un groupe, ce sont toutes les zones élémentaires du groupe qui sont considérées comme d'usage index.

Pour sauvegarder la valeur de IND2, SET INDEX-DE-RESERVE TO IND2
Pour lui retourner sa valeur plus tard , SET IND2 TO INDEX-DE-RESERVE

14. L'INSTRUCTION SEARCH



Cette instruction permet de rechercher en table un élément qui satisfasse à une ou plusieurs conditions données.

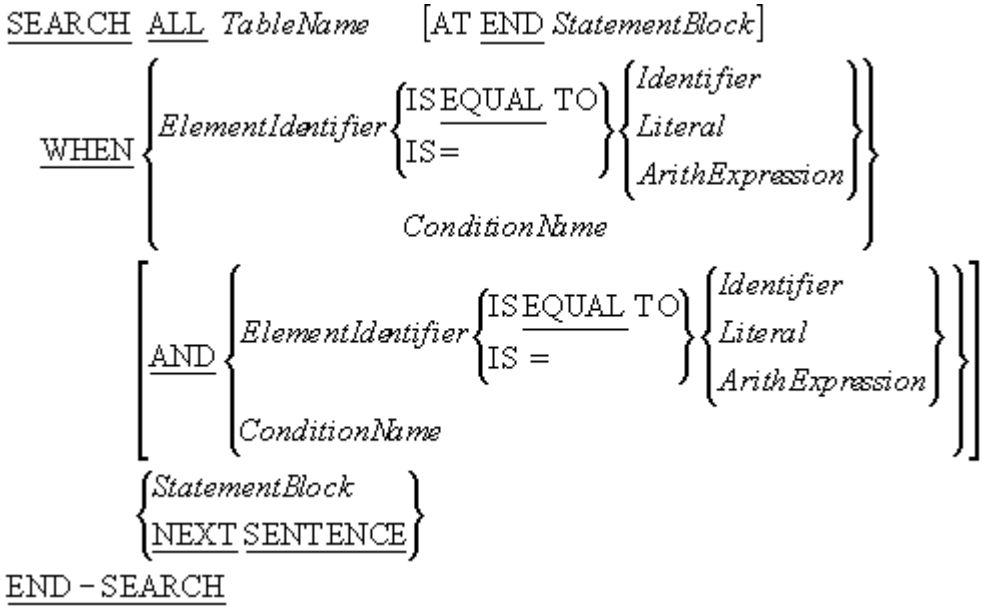
```
IDENTIFICATION DIVISION.
PROGRAM-ID. LetterSearch.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LetterTable.
   02 LetterValues.
      03 FILLER PIC X(26)
         VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
   02 FILLER REDEFINES LetterValues.
      03 Letter PIC X OCCURS 26 TIMES      INDEXED BY LetterIdx.
01 LetterIn      PIC X.
01 LetterPos     PIC 99.
01 PrnPos        PIC Z9.
```

```
PROCEDURE DIVISION.
Begin.
    DISPLAY "Votre lettre svp "          WITH NO ADVANCING
    ACCEPT LetterIn
    SET LetterIdx   LetterPos   TO   1
    SEARCH Letter VARYING LetterPos
      AT END DISPLAY "Letter " LetterIn " not found!"
      WHEN Letter(LetterIdx) = LetterIn
          MOVE LetterPos TO PrnPos
          DISPLAY LetterIn, " is in position ", PrnPos
    END-SEARCH


 IDENTIFICATION DIVISION.
PROGRAM-ID. TaxTable.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT TaxFile ASSIGN TO "CountyTaxes.DAT"    ORGANIZATION IS LINE SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD TaxFile.
01 TaxRec.
   88 EndOfTaxFile   VALUE HIGH-VALUES.
   02 PAYENum        PIC 9(8).
   02 County         PIC X(9).
   02 TaxPaid        PIC 9(7)V99.
WORKING-STORAGE SECTION.
01 CountyTaxTable.
   02 CountyTaxDetails OCCURS 26 TIMES.
      03 CountyTax   PIC 9(8)V99.
      03 PayerCount  PIC 9(7).
         88 NoOnePaidTax VALUE ZEROS.

01 Idx               PIC 99.
01 CountyTaxLine.
   02 PrnCounty      PIC X(9).
   02 FILLER         PIC X(7) VALUE " Tax = ".
   02 PrnTax         PIC ZZZ,ZZZ,ZZ9.99.
   02 FILLER         PIC X(12) VALUE "   Payers = ".
   02 PrnPayers      PIC Z,ZZZ,ZZ9.
```

```
01 CountyNameTable.
   02 TableValues.
      03 FILLER  PIC X(9)  VALUE "Carlow".
      03 FILLER  PIC X(9)  VALUE "Cavan".
      03 FILLER  PIC X(9)  VALUE "Clare".
      03 FILLER  PIC X(9)  VALUE "Cork".
      03 FILLER  PIC X(9)  VALUE "Donegal".
      03 FILLER  PIC X(9)  VALUE "Dublin".
      03 FILLER  PIC X(9)  VALUE "Galway".
      03 FILLER  PIC X(9)  VALUE "Kerry".
      03 FILLER  PIC X(9)  VALUE "Kildare".
      03 FILLER  PIC X(9)  VALUE "Kilkenny".
      03 FILLER  PIC X(9)  VALUE "Laois".
      03 FILLER  PIC X(9)  VALUE "Leitrim".
      03 FILLER  PIC X(9)  VALUE "Limerick".
      03 FILLER  PIC X(9)  VALUE "Longford".
      03 FILLER  PIC X(9)  VALUE "Louth".
      03 FILLER  PIC X(9)  VALUE "Mayo".
      03 FILLER  PIC X(9)  VALUE "Meath".
      03 FILLER  PIC X(9)  VALUE "Monaghan".
      03 FILLER  PIC X(9)  VALUE "Offaly".
      03 FILLER  PIC X(9)  VALUE "Roscommon".
      03 FILLER  PIC X(9)  VALUE "Sligo".
      03 FILLER  PIC X(9)  VALUE "Tipperary".
      03 FILLER  PIC X(9)  VALUE "Waterford".
      03 FILLER  PIC X(9)  VALUE "Westmeath".
      03 FILLER  PIC X(9)  VALUE "Wexford".
      03 FILLER  PIC X(9)  VALUE "Wicklow".
   02 FILLER REDEFINES TableValues.
      03   CountyName  PIC X(9)
                   OCCURS 26 TIMES
                   INDEXED BY CountyIdx.
PROCEDURE DIVISION.
Begin.
   OPEN INPUT TaxFile
   MOVE ZEROS TO CountyTaxTable
   READ TaxFile       AT END SET EndOfTaxFile TO TRUE
   END-READ
   PERFORM UNTIL EndOfTaxFile
      SET CountyIdx Idx TO 1
```

```
    SEARCH CountyName VARYING Idx
      AT END DISPLAY   County " not found"
      WHEN CountyName(CountyIdx) = County
          ADD TaxPaid TO CountyTax(Idx)
          ADD 1 TO PayerCount(Idx)
    END-SEARCH
    READ TaxFile    AT END SET EndOfTaxFile TO TRUE
    END-READ
  END-PERFORM
  PERFORM DisplayCountyTaxes VARYING Idx FROM 1 BY 1
       UNTIL Idx GREATER THAN 26
  CLOSE TaxFile
  STOP RUN.
DisplayCountyTaxes.
  IF NOT NoOnePaidTax(Idx)
    MOVE CountyName(Idx) TO PrnCounty
    MOVE CountyTax(Idx)  TO PrnTax
    MOVE PayerCount(Idx) TO PrnPayers
    DISPLAY CountyTaxLine
  END-IF.
```

Second format du SEARCH



La clause **ASCENDING** ou **DESCENDING KEY doit être utilisée dans la déclaration de la table,** car la condition-1 doit faire intervenir un des arguments de classement (KEY) liés à élément1.

Exemple : 02 MAISON OCCURS 250 ASCENDING KEY PROVINCE VILLE
INDEXED BY I.

04 PROVINCE PIC 99.
04 VILLE PIC 999.
04 RUE PIC X(30).

Le fait d'utiliser VILLE dans condition-1 oblige l'utilisation de PROVINCE comme argument de classement.
On peut dans ce cas écrire **WHEN PROVINCE = 3 AND VILLE = 112 PERFORM P3**.

La condition-1 ne peut faire intervenir que des signes égal **(=)** et seulement des opérations logiques **AND** dans le cas d'une condition composée**.**
Le **SEARCH ALL** est destiné à l'exploration complète de la table, il n'est donc pas nécessaire de positionner l'index à 1, le COBOL s'en charge.

Exemple :

```
CLIENT.
02  ENREG  OCCURS  500 ASCENDING KEY  BANQUE SERVICE
                      INDEXED BY I.
     04  BANQUE    PIC 99.
     04  SERVICE    PIC 9999.
     04  COMPTE    PIC 9999.
     04  NOM           PIC X(22).

     SEARCH ALL ENREG AT END PERFORM INCONNU
               WHEN BANQUE(I) = BQUE  AND  SERVICE(I) = SERV
                         MOVE COMPTE(I) TO NUMERO.
     END-SEARCH


01  LetterTable.
    02 LetterValues.
       03 FILLER PIC X(26)
          VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
    02 FILLER REDEFINES LetterValues.
       03 Letter PIC X OCCURS 26 TIMES
                       ASCENDING KEY IS Letter
                       INDEXED BY LetterIdx.

SEARCH ALL Letter
  AT END DISPLAY "Letter " SearchLetter " was not found!"
  WHEN Letter(LetterIdx) = SearchLetter
       SET LetterPos TO LetterIdx
       DISPLAY SearchLetter " is in position " LetterPos
END-SEARCH.
```

15. LES DECLARATIVES

Cobol a prévu un traitement particuliers en cas d'erreurs des E/S sur les fichiers. Ces traitements doivent être réalisés sous forme de séquences indépendantes regroupées dans une section de la PROCEDURE DIVISION sous le nom de DECLARATIVES.

Les DECLARATIVES sont importantes dans le cas où le programmeur n'aurait pas prévu de traitement dans le cas des erreurs d'E/S. En effet le système d'exploitation de l'ordinateur mettra fin à l'exécution du programme.

Forme générale :

```
PROCEDURE DIVISION.
DECLARATIVES.
Nom-de-section SECTION. USE …….
Nom-de-paragraphe.

        Instructions Cobol

END DECLARATIVES.
```

Les DECLARATIVES commencent toujours par le mot DECLARATIVES, immédiatement après PROCEDURE DIVISION, sont toujours dans une SECTION et se terminent toujours par END DECLARATIVES.

FORMAT :

```
Nom-de-section SECTION. USE AFTER STANDARD ERROR
                          { Nom-de-fichier-1 nom-de-fichier-2 … }
PROCEDURE ON              { INPUT                               }
                          { OUTPUT                              }
                          { I-O                                 }
```

Si nom-de-fichier est spécifié les DECLARATIVES ne s'appliquent que pour ce ou ces fichiers.

INPUT indique que la section DECLARATIVES sera utilisée pour tous les fichiers ouverts en lecture.

OUTPUT indique que la section DECLARATIVES sera utilisée pour tous les fichiers ouverts en écriture.

I-O indique que la section DECLARATIVES sera utilisée pour tous les fichiers ouverts en lecture et en écriture.
L'instruction USE indique que le branchement n'a lieu qu'après que les routines du système d'exploitation aient été effectuées.

L'option AFTER peut être remplacée par BEFORE.

15. LES SOUS-PROGRAMMES EXTERNES

Le branchement du traitement depuis le programme principal COBOL vers le sous-programme est réalisé par l'instruction CALL.

CALL 'Nom du sous programme' USING nom de donnée1 [nom de donnée2] …

CALL 'Nom du point d'entrée' USING nom de donnée1 [nom de donnée2] …

'Nom du sous programme' : désigne le nom du sous-programme appelé, nom doit être de maximum 8 caractères de long.

'Nom du point d'entrée' : désigne une adresse d'entrée dans un sous-programme, son nom doit être différent de celui du sous-programme et doit correspondre à un **ENTRY.**

Exemple : CALL "SOUSPROG" USING ZON1 ZON2.

CALL "ENTREE" USING ZON1

LE SOUS-PROGRAMME COBOL

Le sous-programme en COBOL est construit comme tout programme COBOL avec ses 4 Divisions. La différence avec le programme principal réside dans l'introduction de clauses USING et de la LINKAGE SECTION nécessaire à l'établissement d'une liaison entre les variables du programme principal et du sous-programme.

Cette correspondance est établie au moyen du USING (sous-programme) et du CALL (programme).

La clause USING dans le sous-programme :

PROCEDURE DIVISION USING nom-de-donnée-1 (nom-de-donnée-2) …..

Au moment de l'appel du sous-programme la correspondance entre les nom-de-donnée-1, nom-de-donnée-2 … est établie et le sous-programme démarre en utilisant les valeurs transférées par le programme principal.

Si l'entièreté du sous-programme ne doit pas être utilisée, nous disposons d'une option ENTRY pour indiquer au sous-programme l'endroit ou le sous-programme doit débuter.

Format : **ENTRY** "nom-symbolique" **USING** nom-de-donnée-1 (nom-de-donnée-2) ….

Le même système opératoire sera activé et les nom-de-donnée-1 nom-de-donnée-2… seront mis en correspondance.

LA LINKAGE SECTION.

Le programme principal et le sous-programme COBOL étant compilés séparément, il est nécessaire que la description des zones de données communes apparaisse dans les DATA DIVISION respectives des programmes.

Mais il est exclu de réserver 2 fois ces zones en mémoire centrale, c'est pourquoi une LINKAGE SECTION a été créée en DATA DIVISION.

Une zone de données commune "TRANSFERT" sera donc décrite normalement dans le programme principal soit en FILE SECTION soit en WORKING-STORAGE SECTION, et sa correspondance dans le sous-programme en LINKAGE SECTION.

L'ordre d'appel de sous-programmes CALL, aura donc pour effet d'attribuer les mêmes adresses mémoire à ces deux zones de données.

Remarque : rien n'oblige le programmeur à utiliser des descriptions de zones rigoureusement identiques entre le programme principal et le sous-programme. Les noms ne doivent pas être les mêmes ainsi que les descriptions de données.

La terminaison d'un sous-programme :

Dans un sous-programme, la terminaison est réalisée par GOBACK

16. LE TRI - SORT

Il arrive souvent dans le traitement de l'information que les enregistrements soient présentés dans un ordre quelconque. Lorsque chaque enregistrement est traité seul, indépendamment de ceux qui le suivent et le précédent cela ne pose pas de problèmes, à moins qu'un contrôle particulier ne doit être effectué ou que les résultats ne doivent être présentés dans un ordre bien précis.

Il est dans ce cas nécessaire de trier ces enregistrements, càd de les présenter dans un ordre particulier. Par exemple, un fichier du personnel est souvent trié par ordre alphabétique, et en cas de doublon sur le nom par ordre croissant sur le prénom et dans le cas ou le prénom est le même sur le numéro de matricule.

Pour effectuer un tel tri on utilise 3 informations différentes contenues dans chaque enregistrement du fichier (le nom – le prénom – le matricule).

Nous dans cet exemple le besoin d'utiliser 3 clés, le nom sera alors considéré comme la clé majeure et le matricule comme la clé mineure.

En triant par ordre alphabétique, on trie par ordre croissant (ASCENDING). On pourrait de la même façon trier en ordre décroissant (DESCENDING) le résultat serait que ZORRO serait placé avant ALEXANDRE.

Ce tri peut se faire à l'aide d'un programme écrit par l'utilisateur, mais il existe un ensemble de programmes appelé SORT qui permet de faire le TRI des fichiers.

(Il est inconcevable de placer l'entièreté d'un fichier dans la mémoire de l'ordinateur et de le trier, pour cela il faut que le fichier soit très petit, ce qui n'est pas souvent le cas)

Le SORT nécessite la description d'un fichier particulier le fichier de tri

```
SORT sort-file
                    { ON {ASCENDING } KEY {key-name} } ...
                       {DESCENDING}

                    { ON {ASCENDING } KEY {key-name} } ...
                       {DESCENDING}

                     { INPUT PROCEDURE IS proc-name  }
                    { USING {in-file} ...          }

                    { OUTPUT PROCEDURE IS proc-name }
                    { GIVING {out-file} ...        }
```

```
SD  WorkFile.
01  WorkRec.
    02  WSalesmanNum     PIC 9(5).
    02  FILLER           PIC X(5).
```

SD signifie fichier TRI il est associé à SELECT **WorkFile ASSIGN TO DISK.**
Ne nécessite aucune description particulière est ouvert et fermé par le système.

Forme Simple utilisant un fichier à trier le USING et un fichier résultat le Giving, passant par un intermédiaire le WORKFILE

```
SORT WorkFile ON ASCENDING KEY WSalesmanNum
           USING SalesFile
           GIVING SortedSalesFile.
```

Dans une seconde version on désire ne traiter qu'un certain type de records du fichier à trier dans ce cas l'input procédure peut extraire les enregistrements désirés.
Pour écrire dans le fichier de tri on dispose d'une instruction

RELEASE nom de l'enregistrement from nom de données

Cette instruction est équivalente à WRITE

Exemple

```
FD  SalesFile.
01  SalesRec.
    88  EndOfSales      VALUE HIGH-VALUES.
    02  FILLER      PIC 9(5).
    02  FILLER      PIC X.
        88  HatRecord   VALUE "H".
    02  FILLER      PIC X(4).

SD  WorkFile.
01  WorkRec.
    02  WSalesmanNum     PIC 9(5).
    02  FILLER      PIC X(5).

FD  SortedSalesFile.
01  SortedSalesRec.
    02  SalesmanNum PIC 9(5).
    02  ItemType         PIC X.
    02  QtySold          PIC 9(4).
PROCEDURE DIVISION.
Begin.

     SORT WorkFile ON ASCENDING KEY WSalesmanNum
         INPUT PROCEDURE IS SelectHatSales
         GIVING SortedSalesFile.
```

```
SelectHatSales.
     OPEN INPUT InFileName
     READ InFileName RECORD
     PERFORM UNTIL READ-EOF
             IF CONDITION
                  RELEASE SDWorkRec
             END-IF
             READ InFileName RECORD
     END-PERFORM
     CLOSE InFile.
```

Dans le cas ou un travail doit être éffectué sur le fichier trié on dispose d'ine instruction capable d'aller lire dans le fichier de travail.

RETURN nom du fichier de tri INTO nom de donnée AT END instruction de fin

Pour pouvoir l'utiliser il faudra créer une **OUTPUT PROCEDURE**

```
FD  SalesFile.
01  SalesRec            PIC X(10).
SD  WorkFile.
01  WorkRec.
    88  EndOfWorkFile   VALUE HIGH-VALUES.
    02  WSalesmanNum     PIC 9(5).
    02  FILLER      PIC X.
    02  WQtySold        PIC X(4).
FD  SalesSummaryFile.
01  SummaryRec.
    02  SalesmanNum PIC 9(5).
    02  TotalQtySold     PIC 9(6).
PROCEDURE DIVISION.
Begin.
    SORT WorkFile ON ASCENDING KEY WSalesmanNum
         USING SalesFile
         OUTPUT PROCEDURE IS SummariseSales.
SummariseSales.
   OPEN OUTPUT SalesSummaryFile
   RETURN WorkFile
       AT END SET EndOfWorkFile TO TRUE
   END-RETURN
   PERFORM UNTIL EndOfWorkFile
      MOVE WSalesmanNum TO SalesmanNum
      MOVE ZEROS TO TotalQtySold
      PERFORM UNTIL WSalesManNum NOT = SalesmanNum
                    OR EndOfWorkFile
         ADD WQtySold TO TotalQtySold
         RETURN WorkFile AT END SET EndOfWorkFile TO TRUE
         END-RETURN
      END-PERFORM
      WRITE SummaryRec
   END-PERFORM
   CLOSE SalesSummaryFile.
```

17. LES FICHIERS INDEXE-SEQUENTIEL

FORMAT GENERAL

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FileName
ASSIGN TO FileSpec
[ORGANIZATION IS] INDEXED
[ACCESS MODE IS { SEQUENTIAL | *RANDOM* | *DYNAMIC* }]
[RECORD KEY IS UniqueRecKey]
[ALTERNATE RECORD KEY IS AltKey [WITH DUPLICATES]]
[FILE STATUS IS FileStatus]

FD FileName
[IS EXTERNAL]
[IS GLOBAL]
[BLOCK CONTAINS BlockSize { RECORDS | CHARACTERS }]
[VALUE OF ID IS FileSpec]

OPEN { INPUT | OUTPUT | I-O | EXTEND } FileName …

READ FileName RECORD [INTO DestItem]
[KEY IS KeyName]
[INVALID KEY StatementBlock]
[*END – READ*]

READ FileName NEXT RECORD [INTO DestItem]
[AT END StatementBlock]
[*END – READ*]

L'option NEXT peut être remplacée par le mot PREVIOUS pour une lecture vers l'avant.

WRITE RecName [FROM SourceItem]
[INVALID KEY StatementBlock]
[*END – WRITE*]

REWRITE RecName [FROM SourceItem]
[INVALID KEY StatementBlock]
[*END – REWRITE*]

DELETE FileName RECORD
[INVALID KEY StatementBlock]
END – DELETE

START FileName [KEY { IS EQUAL TO | IS = | IS GREATER THAN | IS > | IS NOT LESS THAN | IS NOT < } KeyName]
[INVALID KEY StatementBlock]
END – START

Le mode d'accès SEQUENTIAL est réservé au traitement séquentiel, il permet

- la création séquentielle du fichier
- la lecture séquentielle du fichier
- le traitement séquentielle en lecture-écriture du fichier avec modifications ou suppressions d'enregistrements

Le mode RANDOM est réservé au traitemnt directe

Le mode DYNAMIC permet à la fois l'accès séquentiel ou direct au fichier.

Création d'un fichier indexé

```
IDENTIFICATION DIVISION.
PROGRAM-ID. IndexedFromSeq.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT OPTIONAL VideoFile ASSIGN TO "VIDEO.DAT"
          ORGANIZATION IS INDEXED
          ACCESS MODE  IS RANDOM
          RECORD KEY   IS VideoCode
          ALTERNATE
          RECORD KEY   IS VideoTitle
                          WITH DUPLICATES
          FILE STATUS  IS VideoStatus.
   SELECT SeqVideoFile ASSIGN TO "INVIDEO.DAT".
DATA DIVISION.
FILE SECTION.
FD VideoFile.
01 VideoRecord.
   02 VideoCode                PIC 9(5).
   02 VideoTitle               PIC X(40).
   02 VideoSupplierCode        PIC 99.
FD SeqVideoFile.
01 SeqVideoRecord.
   88 EndOfFile                VALUE HIGH-VALUES.
   02 SeqVideoCode             PIC 9(5).
   02 SeqVideoTitle            PIC X(40).
   02 SeqVideoSupplierCode     PIC 99.
WORKING-STORAGE SECTION.
01   VideoStatus                PIC X(2).
PROCEDURE DIVISION.
Begin.
   OPEN INPUT SeqVideoFile,
        OUTPUT VideoFile.
   READ SeqVideoFile
      AT END SET EndOfFile TO TRUE
   END-READ.
   PERFORM UNTIL EndOfFile
      WRITE VideoRecord FROM SeqVideoRecord
         INVALID KEY DISPLAY "VIDEO STATUS :- ", VideoStatus
      END-WRITE
      READ SeqVideoFile
         AT END SET EndOfFile TO TRUE
      END-READ
   END-PERFORM.
   CLOSE VideoFile, SeqVideoFile.
   STOP RUN.
```

```
SELECT FICHIER-MAITRE
                ASSIGN TO 'maitre.dat'
                ORGANIZATION IS INDEXED
                ACCESS IS RANDOM
                RECORD KEY IS numero-maitre.
```

Ajouter un enregistrement

```
    MOVE numero-trans to numero-maitre.
    READ fichier-maitre
        INVALID KEY
            MOVE enr-trans to enr-maitre
            WRITE enr-maitre
        NOT  INVALID KEY
            traiter-erreur
    END-READ.
```

Modifier un enregistrement

```
    MOVE numero-trans to numero-maitre.
    READ fichier-maitre
        INVALID KEY traiter-erreur
        NOT INVALID KEY
            MOVE enr-trans to enr-maitre
            REWRITE enr-maitre
    END-READ.
```

Supprimer un enregistrement MOVE numero-trans to numero-maitre.

```
    READ fichier-maitre
        INVALID KEY traiter-erreur
        NOT INVALID KEY
            DELETE fichier-maitre RECORD
    END-READ.
```

Lecture d'un fichier indexé - séquentiellement

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ReadIndexedFile.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT VideoFile ASSIGN TO "VIDEO.DAT"
          ORGANIZATION IS INDEXED
          ACCESS MODE  IS DYNAMIC
          RECORD KEY   IS VideoCode
          ALTERNATE
          RECORD KEY   IS VideoTitle
                       WITH DUPLICATES
          FILE STATUS  IS VideoStatus.
DATA DIVISION.
FILE SECTION.
FD VideoFile
01 VideoRecord.
   88 EndOfFile VALUE HIGH-VALUE.
   02 VideoCode               PIC 9(5).
   02 VideoTitle              PIC X(40).
   02 SupplierCode            PIC 99.
WORKING-STORAGE SECTION.
01 VideoStatus                PIC X(2).
01 RequiredSequence            PIC 9.
   88 VideoCodeSequence       VALUE 1.
   88 VideoTitleSequence      VALUE 2.
01 PrnVideoRecord.
   02 PrnVideoCode            PIC 9(5).
   02 PrnVideoTitle           PIC BBBBX(40).
   02 PrnSupplierCode         PIC BBBB99.

PROCEDURE DIVISION.
Begin.
   OPEN INPUT VideoFile.

   DISPLAY "Enter key : 1=VideoCode, 2=VideoTitle ->"
      WITH NO ADVANCING.
   ACCEPT RequiredSequence.

   IF VideoTitleSequence
      MOVE SPACES TO VideoTitle
      START VideoFile KEY IS GREATER THAN VideoTitle
         INVALID KEY  DISPLAY "VIDEO STATUS :- ", VideoStatus
      END-START
   END-IF

   READ VideoFile NEXT RECORD
      AT END SET EndOfFile TO TRUE
   END-READ.
```

```
   PERFORM UNTIL EndOfFile
      MOVE VideoCode TO PrnVideoCode
      MOVE VideoTitle TO PrnVideoTitle
      MOVE SupplierCode TO PrnSupplierCode
      DISPLAY  PrnVideoRecord
      READ VideoFile NEXT RECORD
         AT END SET EndOfFile TO TRUE
      END-READ
   END-PERFORM.
   CLOSE VideoFile.
   STOP RUN.
```

Lecture d'un fichier indexé - directement

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ReadingIndexedFile.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT VideoFile ASSIGN TO "VIDEO.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS VideoCode
      ALTERNATE RECORD KEY IS VideoTitle
                WITH DUPLICATES
      FILE STATUS IS VideoStatus.
DATA DIVISION.
FILE SECTION.
FD VideoFile.
01 VideoRecord.
   02 VideoCode          PIC 9(5).
   02 VideoTitle         PIC X(40).
   02 SupplierCode       PIC 99.
WORKING-STORAGE SECTION.
01 VideoStatus           PIC X(2).
   88  RecordFound       VALUE "00".
01 RequiredKey           PIC 9.
   88 VideoCodeKey       VALUE 1.
   88 VideoTitleKey      VALUE 2.
01 PrnVideoRecord.
   02 PrnVideoCode       PIC 9(5).
   02 PrnVideoTitle      PIC BBBBX(40).
   02 PrnSupplierCode    PIC BBBB99.

PROCEDURE DIVISION.
Begin.
   OPEN INPUT VideoFile.
   DISPLAY "Chose key VideoCode = 1,  VideoTitle = 2 ->  "
                     WITH NO ADVANCING.
   ACCEPT RequiredKey.
```

```
IF VideoCodeKey
   DISPLAY "Enter Video Code (5 digits) -> " WITH NO ADVANCING
   ACCEPT VideoCode
   READ VideoFile
      KEY IS VideoCode
      INVALID KEY  DISPLAY "VIDEO STATUS :- ", VideoStatus
   END-READ
END-IF
IF VideoTitleKey
   DISPLAY "Enter Video Title (40 chars) -> " WITH NO ADVANCING
   ACCEPT VideoTitle
   READ VideoFile KEY IS VideoTitle
      INVALID KEY  DISPLAY "VIDEO STATUS :- ", VideoStatus
   END-READ
END-IF
IF RecordFound
   MOVE VideoCode TO PrnVideoCode
   MOVE VideoTitle TO PrnVideoTitle
   MOVE SupplierCode TO PrnSupplierCode
   DISPLAY  PrnVideoRecord
END-IF.
CLOSE VideoFile.
STOP RUN.
```

Exemple de MAJ séquentiel d'un I/S sans utilisation des clés.

```
SELECT STIS ASSIGN TO DISK
        ORGANIZATION IS INDEXED ACCESS MODE IS DYNAMIC
         RECORD KEY IS NUMART.

FD……

PROCEDURE DIVISION.
      OPEN I-O STIS.
      START STIS KEY GREATER THAN NUMART.
      READ STIS NEXT RECORD AT END MOVE 1 TO FIN.
      …
      REWRITE ENR-STIS.
      …
      CLOSE STIS.
```

Dans la clause **RECORD KEY** IS nom-de-clé
ALTERNATE RECORD KEY IS nom-de-donnée
WITH DUPLICATES.

Cette clause permet de définir des clés secondaires, elles doivent également se trouver dans la déclaration de l'enregistrement du fichier. Contrairement à la clé primaire qui doit toujours être unique, la clé secondaire peut avoir plusieurs fois la même valeur si le ***WITH DUPLICATES*** est utilisé. Au niveau du système un index secondaire est créé dans le fichier "KEY".

```
SELECT STIS ASSIGN TO DISK
                    ORGANIZATION IS INDEXED
                    ACCESS MODE IS DYNAMIC
                     RECORD KEY IS NUMFACT
                    ALTERNATE RECORD KEY IS  NUMCLI WITH DUPLICATES.

FD STIS.
01  ENREG.
        02 NUMFACT  PIC 9999.
        02 NUMCLI     PIC 9999.
        02  NOM         PIC XXXXXXXXX.

PROCEDURE DIVISION.

        OPEN INPUT STIS
        MOVE MACLE TO NUMCLI
        START STIS KEY EQUAL NUMCLI
                INVALID KEY DISPLAY  MESERREUR
                NOT INVALID KEY
                        PERFORM UNTIL EOF=1
                                READ STIS NEXT INTO ZONE AT END MOVE 1 TO EOF
                                NOT AT END
                                        IF NUMCLI = MACLE THEN MOVE …

                                        END-IF
```

REMARQUE

LE FICHIER DOIT ETRE OUVERT EN INPUT.