CHAPITRE I

INTRODUCTION AU LANGAGE COBOL

****

Le langage COBOL fut conçu en 1957, sur demande du gouvernement des U.S.A. L'étude fut développée par un comité d'utilisateurs et de constructeurs d'ordinateur.

COBOL = **C**ommon **O**rganization **B**usiness **O**riented **L**anguage

1ère étape achevée en 1959 : Système COBOL, langage.

* Basé sur l'anglais

* Extrêmement structuré en

  DIVISIONS
  SECTIONS
  PARAGRAPHES
  PHRASES
  VERBES, NOMS, OPERATEURS, SIGNES DE PONCTUATION

D'autres normes viendront corriger, améliorer ce premier travail :

1968 : COBOL A.N.S. (**A**merican **N**ational **S**tandard)
1974 (Ajouts sur 1968 : sous programmes, tri intégré)
1985.
.

Les règles de syntaxe associées à ce langage sont relativement rigides et l'une de ses difficultés est donc d'apprendre à les maîtriser.

I-1 – STRUCTURE DE BASE D'UN PROGRAMME COBOL

Tout programme COBOL se compose de 4 divisions obligatoirement présentes (même vides):

IDENTIFICATION DIVISION
ENVIRONMENT DIVISION
DATA DIVISION
PROCEDURE DIVISION

A – "IDENTIFICATION DIVISION" :

Elle sert à identifier le programme et à renseigner l'utilisateur sur le nom de l'auteur, la date d'écriture,…

B – "ENVIRONMENT DIVISION" :

Contient une description sommaire de la configuration du calculateur utilisé ainsi que des informations relatives à la gestion des entrées-sorties, aux conditions spéciales de traitement des fichiers.

C – "DATA DIVISION" (très importante_ !)

Décrit la structure des informations que le programme reçoit, traite, ou produit. Les données sont groupées en deux catégories :

* celles appartenant aux fichiers manipulés par le programme
* celles qui sont créées de façon interne et placées dans des zones intermédiaires selon un format donné.

Cette DATA DIVISION comporte 3 sections (parfois 4)

1) FILE SECTION, définissant la structure des fichiers et des articles qu'ils contiennent.

2) WORKING-STORAGE SECTION : toutes les données internes autres que celles des fichiers.

3) "LINKAGE SECTION" : données communes à un programme et à un sous-programme.

D – "PROCEDURE DIVISION" (très importante)

Le programmeur y décrit le traitement à effectuer par le calculateur. C'est donc le lieu de la traduction de l'algorithme conçu par le programmeur. Elle est composée d'un ensemble de sections ou de paragraphes.

CHAPITRE II

NOTATIONS UTILISEES

DANS LES FORMATS ET REGLES

Un format général montre la disposition des éléments d'une clause ou d'une instruction.

* les **ELEMENTS** qui composent une clause ou une instruction sont les MOTS écrits en majuscules, ou en minuscules, les nombres niveaux, les crochets, les accolades et les caractères spéciaux.

(1) Les **MOTS**

Les mots en majuscules, soulignés, sont appelés MOTS-CLES et sont obligatoires. Les non-soulignés sont facultatifs pour l'utilisateur.

Les mots en minuscules sont des termes génériques utilisés pour représenter les mots utilisateurs, les littéraux,…

(2) Les **CROCHETS**

Une partie du format général, placée entre crochets, peut être incluse ou omise, au choix de l'utilisateur.

(3) **POINTS DE SUSPENSIONS (…)**

Ils peuvent indiquer l'omission d'une portion de programme origine (voir contexte). Dans les formats généraux, ils représentent un endroit où une répétition peut se produire.

(4) **Les CARACTERES** +, -, <, >, = apparaissant dans les formats sont obligatoires bien que non soulignés.

(4) Les ACCOLADES

Elles permettent le choix entre plusieurs options.

1) OPEN { **INPUT** **n-fich-1** **[WITH NO REVIND]** | **OUTPUT** **n-fich-2** **[WITH NO REWIND]** | **I-O** **n-fich-5** **[N-FICH-6]** | **EXTEND** **n-fich-7** **[N-FICH-8]** }

2) BLOCK CONTAINS ENTIER-1 [TO ENTIER-2 { RECORDS | CHARACTERS }]

CHAPITRE III – LES ELEMENTS DU LANGAGE

****

Le langage COBOL comme tout langage possède :

- une grammaire

 qui doivent être respectées au maximum.

- une orthographe

I – jeu de caractères COBOL

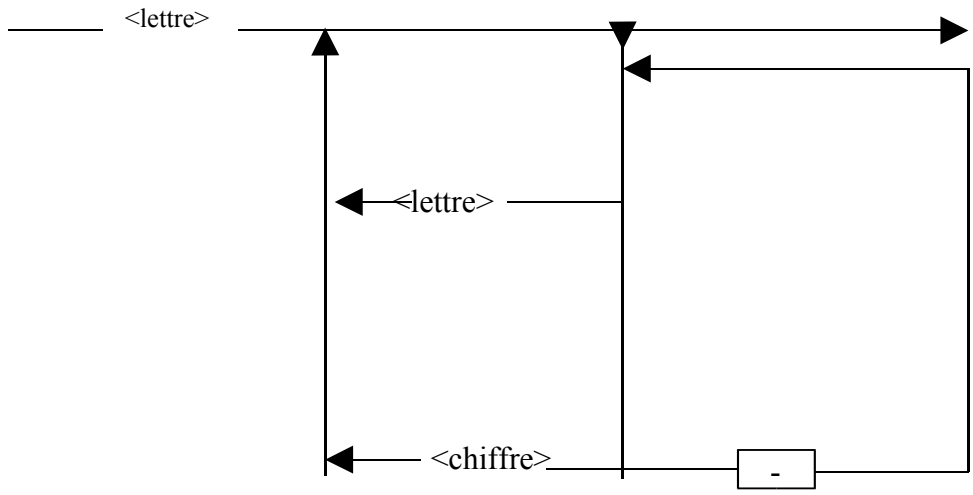
Les 51 caractères suivants sont admissibles :

0, 1, 2, --- 9	Chiffres
A, B, C, --- Z	Lettres majuscules
B	Espace ou blanc
+ -/*	
/	Slash ou barre transversale
=	Egal
$	Signe monétaire
.;.("") <>	

on notera les espaces dans le texte par un b

II – CREATIONS DE MOTS COBOL

- Chiffres de 0 à 9 , lettre de A à Z,



30 caractères au maxi

Exemples de déclarations

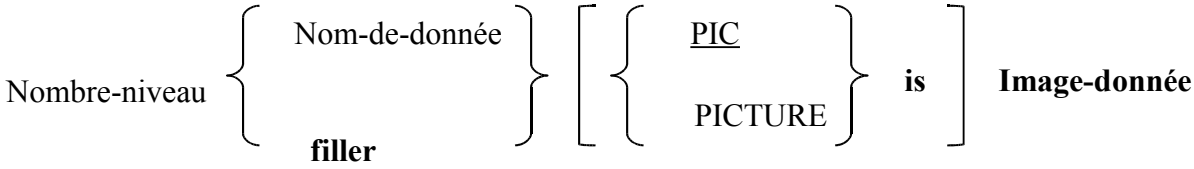
77 J PICTURE 9

77 K PIC 99.

77 INDIC-1 PIC 9(3).

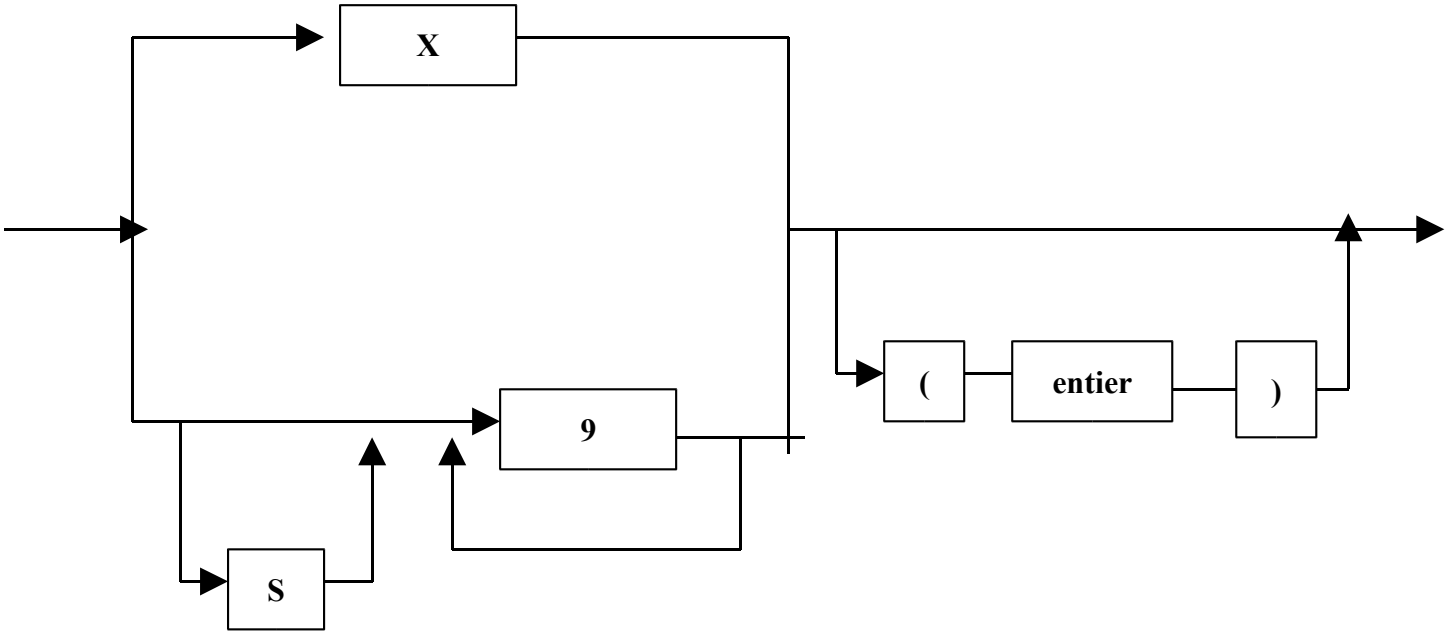
77 MODE-DE-PAIEMENT PIC X.

III – SCHEMA GENERAL D'UNE DECLARATION EN DATA DIVISION



(01 à 49 ou 77)

Ce schéma n'est pas complet, mais permet de décrire, dans un premier temps, l'essentiel des données à traiter
:
La notion **image donnée** peut-être représentée par le diagramme.:



Règles

Les pictures" alphanumériques (X) peuvent être <= 255
ou <= 120 selon les ordinateurs ; mais actuellement (c'est le cas sur l'AS 400) non limitées.

- les "pictures" numériques (9) doivent être <= 18

IV – LES LITTERAUX

C'est une CONSTANTE non identifiée par un nom symbolique

A) Littéral alphanumérique

C'est une combinaison de caractères délimitée à ses deux extrêmités par des guillemets et composée de caractères quelconques appartenant au jeu de caractères du calculateur. La longueur d'un littéral doit être comprise, en général, entre 1 et 128.

B) **Littéral numérique (de 1 à 18 chiffres)**

C'est un nombre de 18 chiffres maximum, écrit seul, avec un point décimal si nécessaire et précédé d'un + ou -.

Ex 747 + 3.14159 - .58

N.B. Certains COBOL (IBM) autorisent l'usage des nombres en notation anglo-saxonne

[+] **Mantisse** **E** [+] **exposant**

V – Les Constantes Figuratives

Ce sont des littéraux alphanumériques ou numériques désignés par des noms réservés et connus du compilateur.

ZERO : représente la valeur zéro dans le cas de données numériques.

SPACE : désigne le (ou une suite de) caractère(s) blanc(s) pour les données alphanumériques.

[HIGH-VALUE] : pour les données alphanumériques désignent un ou
[LOW-VALUE] : plusieurs caractères

ayant [la plus haute]
[la plus basse]

valeur dans la hiérarchie de codification interne

Soit [**FF**] en héxadécimal
[**00**]

ALL littéral : représente une ou plusieurs fois la chaîne de caractères composant le littéral.

VI – REGLES D'ECRITURE EN COBOL

Tout élément du programme sera précédé et suivi d'un ou plusieurs espaces, exception faite pour :

* la (qui n'est pas suivie d'espace

* les ; , . ne sont pas précédés d'espace

* la) n'est pas précédée d'espace.

CHAPITRE IV

LA FEUILLE DE PROGRAMME COBOL

****

Le format de la feuille de programmation offre une méthode normalisée d'écriture de programme COBOL.

Chaque ligne représente l'image d'un enregistrement type (carte perforée autrefois, ligne d'un terminal). Le compilateur accepte des programmes écrits dans le format de référence.

1) Colonnes 1 à 6 : contiennent les numéros de séquence, en général

1,2,3 → page
4,5,6 → ligne

2) Colonne 7 : zone d'indicateurs permettant d'utiliser :

- un commentaire si on a placé le signe *

- une suite d'instruction COBOL n'ayant pu être achevée sur la ligne précédente (Ex mot coupé en deux ou littéral), si on utilise le signe – (tiret)

3) Colonnes 8 à 72 : utilisées pour écrire les instructions du programme-source.

- col 8 à 11, marge A : en-têtes de division, noms-de-section, noms-de-paragraphe**,** certains nombre-niveau.

- col 12 à 72, marge B : les instructions courantes.

4) Colonne 73 à 80 : identification du programme (peu utile au programmeur)

CHAPITRE V
QUELQUES NOTIONS SUR LES DONNEES MANIPULEES EN COBOL

****

I – DONNEES SIMPLES NON STRUCTUREES OU ELEMENTAIRES

Ces données sont décrites en DATA DIVISION, WORKING-STORAGE SECTION, à l'aide d'un nombre niveau 77 (marge A), d'un nom-donnée choisi par le programmeur et d'une image-donnée introduite par le mot PIC.

Ex : 77 TITRE PIC X(10).
77 K PIC 999.

II – Les structures ou groupes

Une structure est un ensemble d'informations placées dans un ordre défini et strict du point de vue TYPE et LONGUEUR.

	NOM	PRENOM
Ex1 : ZONE →	10Cα N	8CαN

Ex2 : LIGNE → NOM = ┌──────────┐ PRENOM = ┌──────────┐
└──────────┘ └──────────┘

Nous décrivons ces structures en algorithmique de la façon suivante :

Soit ZONE de type structure.
dont NOM chaîne (10).
dont PRENOM chaîne (8).

Soit LIGNE de type structure,
dont LIB1 chaîne (6) initial 'NOM b = b '.
dont NOML chaîne (10),
dont LIB2 Chaîne (9) initial 'PRENOM b = b '.
dont PRENOML chaîne (8).

Pour décrire les mêmes objets COBOL on utilise les nombres-niveau allant de 01 à 49

EX 1

```
1       ZONE.
        02      NOM PIC X (10).
        02      PRENOM PIC X (8).

1       LIGNE

        02      FILLER PIC X(6) VALUE "NOM b = b".
        02      NOML PIC X(10).
        02      FILLER PIC X(9) VALUE "PRENOM b = b".
        02      PRENOML PIC X(8).
```

Le mot-clé FILLER permet de réserver une zone sans la nommer.

CHAPITRE VI

NOTIONS SUR DES ENTREES-SORTIES SIMPLES

PREMIERS EXEMPLES DE PROGRAMMES

****

I – ENTREE D'INFORMATIONS A PARTIR DU CLAVIER

L'instruction ACCEPT, dans son format le plus simple, permet de rendre disponible des informations de faible volume dans la donnée spécifiée.

ACCEPT identificateur

Identificateur est évidemment un nom de zone ayant été décrit dans la DATA DIVISION.

ATTENTION

1) Il existe d'autres formes du ACCEPT que nous verrons ultérieurement.

2) Si la longueur de ce que vous tapez est inférieure à celle de l'identificateur, vous ne pouvez prévoir ce qui va subsister dans la partie droite de cet identificateur.

Ex : 77 IDENT PIC X(10).

ACCEPT IDENT on rentre IUT b INFO puis return

Qu' y a-t-il dans IDENT | IUT INFO ?? |

Les ?? signifient que l'utilisateur ignore la valeur de ces deux caractères.

Ceci oblige très souvent le programmeur à réinitialiser une zone (à blanc par exemple) avant de prévoir un ACCEPT dessus.

Par ailleurs l'ordre ACCEPT ne peut permettre de lire une série de données comme dans d'autres langages.

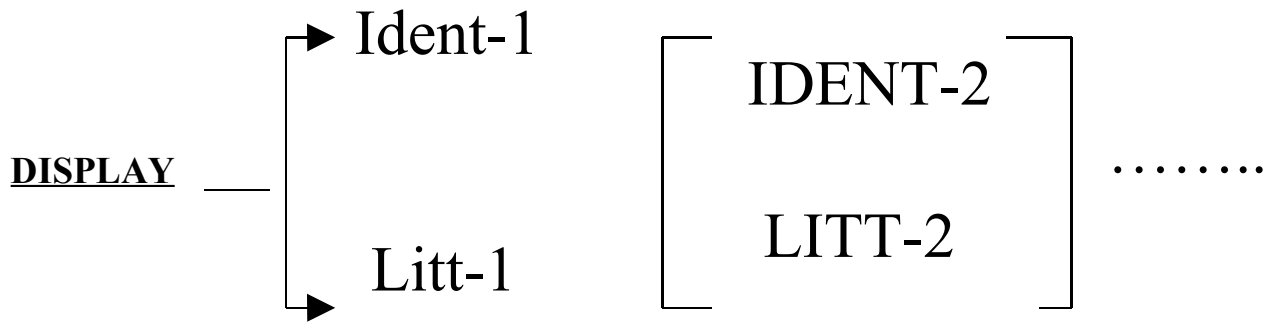
ACCEPT X Y Z **est à OUBLIER**

Si l'on veut absolument n'avoir qu'un seul ACCEPT il faut placer X,Y,Z dans une structure (voir Travaux Dirigès).

II – Sortie d'informations sur l'écran

C'est l'opération quasi "symétrique" du ACCEPT puisqu'il s'agit d'afficher le contenu de zones de mémoire centrale par l'intermédiaire des noms-données manipulés dans le programme.

Mais en plus, on a la possibilité de rajouter des chaînes de caractères ne figurant pas dans la mémoire centrale mais directement dans le programme.



Cette instruction provoque l'écriture de faible volume d'informations à l'écran.

ATTENTION :

Cette instruction n'est pas l'instruction privilégiée de sortie du langage COBOL, mais elle est très facile à utiliser pour débuter :

```
Ex      77      TEXTE PIC X(8) VALUE "IUT INFO".
|
|
(1) DISPLAY TEXTE
(2) DISPLAY "LE TEXTE EST "  TEXTE
```

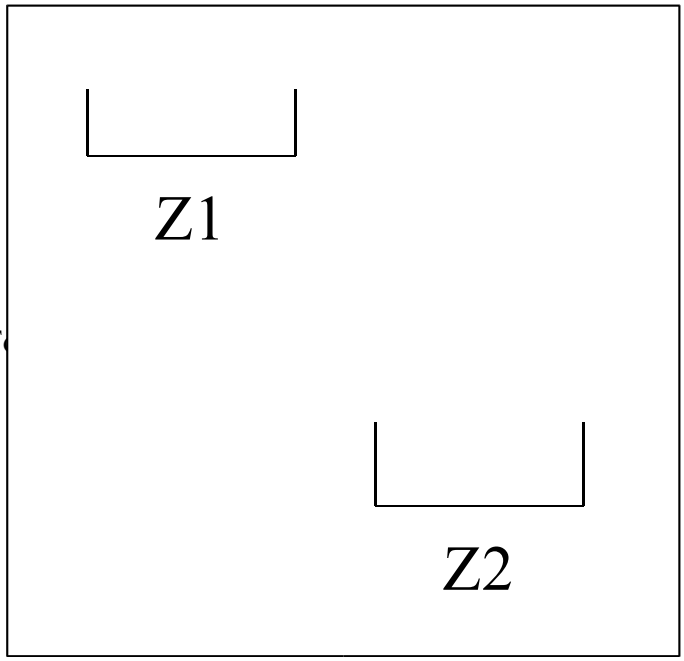
(1) provoque à l'écran l'affiche de IUT INFO

(2) provoque à l'écran l'affiche de LE TEXTE EST IUT INFO

ATTENTION :

Une impression à l'écran ne fait que RECOPIER un état de la Mémoire Centrale mais ne la modifie en rien

III – MOUVEMENT EN MEMOIRE CENTRALE



F

En algorithme Z2 = Z1
le verbe provoquant un
transfert de caractères
d 'une zone à l'autre
est le verbe MOVE

Format général

MOVE { Ident-1 / littéral } TO ident-2 [ident-3]….

1er Cas : Z1 et Z sont des données

alphanumérique
MOVE Z1 TO Z2

- longueur Z1 = longueur Z2 aucun problème
- longueur Z1 < longueur Z2 on complète avec des b à droite
- longueur Z1 > longueur Z2 on tronque A la longueur de Z2

Numérique
MOVE Z1 TO Z2

- longueur Z1 = longueur Z2, aucun problème
- longueur Z1 < longueur Z2, on complète Avec des zéros à gauche
- longueur Z1 > longueur Z2, on tronque à gauche!!

IV – EXEMPLE DE PROGRAMME TRES SIMPLE

```
Col 8   Col 12
↓    ↓

IDENTIFICATION DIVISION.
PROGRAM-ID. b EX1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  Z1  PIC X(10).
77  Z2  PIC X(10).
PROCEDURE DIVISION.
PRINCIPAL SECTION.
DEBUT.
        DISPLAY "ENTREZ UNE VALEUR POUR ZONE".
        MOVE SPACES TO Z1.
        ACCEPT Z1.
        MOVE Z1 TO Z2.
        DISPLAY Z2.
FIN.
        STOP RUN.
```

NB.- **Attention !!** Sur l'AS/400, le traitement des verbes ACCEPT ou DISPLAY est un peu particulier : chaque appel à l'un de ces deux verbes provoque une interruption du déroulement du programme et l'apparition d'un écran où s'affichent les MESSAGES engendrés par les DISPLAY. Les zônes à saisir par ACCEPT doivent l'être sur une ligne réservée à cet usage. Cet aspect confirme que les verbes ACCEPT et DISPLAY doivent intervenir le moins souvent possible en COBOL.
En fait, ACCEPT et DISPLAY utilisent la console associée à l'écran sur lequel le programme cobol tourne, cette console est normalement prévue pour les messages systèmes ou la mise au point des programmes.

V – DEUXIEME PROGRAMME UTILISANT DES DONNEES EN GROUPE

Cet exercice reprend les données ZONE et LIGNE du chapitre. On veut savoir (en une seule lecture) le nom et le prénom d'un individu afin de les imprimer à l'écran sous forme explicite (donc en utilisant la structure de LIGNE).

```
Col18  Col 12
↓      ↓

IDENTIFICATION DIVISION.
PROGRAM-ID. b EX2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ZONE.
        02 NOM PIC X(10).
        02 PRENOM PIC X(8).
01 LIGNE.
        02 FILLER PIX X(6) VALUE "NOMb=b".
        02 NOML PIC X(10).
        02 FILLER PIC X(9) VALUE "PRENOMb=b".
        02 PRENOML PIC X(8).
PROCEDURE DIVISION.
PRINCIPALE SECTION.
DEBUT.
        DISPLAY "ENTREZ LES DONNEES SELON LE FORMAT:".
        DISPLAY "NOM(10),PRENOM(8)".
        ACCEPT ZONE.
        MOVE NOM TO NOML.
        MOVE PRENOM TO PRENOML.
       DISPLAY LIGNE.
FIN.
        STOP RUN.
```

N.B.- 1) Que manque-t-il par rapport à EXI ,

2) Pouvez-vous améliorer la saisie du nom et du prénom ?

3) Que faut-il pour écrire le nom et le prénom sur deux lignes séparées ?

CHAPITRE VII

NOTIONS SUR LES OPERATIONS ARITHMETIQUES SIMPLES

****

Nous passerons en revue dans ce chapitre les quatre verbes :

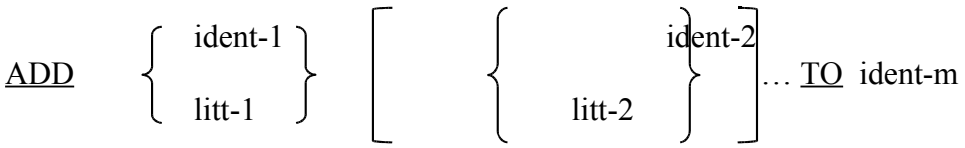
ADD
SUBTRACT
MULTIPLY
DIVIDE

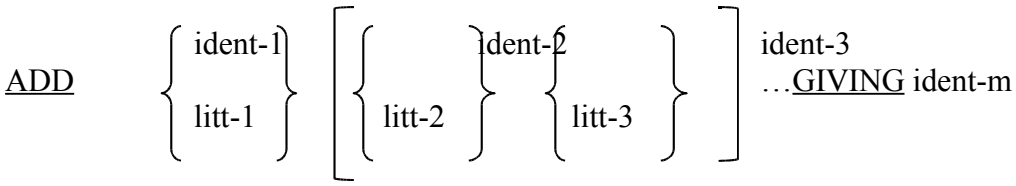
dans leur format le plus utile. Pour chaque verbe, une succession d'exemples est fournie pour vous inviter à toujours trouver la bonne formulation dans vos programmes.

I – ADDITION

Cet ordre ne s'adresse évidemment qu'à des noms de données strictement numériques.

FORMAT GENERAL





EXEMPLES

ALGORITHMIQUE	COBOL
I ← I + 1	ADD 1 TO I
J ← J + I + K	ADD I K TO J
J ← I + 1	ADD 1 I GIVING J
I ← K + L + M	ADD K L M GIVING I

II – SOUSTRACTION

FORMAT GENERAL

(1) SUBTRACT { ident-1 / litt-1 } [ident-2 / litt-2] ….. FROM ident-m

(2) SUBTRACT { ident-1 / litt-1 } [ident-2 / litt-2] … FROM { litt-m } ident-m

GIVING ident-n

EXEMPLES

ALGORITHMIQUE	COBOL
I ← I – 1	SUBTRACT 1 FROM I
J ← J – I – K	SUBTRACT I K FROM J
J ← I – 1	SUBTRACT 1 FROM I GIVING J
I ← J – K – L – M	SUBTRACT K L M FROM J GIVING I

III – MULTIPLICATION

FORMAT GENERAL

(1) MULTIPLY { Ident-1 / Litt-1 } BY ident-2

(2) MULTIPLY { ident-1 / litt-1 } BY { ident-2 / litt-2 } GIVING ident-3

ALGORITHMIQUE	COBOL
I ← I * 2	MULTIPLY 2 BY I
K ← I * J	MULTIPLY I BY J GIVING K

IV – DIVISION

FORMAT GENERAL

(1) DIVIDE { ident-1 / litt-1 } INTO ident-2

(2) DIVIDE { ident-1 / litt-1 } { BY / INTO } { ident-2 / litt-2 } GIVING ident-3
[REMAINDER ident-4]

ALGORITHMIQUE		COBOL
B ← B/A		DIVIDE A INTO B
	ou	DIVIDE B BY A GIVING B
C ← B/A		DIVIDE A INTO B GIVING C
	ou	DIVIDE B BY A GIVING C

Si on veut le reste on rajoute REMAINDER D

Il faut expliquer ici la spécificité du COBOL, comme langage de gestion : tous les calculs s'effectuent en entier, la précision décimale étant décrite par les clauses PICTURE.
C'est véritablement un atout du cobol, et la justification de son utilisation actuelle : les calculs sont justes, sans arrondi !

CHAPITRE VIII –

LES VERBES DE TRAITEMENT

(PROCEDURE DIVISION) INITIATION

****

FORMAT GENERAL SIMPLIFIE

PROCEDURE DIVISION [nom-de-paragraphe, [phrase]…}…

La "Procédure Division" doit exister dans tout programme COBOL.
Elle contient essentiellement les traitements.

Les ordres essentiels sont :

OPEN, CLOSE, STOP RUN, ADD, SUBTRACT, MULTIPLY, DIVIDE, PERFORM, READ, WRITE, REWRITE, GO TO, IF

I – INSTRUCTION MOVE

Cette instruction est fondamentale puisqu'elle permet le transfert de données vers une ou plusieurs zone de données.

FORMAT GENERAL :

MOVE { ident-1 / littéral } TO ident-2 [ident-3]…

Ex

MOVE A (B) TO X. (A(B) désigne une case de vecteur)

MOVE X TO C.

MOVE X TO D(E).

Cette instruction est simple à comprendre, mais parfois délicate à utiliser (rôle des PICTURE). Nous y reviendrons ultérieurement.

II – EXPRESSIONS CONDITIONNELLES (voir COURS ALGO)

Elles identifient les conditions à évaluer pour permettre le choix d'une alternative. IF, PERFORM sont les deux instructions essentielles utilisant de telles expressions, COBOL est riche en conditions :

- conditions simples (comparaison, classe, nom-condition, signe)
- conditions composées (OR, AND, NOT)

A – CONDITION DE COMPARAISON

Elle nous permet de fournir un premier format pour le IF :

```
IF { exp-arith-1 }        IS [NOT] { GREATER THAN }        { exp-arith-2 }
   { ident-1     }                 { LESS    THAN }        { ident-2     }
   { litt-1      }                 { EQUAL   TO   }        { litt-2      }
                                   { >            }
                                   { <            }
                                   { =            }
```

Il y a naturellement équivalence de sens entre :

GREATER ET	>
LESS ET	<
EQUAL ET	=

On note l'absence de signe pour > = et <= il faut traduire par NOT < et NOT > (**sur l'AS/400**, **les signes > = et < = sont admis**).

B – CONDITION DE CLASSE

FORMAT GENERAL :

```
IF  ident  IS [NOT]  { NUMERIC      }
                     { ALPHABETIC   }
```

Cette condition détermine si l'opérande est numérique (composé uniquement de chiffres avec ou sans signe) ou alphabétique (A'…Z', espace).

Le test ALPHABETIC ne peut en principe pas être utilisé avec une donnée numérique (PIC 9( )). Ces deux dernières phrases correspondent à la norme ANS 74 mais peuvent être élargies comme c'est le cas en Cobol-Microsoft.

Les autres conditions simples seront vues ultérieurement.

C – CONDITIONS COMPOSEES

Celles qui font intervenir les opérateurs logiques AND, OR, NOT.

D – REGLES D'EVALUATION DES CONDITIONS

(1) Les expressions arithmétiques sont calculées

(2) Les conditions simples sont établies dans l'ordre : comparaison, classe, nom-de-condition, signe.

(3) Les conditions simples négatives sont établies.

(4) Les conditions composées dans l'ordre AND, OR;

(5) Les conditions composées négatives sont évaluées.

N.B - L'utilisation des parenthèses est conseillée pour augmenter la lisibilité du programme et éviter les erreurs d'interprétation.

III – EXEMPLES DE CONDITIONS COMPOSEES

1) IF A = B OR C = D action-2

2) IF (A = B OR C = D) AND (E > F) action-2

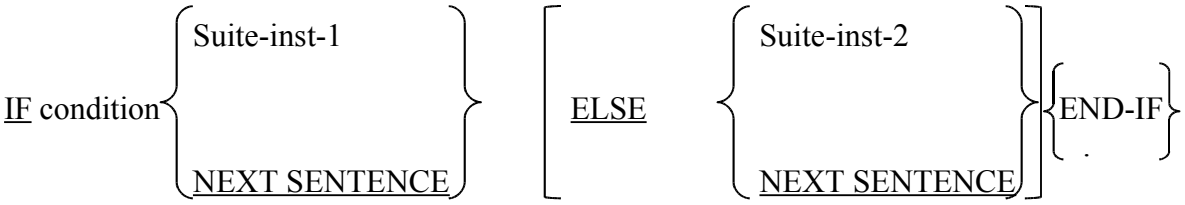
3) IF (A = B) OR (C = D AND E> F) action 3.

IV – L'INSTRUCTION IF

C'est l'instruction qui traduit les schémas algorithmiques bien connu :

```
Si    exp. Cond.   Alors          Si    exp. cond.   Alors
|     action-1                    |
|                          ou     |     action-3
Sinon                            |
|     Action-2                   Fsi
Fsi
```

FORMAT GENERAL :



Le point terminal est l'élément le plus important de cette instruction. En effet suite-inst-1,2 représentent une série d'instructions COBOL **SANS POINT .**

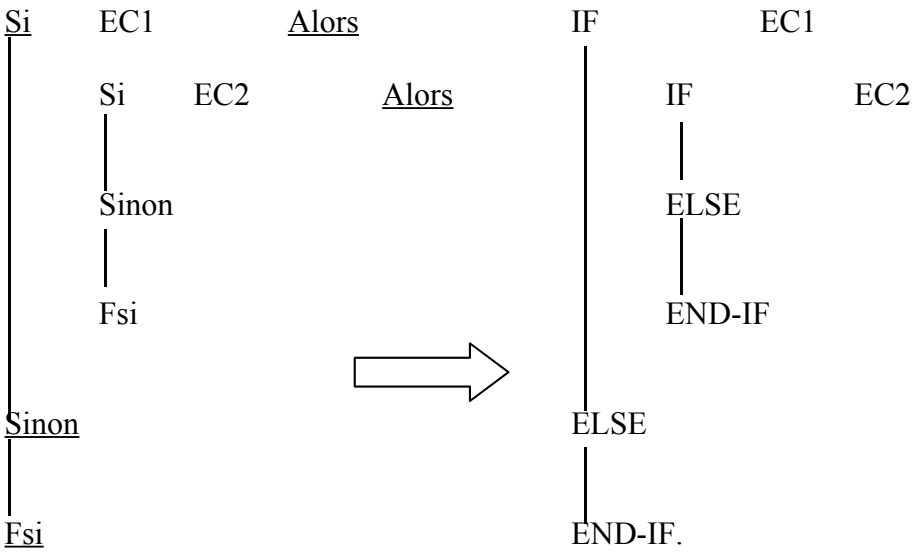
Le END-IF n'est pas obligatoire en COBOL, parce qu'il a été introduit après coup dans le langage.
NOUS VOUS CONSEILLONS D'UTILISER SYSTEMATIQUEMENT LE END-IF.
Sans lui, s'est le point qui met fin au IF. Mais il met fin A TOUS LES IF EN COURS !

Cette règle est malheureusement très dangereuse et on l'apprend vite à ses dépens !

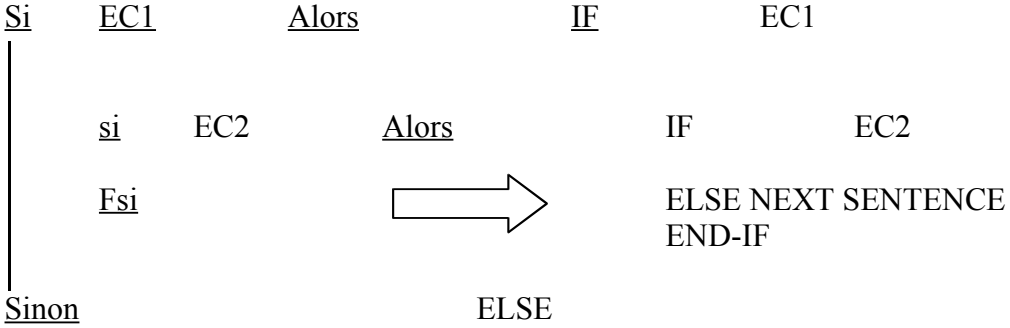
En particulier, il est naturellement possible d'imbriquer des IF mais avec quelques précautions. La règle moderne, et vivement conseillée, est de TOUJOURS utiliser le terminateur END-IF !

EXEMPLES DE STRUCTURE DE IF IMBRIQUES

1



2



Fsi END-IF.

D'ou le rôle indispensable dans certains cas de la locution NEXT SENTENCE !
Par contre on ne peut traduire :

3 Si EC Alors

Si EC2 Alors

Non traduisible pour
L'instant (nécessité de
L'appel d'action)

Fsi

Sinon → PERFORM

SAUF SI ON UTILISE
LE END-IF

Fsi

IF EC1

MOVE A TO B

IF EC2

MOVE 1 TO N

END-IF
MOVE B TO C
ELSE
MOVE B TO C
END-IF.

V – L'INSTRUCTION EVALUATE

L'instruction EVALUATE (récente est celle que vous devez utiliser quand l'utilisation du « IF » devient délicate (essentiellement au delà de deux imbrications)

FORMAT GENERAL :

EVALUATE { Id-1 | Litt-1 | expr-1 | TRUE | FALSE }

WHEN phrase-1 inst-imp-1…….

[WHEN OTHER inst-imp-2]
END-EVALUATE

L'instruction évalue l'égalité de l'entité qui suit EVALUATE avec l'une des phrases suivant WHEN.
Si aucun WHEN n'a été sélectionné, et si WHEN OTHER est écrit, on effectue inst-imp-2, sinon on passe directement à l'instruction qui suit END-EVALUATE.

Exemple 1

```
EVALUATE   A
        WHEN 1          PERFORM  P1
        WHEN 2          PERFORM  P2
        WHEN OTHER      PERFORM  P3
END-EVALUATE.
```

Si A = 1 on exécute P1 et on continue après le END-EVALUATE, sinon, si A = 2 on exécute P2. Dans tous les autres cas on exécute P3.

Exemple 2

```
EVALUATE        A + B = 10
        WHEN TRUE               PERFORM  P1
        WHEN FALSE      PERFORM  P2
END-EVALUATE.
```

CHAPITRE IX

NOTION DE PARAGRAPHES

STRUCTURE DE REPETITION (TQ) INSTRUCTION GO TO INSTRUCTION PERFORM SIMPLE

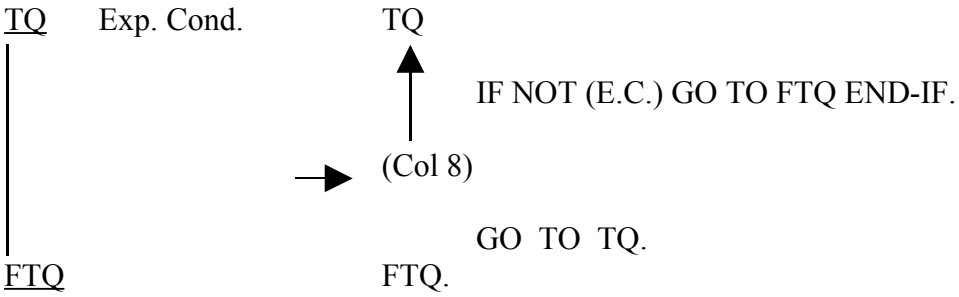
****

I – NOM DE PARAGRAPHE

Un nom-de-paragraphe est un mot choisi par l'utilisateur qui désigne un paragraphe de la « PROCEDURE DIVISION ». Ces noms sont formés selon la même règle que les identificateurs de données. Ce sont les seuls noms commençant en marge A (colonne 8). Ils sont immédiatement suivis d'un point.

II – INSTRUCTION TANT QUE : 1ère TRADUCTION

Il s'agit de traduire le schéma itératif

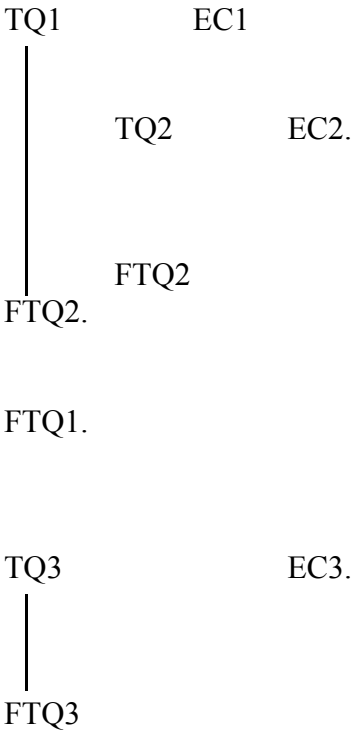


On introduit de même coup l'instruction

GO TO nom-de-paragraphe.

Dont le fonctionnement rend évident le rôle des noms-paragraphe. Comme pour les instructions IF imbriquées, il faut fournir une traduction de plusieurs TQ imbriqués :

Algorithmique



Une traduction COBOL

col 8 col12

```
TQ1.
        IF NOT (EC1) GO TO FTQ1
        END-IF.
TQ2.
                IF NOT (EC2) GO TO
        END-IF.
        GO TO TQ2
FTQ2.
        GO TO TQ1.
FTQ1.


TQ3.
        IF NOT (EC3) GO TO FTQ3
        END-IF.
       GO TO TQ3
FTQ3.
```

III NOTION D'ACTION SANS PARAMETRES – VERBE PERFORM

La notion de paramètre n'existe pas en COBOL. Seules les actions sans paramètre et ne retournant aucun résultat sont possibles. Les actions manipulent donc des variables globales.

A chaque action algorithmique, nous allons faire correspondre une SECTION de PROCEDURE DIVISION.
La PROCEDURE DIVISION commençant obligatoirement par la PRINCIPALE SECTION : action principale (correspondant au main java ou C).
L'appel d'action s'écrit alors :

PERFORM nom-de-section

Chacune de ces sections sera structurée comme suit :

```
PROCEDURE DIVISION.
PRINCIPALE SECTION.
DEBUT.

        Instructions cobol
        PERFORM TRAVAIL1.
        Instructions cobol.
FIN.
        STOP RUN.
```

Suivi des différentes sections correspondant aux sous-actions du programme :

```
TRAVAIL1 SECTION.
DEBUT.

        Instructions cobol.
        PERFORM TRAVAIL2.
        Instructions cobol.
FIN.
        EXIT.
TRAVAIL2 SECTION.
DEBUT.

        Instructions cobol.
FIN.
        EXIT.
```

D'autres formes du PERFORM sont pour leur part, intéressantes :

PERFORM nom-de-section UNTIL condition.

La forme ci-dessus correspond exactement à la structure tant-que algorithmique à la négation du test près : la condition citée est celle d'arrêt, alors que celle du tant-que indique de continuer. Nous détaillons cette forme du PERFORM dans le paragraphe suivant.

PERFORM nom-de-section VARYING nom-de-variable FROM valeur1 BY valeur2 UNTIL condition.

La forme ci-dessus correspond à la structure for de Java ou du C. La section sera effectuée tant que la condition est fausse, la variable initialisée à valeur1 étant incrémentée à chaque exécution de la section d'une valeur valeur2.

Plus simplement : la forme suivante répète n fois une section donnée :
PERFORM nom-de-section valeur TIMES

Dans les exemples précédents, les valeurs sont des entiers ou des variables numériques.

L'utilisation de sections pour l'instruction PERFORM permet une meilleure structuration des programmes. De plus, elle permet d'utiliser des noms de paragraphes locaux à la section (TQ, FTQ etc…).
NOUS VOUS CONSEILLONS L'UTILISATION SYSTEMATIQUE DES SECTIONS.

Malheureusement, ce conseil n'est parfois pas suivi. En effet, le COBOL possède des présentations différentes de l'instructions PERFORM :

PERFORM nom-de-paragraphe. *N'exécute que le paragraphe mentionné.*

PERFORM paragraphe1 THRU paragraphe2. *Exécute le contenu des paragraphes 1 à 2. Dans ce cas le paragraphe2 doit être déduit à l'instruction EXIT.*

IV –TRADUCTION DU « TQ ».

Le « PERFORM non-section UNTIL cond-1 END-PERFORM » permet de donner un équivalent du TQ de l'algorithmique.

Exemple

```
I ← 1                 MOVE 1 TO 1
TQ  I <10             PERFORM BOUCLE UNTIL I > 10
    s ← s + v →       END-PERFORM
    i ←  i + 1        suite

Ftq
        Suite         BOUCLE SECTION
                      DEBUT.
                            ADD V TO S.
                            ADD 1 TO I.
                      FIN.
                            EXIT.
```

On peut éviter l'appel à une section comportant aussi peu d'instructions en supprimant le nom BOUCLE ; Voici l'écriture la plus proche de ce que vous écrivez en cours d'algorithmique :

```
MOVE 1        TO  I.
PERFORM     UNTIL I > 10
        ADD S  TO  V
        ADD 1  TO  I
END-PERFORM.
```

Exemple de traitement d'une file de caractères terminée par un caractère fixé à l'avance (*)

Algorithmique	Cobol
	77 ZONE PIC X.
	en "PROCEDURE DIVISION"
Lire zone	ACCEPT ZONE
	TQ.
TQ zone # "*"	IF ZONE = "*" GO TO FTQ.
traiter-zone (action)	PERFORM TRAITER-ZONE
	ACCEPT ZONE
lire zone	GO TO TQ.
	FTQ.
FTQ.	EXIT.

Ou

```
ACCEPT ZONE
PERFORM UNTIL ZONE = " * "
        PERFORM TRAITER-ZONE
        END-PERFORM
END-PERFORM.

TRAITER-ZONE.
        ACCEPT-ZONE.
```

CHAPITRE X

COMPLEMENTS SUR LES STRUCTURES (NIVEAUX) CONSEQUENCES POUR LE VERBE « MOVE »

****

RAPPEL : Une structure (« article ») est le regroupement d'informations de types divers. COBOL permet la description en faisant ressortir l'état de la hiérarchie qui relie ces données élémentaires.

Une donnée est donc écrite par :

1) Un numéro-de-niveau

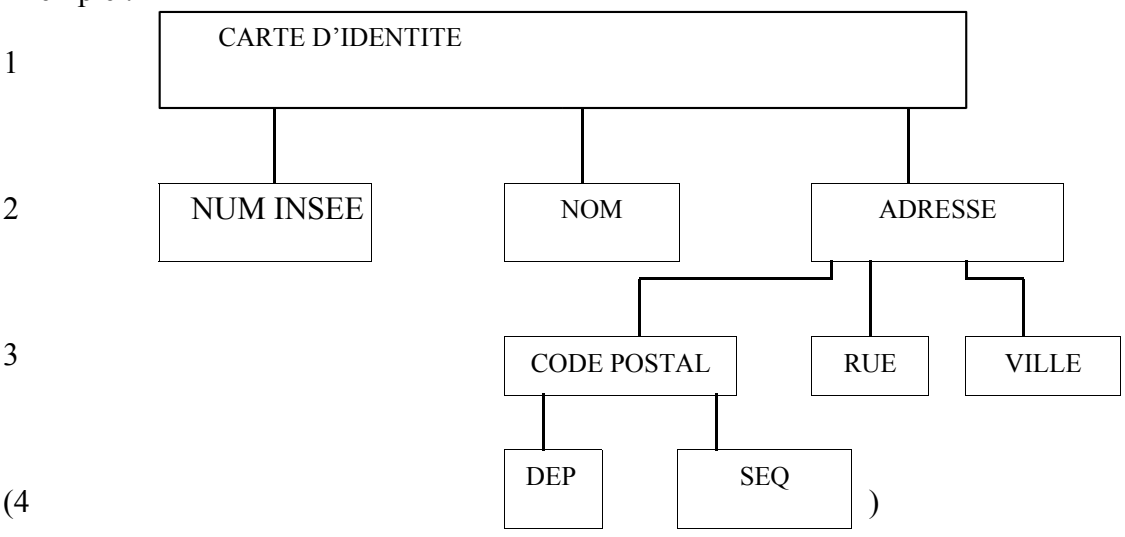
2) Une référence symbolique (nom-donnée)

3) Des attributs permettant de préciser la forme de cette donnée.

I – LES NUMEROS DE NIVEAUX (ou nombres-niveau)

Un système de numéros-de-niveau précise l'organisation des données élémentaires. Le niveau le plus général de la structure est 01. Les données internes reçoivent un numéro supérieur (pas nécessairement consécutif) allant de 02 à 49.

Exemple :



Se traduira en COBOL

```
01  CART-IDENT.                          01  CART-IDENT.
  02  NUM-INSEE PIC 9(13).                02  FILLER PIX X(33).
  02  NOM     PIC X(20).                  02  ADRESSE.
  02  ADRESSE                  ou           03 CODPOS.
      03  CODPOS PIC 9(5).                      04  DEP PIC 99
.
                                             04 SEQ PIC 999
.
      03  RUE        PIC(22).             03 RUE PIC X(22).
      03  VILLE PIC X (20)                03 VILLE PIC X(20).
```

A partir de l'une de ces descriptions, tout mouvement (MOVE) partiel ou global est possible à condition que des zones correspondantes existent.

Ex. :

```
01 LIGNE1.
     02 ADRESSE1 PIC X(47).
```

permet le transfert MOVE ADRESSE TO ADRESSE1.

CHAPITRE XI

LA CLAUSE PICTURE
LA CLAUSE BLANK WHEN ZERO

****

I – FONCTION DE PICTURE

Cette clause définit la catégorie d'une donnée élémentaire et ses caractéristiques d'édition.

FORMAT GENERAL :

{ PICTURE / PIC } IS Chaîne-de-caractères

CETTE CLAUSE NE PEUT ETRE UTILISEE QU'AU NIVEAU ELEMENTAIRE.

II – PICTURE DE TRAITEMENT

A – DONNEE ALPHABETIQUE

A(n) n ← 120 en général domaine restreint aux lettres de l'alphabet et de l'espace (rarement utilisée).

B – DONNEE NUMERIQUE

9,S,V

Le contenu de la donnée est une combinaison de dix chiffres de 0 à 9, plus éventuellement un signe algébrique.

Cette picture permet d'effectuer toutes les opérations arithmétiques sur de telles zones.

Ex :

9(n) O < n <= 18 représente un entier positif

S9(n) O < n <= 18 représente un entier signé

9(n)V9(m) O < n + m <= 18 représente un nombre décimal non signé

S9(n)V9(m) O < n + m <= 18 représente un nombre décimal signé.

Exemples :

77 I PIC 9(5).

77 J PIC S9(8).

77 K PIC 9(8)V99.

77L PIC S9(7)V9(4).

C – DONNEE ALPHANUMERIQUE X(n) n <= 120 en général

Le contenu d'une telle donnée est formé de caractères quelconques du jeu de caractères du calculateur.

Il est hors de question de faire subir des opérations arithmétique à de telles données !

III – PICTURE D'EDITION

La chaîne-de-caractères d'une telle donnée est limitée à certaines combinaisons des 13 symboles suivants en plus du symbole monétaire.

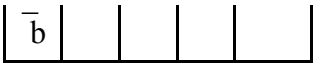
B / V Z O 9, . + - * CR DB

Les symboles S, V, . , CR, DB ne peuvent figurer qu'une seule fois.

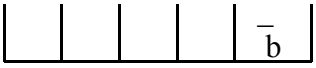
A – IMPRESSION D'UN SIGNE

Ex

-9(4) à l'impression

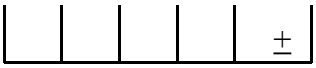


9(4)-

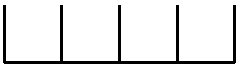


+9(4)

9(4)+



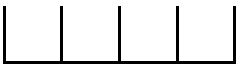
9(4)CR si valeur > O



bb

9(4)DB

si valeur < O



CR
DB

B – LES SYMBOLES ',' ET '.'

Le symbole ',' représente la position où une virgule sera insérée. Cette position est comptée dans la longueur de la donnée.

Le symbole '.' figure la marque décimale pour le cadrage, il ne doit pas être le dernier de la chaîne-de-caractères.

Ex :

9(4) , 9 9 permet d'éditer une donnée en séparant en 2 parties sans rien présupposer de la donnée interne

9(4).99 permet d'éditer sou la forme 

et nécessite une picture de la zone émettrice avec un symbole V car le cadrage se fera du 'V' sur '.'.

C – LES SYMBOLES 'Z' et '*'

Ce sont les symboles de tête de chaîne-de-caractères permettant respectivement :

- Pour 'Z' de supprimer tous les zéros de gauche non significatifs.
- Pour '*' de marquer une protection (usage bancaire) par apparition du caractère '*' en lieu et place des zéros de gauche non significatifs.

Ex :

77 I PIC Z(4) Valeur 0123 à l'édition b123
Valeur 0000 « « « bbbb

77 J PIC *(3)99 Valeur à l'édition

00001 ***01
00000 ***00
00134 **134
12345 12345

D – LE SYMBOLE MONETAIRE '$'

Ce symbole (ainsi que le '+' ou '-') peut-être utilisé de manière fixe ou flottante :

$999	valeur	édition
	1	$001
$(4)	valeur	édition
	1	bb$1
$(4)	valeur	édition
		bbbb

Ceci permet de faire « coller » le signe à la valeur significative de la donnée.

E – LES SYMBOLES B, O et /

Ces trois caractères représentant une position d'un caractère dans lequel on insérera :

- un blanc ou espace (B)
- un zéro (0)
- un slash (/)

Ex :

77 DATES PIC 99/99/99

fournira à l'édition

12/30/45

L'usage du B peut-être intéressant dans le cas suivant :

```
01  LIGNE1.                                  01  LIGNE1.
    02  Z1  PIC  X(9)B(10).      Remplace        02  Z1 PIC X(9).
    02  Z2  PIC  X(10)B(43).                        02  FILLER PIC  X(10)
                                                         VALUE SPACES.
                                                 02 Z2 PIC X(10).
                                                 02 FILLER PIC X(43).
                                                         VALUE SPACES.
```

IV – LA CLAUSE BLANK WHEN ZERO.

Cette clause, ajoutée après la PICTURE, permet d'éliminer tous les caractères parasites insérés lorsque la zone à éditer est nulle.

Format général : BLANK WHEN ZERO.

Ex :

```
77      SOMME  PIC  9(5)V99.

77      SOM-EDIT  PIC $$$B$$$9.99.
```

Si SOMME contient une valeur nulle et que l'on fasse le transfert
MOVE SOMME TO SOM-EDIT, on obtiendra : $0.00

Par contre, si l'on désire ne rien avoir on décrit SOM-EDIT de la manière suivante :

```
77      SOM-EDIT PIC $$$B$$9.99                BLANK WHEN ZERO.
```

Lors de l'édition, on n'obtient que des espaces. Le seul symbole d'incompatibilité est '*'.

CHAPITRE XII

LES CLAUSES OCCURS, VALUE, REDEFINES (DATA DIVISION)

****

LE VERBE INITIALIZE (Procédure Division)

I – LA CLAUSE VALUE

Elle permet de définir la valeur initiale d'une donnée :

Format général VALUE IS { constante figurative / littéral }

Ex :

77 EXP PIC S9V9(5) VALUE 2.71828.

77 TITRE PIX X(12) VALUE 'FIN DU COURS'.

01 STRUC.

1 CPT PIC 9 VALUE ZERO.

02 LIBEL PIC X(30) VALUE SPACES.

La clause VALUE ne doit pas être utilisée dans une rubrique de description de données contenant une clause OCCURS ou REDEFINES.

II – LA CLAUSE OCCURS

Elle permet d'éviter de décrire séparément des données qui se répètent ; elle définit le nombre de ces répétitions et donne les informations nécessaires à l'emploi d'indices.

On décrit donc des groupes de données composés d'éléments simples ayant tous des attributs identiques, puis, par généralisation, on peut aussi créer des « tableaux de structure ».

A – DESCRIPTION D'UN VECTEUR

TABLE-EX

EL(1) EL(3) EL(9)

Un tel groupe de données sera décrit par :

01 TABLE-EX.

2 EL PIC 9(2) OCCURS 9 TIMES.

Dans la procédure division, un élément peut-être recherché à partir d'un indice entier.

Ex :

ADD 1 TO EL (4).

Ou

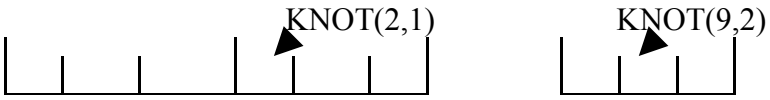
MOVE 7 TO J.

ADD 1 TO EL (J)

J étant un entier positif décrit en WORKING-STORAGE SECTION ou en FILE SECTION.

B – TABLE A 2 OU 3 INDICES

TABLE-EX-2



EL(1) EL(2) EL(9)

C'est, par exemple, le cas où chaque élément du vecteur précédent est à son tour décomposé en éléments consécutifs de même type et de même longueur.

Cette structure sera décrite par :

```
01 TABLE-EX-2.

    02 EL OCCURS 9.

    03 KNOT PIC 99 OCCURS 3.
```

On en déduit immédiatement l'extension à 3 indices.

Dans la procédure division, on utilisera un élément KNOT de la façon suivante

```
SUBTRACT 3 FROM KNOT(9.2)
```

mais aussi un élément EL :

```
MOVE '111111' TO EL (2).
```

RAPPEL : pas de VALUE avec (ou sous) une clause OCCURS.

C – TABLE DE STRUCTURES

TABLE-EX-3

ETUD(1) ETUD(2) ETUD(9)

sera décrite par :

```
01 TABLE-EX-3.

    02 ETUD OCCURS 9.

        03 NOM PIC X(7)

        03 AGE PIC 99.
```

Naturellement, toute donnée figurant dans la description d'une zone en OCCURS doit être convenablement indicée.

Ex :

```
    MOVE 'MARCEL' TO NOM(2)
    MOVE 27 TO AGE(1)
```

mais aussi

```
    MOVE'MARCELb38'TO ETUD(9).
```

III – LA CLAUSE REDEFINES

Elle permet de décrire une zone de mémoire de différentes manières.

FORMAT GENERAL

numéro-de-niveau n-d-1 REDEFINES n-d-2.

REGLES DE SYNTAXE :

1) Cette clause suit immédiatement n-d-1

2) Les deux numéros-de-niveau sont identiques

3) La clause ne peut s'employer au niveau 01 de la « FILE SECTION »

4) Il ne doit pas y avoir d'OCCURS dans la rubrique n-d-2

Exemple :

```
01 ZONE-PAYE-1.
        02 NO-CLIENT PIC X(6)

        02 NOM PIC X(8).

        02 MONTANT PIC 9(6)V99.

        02 DATE.
                03 MOIS PIC XX.

                03 AN PIC XX.

01 ZONE-PAYE-2 REDEFINES ZONE-PAYE-1.

        02 NUM-C PIC 9(6).

        02 VILLE PIC X(15).

        02 CO-POST PIC 9(5).

01 ZONE-PAYE-3 REDEFINES ZONE-PAYE-1.

        02 FILLER PIC X(6).

        02 TOTAL PIC 9(8)V99.

        02 ANNEE PIC XX.

        02 FILLER PIC X(8).
```

CONSEIL : Il faut évidemment se donner les moyens de savoir dans quel type de structure on se trouve (par un test sur une partie de la zone redéfinie par exemple).

IV – EXEMPLES D'UTILISATION SIMULTANEE DES 3 CLAUSES VALUE, OCCURS, REDEFINES.

INITIALISATION D'UNE TABLE sans faire appel à l'affectation en PROCEDURE DIVISION.

Il est impossible de faire suivre la clause OCCURS de la clause VALUE, d'où la manière d'initialiser une table au moyen de REDEFINES (attention : usage réservé aux tables de petites dimensions).

```
01 TABSEM.

    02 SEMAINE PIC X(56) VALUE 'LUNDIbbbMARDI'bbbMERCREDI

    JEUDI… DIMANCHE'.

    02 SEM-BIS REDEFINES SEMAINE.

    03 JOUR PIC X (8) OCCURS 7.
```

V – LE VERBE INITIALIZE

Ce verbe récent permet d'initialiser les données spécifiées selon leur NATURE, par défaut :

- alphabétiques et alphanumériques : SPACE
- numériques : ZERO

Les zones FILLER sont ignorées.

FORMAT GENERAL

INITIALIZE iden-1 [ident-2…]

REPLACING { ALPHANUMERIC / NUMERIC / NUMERIC-EDITEM } BY

{ id-m / lett-1 }

Ce verbe se révèle particulièrement utile pour les grilles d'écran et les structures de Working-storage section.

CHAPITRE XIII

LES FICHIERS * GENERALITES

I – QUELQUES DEFINITIONS

A – ENREGISTREMENT LOGIQUE (ARTICLE)

C'est un ensemble d'informations élémentaires organisé et choisi par le programmeur.

B – FICHIER

C'est un ensemble d'enregistrements logiques stocké sur un support informatique.

C – INDICATIF (CLEF)

Très souvent, chaque article possède une zone privilégiée qui sert d'indicatif repère. Tous les indicatifs sont en principe différents dans le fichier.

D – SUPPORTS

* Clavier, écran, cartes perforées, imprimante, cassettes.

* Bandes, disques, disquettes, cartouches magnétiques.

E – DEROULEMENT D'UNE ENTREE/SORTIE

Toute opération de lecture (entrée) ou d'écriture (sortie) entre un organe périphérique et l'unité centrale est une opération TRES LONGUE, vis à vis des temps de traitements en U.C. (rapport facilement supérieur à 1000).

II – NOTION D'ENREGISTREMENT PHYSIQUE (BLOC)

C'est l'ensemble d'informations transmises d'un seul coup en une opération entrée/sortie.

Ex :

* 1 ligne d'imprimante

Dans le cas des supports magnétiques (bandes, disques) on a intérêt à regrouper des articles trop courts pour former un bloc lu en une seule fois.

Dans les langages évolués (COBOL, PL/1, FORTRAN, PASCAL, BASIC , JAVA), les ordres de lecture/écriture correspondent toujours à des enregistrements logiques.

DEFINITION : Le nombre d'enregistrements logiques contenus dans un enregistrement physique est appelé le FACTEUR DE GROUPAGE (ou de BLOCAGE).

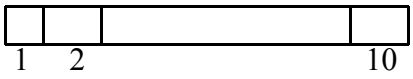
Cette notion présente peu d'intérêt au niveau des micro-ordinateurs. Elle peut-être importante à fixer sur moyens ou gros systèmes.

EXEMPLES :

Soit un fichier muni d'un facteur de blocage = 10

```
Lire (fichier)

TQ  non dernier (fichier)
|     TRAITER
|
|     Lire (fichier)
|
FTQ
```

1ère lecture (en Mémoire Centrale)



10 articles sont lus en une seule fois

Seule, la 11ème lecture déclenchera une nouvelle lecture physique.

Avantage du système précédent :

On divise pour ce programme les temps d'E/S par 10.

Inconvénient :

Pour les E/S sur ce fichier, il faut un buffer 10 fois plus grand.

III – DIFFERENTS TYPES DE FICHIERS

1) Fichiers à enregistrement de longueur fixe : ceux dont tous les enregistrements logiques ont le même nombre de caractères.

2) Fichiers à enregistrement de longueur variable : on gère une zone permettant d'écrire ou de lire des enregistrements de longueur variable (<= longueur maximale connue). (Dans ce second cas, le programmeur peut avoir à gérer cette longueur dans une zone de WORKING-STORAGE SECTION, en COBOL).

IV – UTILISATION D'UN FICHIER PAR PROGRAMME COBOL

Un fichier est caractérisé par certains ATTRIBUTS qui sont associés à ce dernier à sa création et ne seront pas remis en cause au cours de la vie de ce fichier.

L'attribut majeur est son ORGANISATION :

* séquentielle
* relative
* séquentielle indexée.

Les autres attributs peuvent être : une clé, le maximum de longueur de l'enregistrement logique, des conventions de labe, le type de l'enregistrement (fixe ou variable), le facteur de groupage.

A – ORGANISATION SEQUENTIELLE

Un tel fichier est organisé de telle façon que chaque enregistrement a un unique successeur (sauf le dernier !). La relation de succession est établie lors de la création du fichier (ordre WRITE). En dehors de la consultation et de la modification d'articles, la seule chose que l'on puisse faire sur un tel fichier est de rajouter des enregistrements après le dernier. Un fichier séquentiel sur disque (ou disquette) peut-être mis à jour « sur place », ce qui n'est pas le cas du même fichier sur bande (ou cassette).

B – ORGANISATION RELATIVE (DISQUE, DISQUETTE)

C'est un fichier pour lequel chaque enregistrement peut être rangé, ou retrouvé, en fournissant la valeur de son numéro relatif d'enregistrement.

Le dixième enregistrement est repéré par le numéro 10 qui sera chargé dans une zone adéquate de WORKING-STORAGE SECTION, P par exemple, qui sera explicitement la clé d'accès à ce fichier (voir plus loin l'utilisation pratique).

C – ORGANISATION INDEXEE (DISQUE, DISQUETTE)

C'est un fichier dans lequel chaque enregistrement est caractérisé par la valeur d'une clef spécifique (indicatif) figurant dans l'enregistrement.

Pour chaque clé, un index est maintenu. Il existe donc des Tables d'index associées au fichier lui-même. Chaque index permet l'accès à un enregistrement quelconque du fichier. Le fichier est créé de telle manière qu'il puisse aussi être utilisé en séquentiel pur.

CHAPITRE XIV

PRIMITIVES SUR LES FICHIERS EN COBOL

****

I – OUVERTURE DE FICHIER OPEN

1) OPEN OUTPUT nom-de-fichier : ordre utilisé pour une création de fichier (M.C. ➔ périphérique).

2) OPEN INPUT nom-de-fichier : utilisé pour la lecture (consultation) de fichier (périphérique ➔ M.C.).

3) OPEN I-O nom-de-fichier : utilisé pour mettre à jour des enregistrements d'un fichier déjà existant (périphérique ⟺ M.C.).

3) OPEN EXTEND nom-de-fichier : utilisé pour rajouter de nouveaux articles à un fichier déjà existant.

II – FERMETURE DE FICHIER CLOSE

C'est une des instructions les plus courtes à écrire :

CLOSE nom-de-fichier

mais elle prend beaucoup de temps à l'exécution.

Il est possible, dans un programme, d'ouvrir et de fermer un fichier plusieurs fois, MAIS :

* on ne peut ouvrir un fichier que s'il est fermé

* on ne peut fermer un fichier que s'il est ouvert.

Au début de l'éxécution d'un programme, tous les fichiers sont fermés.

III – PRIMITIVES ET ORGANISATION

1 – FICHIERS SEQUENTIELS

OPEN { OUTPUT / EXTEND } (tout support) permet l'ouverture d'un fichier pour utiliser

WRITE (écriture)

OPEN INPUT (tout support) permet la lecture séquentielle par l'ordre READ

OPEN I-O (disque, disquette) permet la mise à jour par l'utilisation successive :

- d'une lecture (READ)

- d'une modification de l'enregistrement lu

- d'une réécriture (REWRITE)

2 - FICHIERS RELATIFS ET SEQUENTIELS INDEXES

OPEN OUTPUT : WRITE (écriture ➔ création)

OPEN OUTPUT : READ (lecture ➔ consultation)

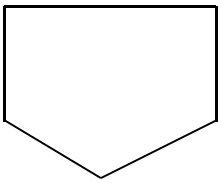
OPEN I-O : 1) READ (lecture ➔ consultation)

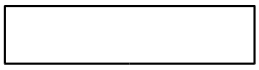
2) WRITE (ajout d'un nouvel enregistrement)
READ

3) Modification (modification du dernier enregistrement lu)
REWRITE

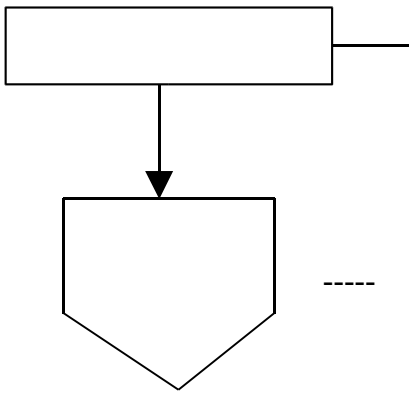
4) DELETE permet de détruire logiquement un enregistrement.

IV – TRAITEMENT COURANTS SUR LES FICHIERS

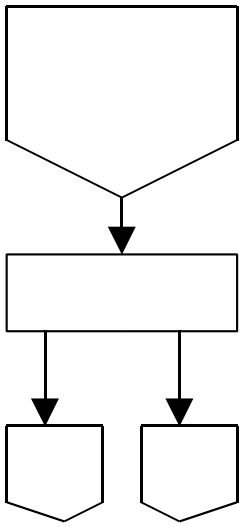
Notations  fichier situé sur un support informatique

 opération manuelle ou exécution d'un programme

1) CREATION



2) CONTROLE



------ Fichier à contrôler

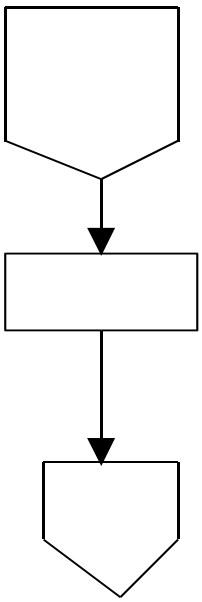
----- Programme de contrôle

fichier sans ---- erreurs

----- Fichiers des erreurs

3) DUPLICATION

Cette opération est essentielle pour les sauvegardes



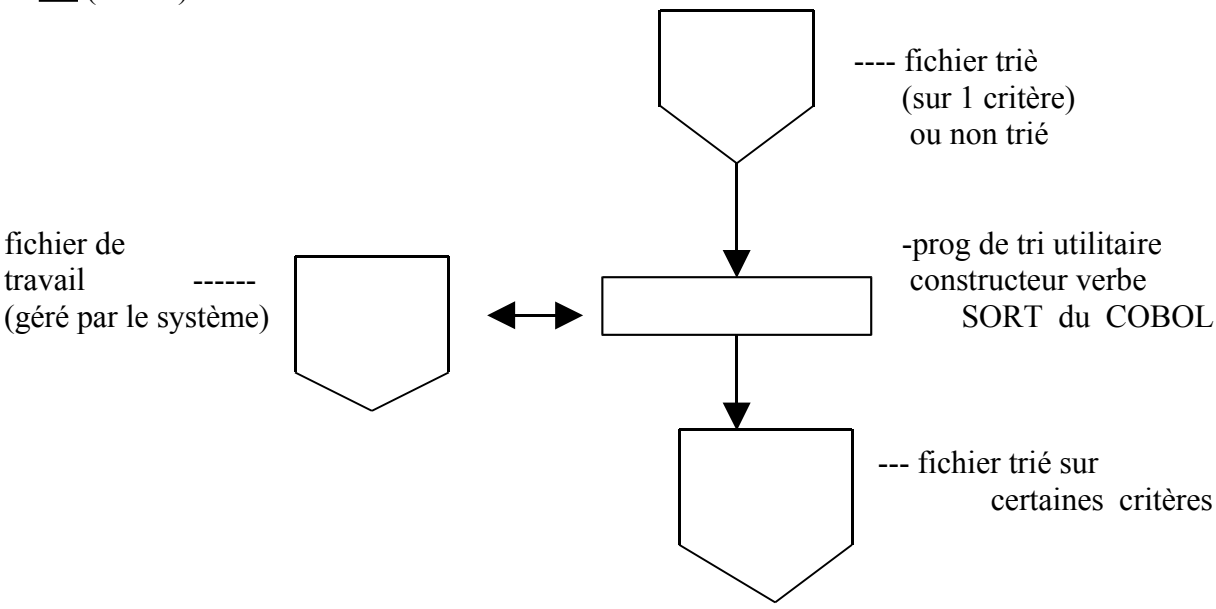
------- -version ‘’père’’ du fichier

-programme utilitaire fourni par le constructeur (COPY)

------ -prog COBOL écrit par l’utilisateur

------ version ‘’fils’’ du fichier

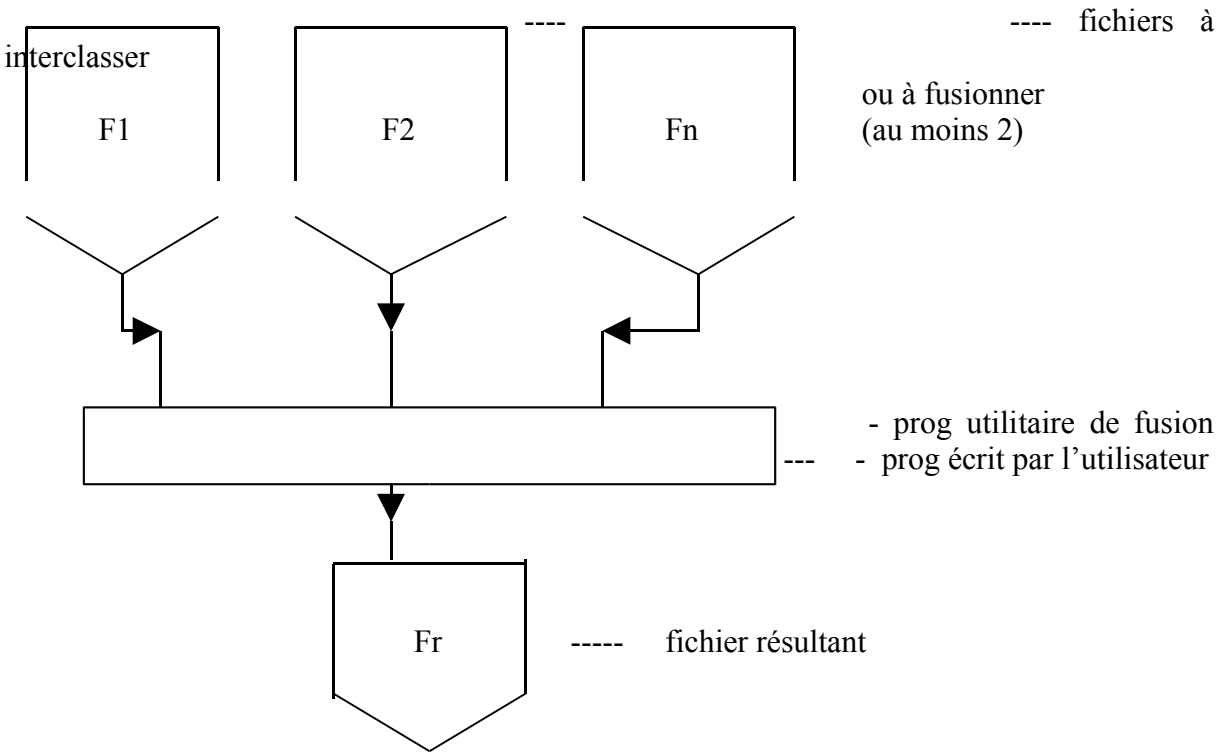
4 -Tri (SORT)



- 2 niveaux de tri : critère majeur, mineur

- 3 niveaux de tri : critère majeur, intermédiaire, mineur.

5 Fusion (MERGE)



6 Eclatement (plus rare)

Schéma inverse du précédent, en général c'est le même utilitaire avec d'autres options.

7 Mise à jour fichier Séquentiel

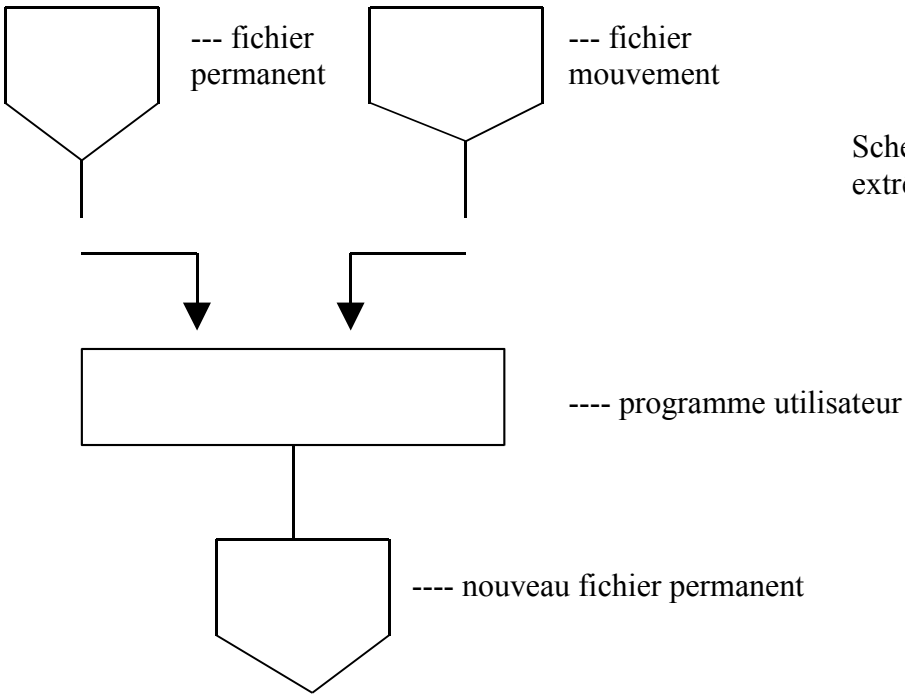


Schéma extrêmement fréquent

8 Mise à jour fichier relatif ou séquentiel indexé

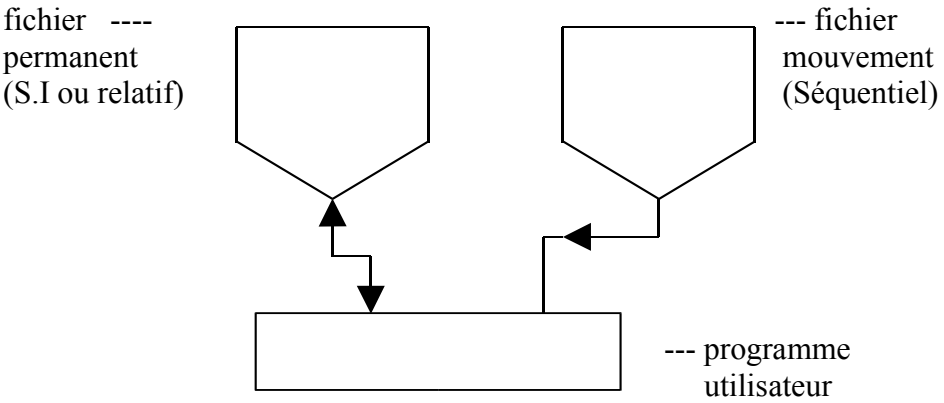


Schéma très fréquent

CHAPITRE XV

TRAITEMENT DES FICHIERS SEQUENTIELS EN COBOL

****

I – INTRODUCTION

Pour chaque fichier, il faut définir

- le type de support
- l'organisation physique
- le facteur de groupage
- le type d'accès (relatif, séquentiel indexé)
- la longueur et la structure des enregistrements.

Ces descriptions se font d'une part dans **l''ENVIRONMENT DIVISION''**, de l'autre dans la ''**DATA DIVISION''.**

II – DESCRIPTION DANS L'ENV. DIV.

A l'intérieur de cette division on crée une section (INPUT-OUTPUT SECTION).

Et un paragraphe FILE-CONTROL. Tous deux écrits en colonne 8.

Pour chaque fichier utilisé dans le programme on va trouver dans ce paragraphe une description commençant par le mot SELECT.

FORMAT GENERAL :

SELECT nom-de-fichier ASSIGN TO nom-externe (1)

[ORGANIZATION IS SEQUENTIAL]

[FILE STATUS IS nom-donnée-1]

* Sur l'AS/400 un nom externe courant aura la forme DATABASE-nom-de-fichier (fichier sur disque) ou PRINTER-QSYSPRT (fichier imprimante).

La clause FIL STATUS sera vue ultérieurement.

III – DESCRIPTION DANS LA DATA DIVISION

A chaque fichier est associée, dans une nouvelle Section (FILE SECTION), une rubrique FD (File description). Cette dernière comporte un certain nombre d'attributs ainsi qu'une ou plusieurs rubriques de description de données, décrivant le ou les articles du fichier.

FORMAT GENERAL

FD nom-de-fichier

[VALUE OF FILE IS { n-d-1 / littéral }]

[BLOCK CONTAINS entier-1 { RECORDS / CHARACTERS }]

LABEL RECORD { OMITTED / IS STANDARD }

[DATA RECORD IS Nom-Enr-1 Nom-Enr-2…]

[LINAGE Clause] ➔ à voir plus loin.

01

[description de structure

classique mais

sans VALUE]

L'ordre d'écriture des clauses de la rubrique FD n'a pas d'importance.

La clause DATA RECORD fournit, à titre de documentation le ou les noms d'articles associés au fichier.

Seule la clause LABEL est obligatoire.

Cas particulier d'un fichier imprimante (utilisation de la clause LINAGE)

Le cobol 74 a introduit pour les fichiers d'édition certaines possibilités de mise en page :

Clause LINAGE :

LINAGE IS { n-d-1 / entier-1 } LINES

[WITH FOOTING AT { n-d-2 / entier-2 }]

[LINES AT TOP { n-d-3 / entier-3 }]

[LINES AT BOTTOM { n-d-4 / entier-4 }]

Entier-1 : désigne le nombre de lignes maximum écrites sur chaque page de l'état.

Entier-2 : désigne le numéro de ligne à partir de laquelle seront éditées les lignes de bas de page (FOOTING). On doit respecter 0 < entier-2 < = entier-1

Entier-3 : désigne le nombre de lignes non écrites en tête de page (TOP)

Entier-4 : désigne le nombre de lignes non écrites en bas de page (BOTTOM)

NB : Page peu utile dans un premier temps

IV – LES VERBES DE TRAITEMENT (PROCEDURE DIVISION)

A – Le verbe OPEN

FORMAT GENERAL :

```
OPEN  [ INPUT          nom-de-fichier-1…. ]
      [ OUTPUT         nom-de-fichier-2…. ]
      [ EXTEND         nom-de-fichier-3…. ]
      [ I-O            nom-de-fichier-4…  ]
```

Il n'est pas obligatoire d'ouvrir tous les fichiers en même temps. Mais chaque fichier utilisé doit être ouvert avant utilisation.

EXTEND permet d'ajouter des enregistrements en fin de fichier.

I-O permet à la fois de lire (READ) des enregistrements d'un fichier puis de les réécrire à la même place par un ordre REWRITE.

B – Le verbe CLOSE

Un ordre de fermeture doit être donné pour tous les fichiers utilisés, donc ouverts, avant la fin du programme.

FORMAT GENERAL :

```
CLOSE          nom-de-fichier-1…..
```

C – Le verbe READ

Cette instruction (utilisable après un OPEN INPUT ou I-O) délivre un enregistrement logique au programmeur par l'intermédiaire de la description faite en FILE SECTION.

FORMAT GENERAL :

```
READ nom-de-fichier RECORD INTO nom-donnée
AT END instruction
END-READ.
```

La clause AT END est obligatoire dans presque tous les systèmes, [mais sur l'IBM-PC, elle est optionnelle]. Toutefois il est conseillé de toujours la mettre.

La clause INTO nom-donnée signifie que le programmeur ne désire pas travailler en zone ''buffer'', mais dans une zone de WORKING-STORAGE SECTION ou de FILE SECTION. C'est le cas lorsque l'on doit comparer des enregistrements successifs.

D – Le verbe WRITE

Ce verbe permet d'écrire un enregistrement logique sur un fichier, à partir de ce qui est fabriqué en mémoire centrale.

On se méfiera ici de la syntaxe COBOL :

- On lit un FICHIER
- On écrit un ENREGISTREMENT

FORMAT GENERAL

WRITE nom-d'enreg [FROM n-d-1]

[BEFORE / AFTER] ADVANCING { n-d-2 / entier / PAGE } LINES

END-WRITE.

Pour les fichiers sur disques la forme la plus courante est WRITE ENREG, ENREG étant défini au niveau 01 de FILE SECTION.

FROM n-d-1

revient à effectuer un MOVE depuis n-d-1 vers la zone ''buffer'' avant l'ordre WRITE.

Les autres options concernent l'édition (donc des fichiers en ASSIGN ….. PRINTER).

Il faut choisir une option AFTER ou BEFORE et la garder pour tous les WRITE sur le même fichier. ''entier'' désigne le nombre de lignes blanches avant l'écriture (AFTER) ou après l'écriture (BEFORE).

La valeur zéro implique une surimpression. Entier est limité à 120.

PAGE commande le passage à la page suivante.

Si la clause LINAGE a été utilisée en FD, l'imprimante est positionnée en fonction de cette clause.

E – Le Verbe REWRITE

FORMAT GENERAL

REWRITE nom-enregistrement FROM n-d-a
END-REWRITE.

L'enregistrement précédemment lu par un ordre READ est réécrit après modifications éventuelles à la même place (supports disquettes ou disques).

CHAPITRE XVI

USAGES ETENDUS DES VERBES ACCEPT, DISPLAY

LES COMPTEURS SPECIAUX

******

I – LE VERBE ACCEPT

Nous avons déjà vu le format le plus simple (ACCEPT) nom-donnée), il en existe deux autres :

A :

ACCEPT nom-donnée-1 FROM { DATE / TIME } DAY

Instruction qui permet d'affecter à nom-donnée-1 la valeur

- du compteur DAY (sous la forme YYQQQ. c.a.d. année, quantième de l'année)

- du compteur DATE (sous la forme YYMMJJ . c.a.d. année, mois, jours)

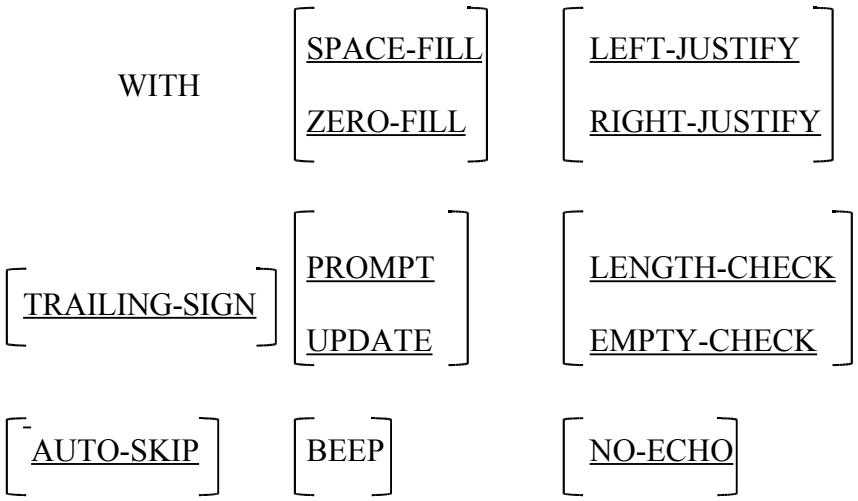
- du compteur TIME (sous la forme HHMMSSCC, heures, minutes, secondes, centièmes)

Il existe aussi la possibilité d'aller chercher une donnée particulière sur le pupitre principal du système (CONSOLE). Exemple :

ACCEPT nom-donnée-1 FROM CONSOLE.

B - *

ACCEPT position sur écran nom-donnée



- **ATTENTION , ce paragraphe B n'est utile que pour le COBOL-MICROSOFT sur mini ordinateur ! !**

Ce format est destiné à l'entrée de données à partir d'un écran lorsqu'on n'utilise pas la gestion « plein écran » (SCREEN SECTION).

Etudions dans l'ordre ces divers attributs :

1) – Position sur écran

Le 1er caractère (à gauche) de la zone est défini par ses coordonnées ligne (LIN) et colonne (COL) suivant le format :

(LIN [{ + } entier-1] , COL [{ + } entier-3])

(entier-2 , entier-4)

LIN, COL sont des compteurs spéciaux COBOL, qu'il ne FAUT pas déclarer et qui ne sont pas affichables directement.

ENTIER-2, ENTIER-4 indiquent un adressage direct du curseur.

L'usage de LIN+2 ou COL-4 signifie que l'on se positionne 2 lignes plus bas ou 4 colonnes plus à gauche par rapport à la dernière position du curseur.

Si l'on omet la position de LIN, ou celle de COL, on reste au même niveau sur la coordonnée manquante.

2) – SPACE-FILL

S'adresse aux zones alphanumériques et signifie que les caractères non remplis sont complétés par des espaces.

3) – ZERO-FILL

S'adresse aux zones numériques non remplies entièrement et complétées par des zéros.

4) – RIGHT JUSTIFY

Permet de cadrer à DROITE une zone de caractères (habituellement cadrée à gauche)

5) – TRAILING-SIN

Signifie que le signe + ou - apparaîtra à la droite de la zone au lieu de la gauche.

6) – PROMPT

- Zone alphanumérique : des points désignant l'étendue des caractères de la zone.
- Zone numérique : un tiret pour chaque chiffre, un point pour marquer le point décimal, un blanc pour le signe.

7) – UPDATE

Signifie que l'on affiche la valeur à modifier de nom-donnée.

8) – LENGTH-CHECK

Impose le remplissage intégral de la zone.

9) – EMPTY-CHECK

Impose d'entrer au moins un caractère

10) – AUTO-SKIP

Evite le return dès que la zone d'écran est remplie.

11) – BEEP

Evident ! prévient l'utilisateur par une sonnerie qu'il faut entrer la donnée.

12) – NO-ECHO

Marque chaque caractère entré par une astérisque (*)

Exemple

```
05  RS  PIC   X(8)
```

(Contenu initial 'ABCDEFGH')

Si l'on exécute ACCEPT (1,b1) RS WITH PROMPT

	Situation de l'écran
	……
Opérateur entre le N	N…….
Opérateur entre O,N,E	NONE……
Opérateur tape Return	NONEbbbb

Valeur en mémoire NONEbbbb

II – LE VERBE DISPLAY (USAGE réservé au COBOL Microsoft sur micro-ordinateur

DISPLAY [position-sur-écran] { ERASE / identificateur / littéral }

[UPON nom-réservé]

→ précisé dans le paragraphe

SPECIAL NAMES (par exemple CONSOLE)

- position-sur-écran a la même signification que pour ACCEPT

- ERASE permet d'effacer l'écran depuis la position spécifiée jusqu'à la fin.

Exemples :

DISPLAY (1,b 1) ERASE efface tout l'écran

DISPLAY (10, b 12) ZONE affiche ZONE sur la 10è ligne à partir de la 12è colonne

III – LES COMPTEURS SPECIAUX

Ce sont des identificateurs que le programmeur peut utiliser sans avoir à les décrire en DATA DIVISION.

1) LINAGE-COUNTER :

Désigne un compteur entier sans signe généré automatiquement par la clause LINAGE. Ce compteur suit ensuite le positionnement des lignes dans la page imposé par l'ordre WRITE.

2) DATE :

Désigne une zone PIC 9(6) contenant la date du jour sous la forme déjà expliquée YYMMJJ.

3) DAY :

Désigne une zone PIC 9(5) contenant la date du jour sous la forme YYQQQ.

4) TIME :

Désigne une zone PIC 9(8) contenant l'heure sous la forme HHMMSSCC.

Ces compteurs sont normalisés depuis 1974, mais il est possible de les trouver sous d'autres noms sur certains systèmes.

CHAPITRE XVII

STRUCTURE DETAILLEE DU PROGRAMME COBOL

****

Ce chapitre permet de consolider la vue d'ensemble sur la structure complète d'un programme COBOL.

I – IDENTIFICATION DIVISION

IDENTIFICATION DIVISION.
PROGRAM-ID.b nom-programme.
[AUTHOR. b rubrique commentaires.]
[INSTALLATION. b rubrique commentaires.]
[DATE-WRITTEN. b rubrique commentaires.]
[DATE-COMPILED. b rubrique commentaires.]
[SECURITY. b rubrique commentaires.]

II – ENVIRONMENT DIVISION

Elle existe dans tout programme COBOL et contient

- des informations relatives à l'ordinateur utilisé (''CONFIGURATION SECTION'')
- des informations relatives aux traitements des Entrées Sorties (''INPUT-OUTPUT SECTION'')

D'où le format général de cette division :

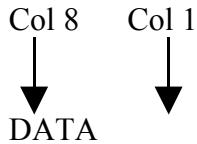
ENVIRONMENT DIVISION.
[CONFIGURATION SECTION].
[SOURCE-COMPUTER. b rubrique d'ordinateur source].
[OBJECT-COMPUTER. b rubrique d'ordinateur résultant].
[SPECIAL-NAMES. b rubrique de noms spéciaux.]
    [DECIMAL POINT IS COMA].
[INPUT-OUTPUT SECTION.]
FILE-CONTROL. b rubrique de FILE-CONTROL.
    (phrases SELECT)

[I-O-CONTROL. b rubrique d'I-O-CONTROL.]

III – DATA DIVISION

Cette division fondamentale en COBOL comporte deux sections principales FILE SECTION, WORKING-STORAGE SECTION et deux sections moins fréquemment utilisées, LINKAGE SECTION et SCREEN SECTION.

L'organisation générale est :



DATA DIVISION

FILE SECTION

FD Nom-de-fichier (Description de fichier)

1 Nom d'enregistrement (Description d'enregistrement)

WORKING-STORAGE SECTION.

77 Nom-donnée (Description de zones élémentaires :

. indépendantes).

01 Nom-de-groupe –Description des structures).

LA LINKAGE SECTION, qui permet de déclarer des zones de mémoire utilisées par plusieurs programmes successifs communicants sera vue en détail dans le chapitre sur les sous programmes externes.

LA SCREEN SECTION permet de gérer les données affichées sur l'écran sous forme de « grille d'écran ». Elle sera étudiée en détail ultérieurement.

IV – PROCEDURE DIVISION

Elle regroupe les véritables instructions spécifiques au traitement.

Comme les autres divisions, elle peut-être divisée en SECTIONS, mais c'est relativement peu utilisé.

Les noms paragraphes servent d'adresses d'instructions du programme en mémoire centrale et apparaissent en marge A dans cette partie.

Les instructions sont des phrases COBOL, inscrites à partir de la marge B.

Signalons, qu'éventuellement, le programmeur peut débuter par des sections DECLARATIVES, détaillées ultérieurement elles aussi.

CHAPITRE XVIII

LES FICHIERS
EN ORGANISATION SEQUENTIELLE INDEXEE

****

I – INTRODUCTION

Cette organisation consiste à permettre la recherche d'un enregistrement logique par consultation de tables successives.

Les consultations sont réalisées à l'aide d'une CLEF D'ENREGISTREMENT (RECORD KEY) présente physiquement dans l'enregistrement logique et de tables de clefs.

Il résulte que chaque fichier logique séquentiel indexé est scindé en 2 fichiers physiques :

Un fichier données :

Qui comporte les informations contenues dans les enregistrements logiques.
Devant chaque enregistrement se trouvent 3 caractères

-2 caractères pour un compteur de longueur

-1 caractère DRAPEAU pour repérer les enregistrements détruits par l'ordre DELETE.

Un fichier CLEFS :

Qui porte le même nom que le fichier "données" suivi du suffixe KEY. *

- **Uniquement valable en Microsoft-Cobol**

II - MODIFICATION APPORTEES DANS LES DIVISIONS COBOL

A – ENVIRONMENT DIVISION

En INPUT-OUTPUT SECTION, FILE-CONTROL, on nomme tous les fichiers utilisés dans le programme, ce qui permet de spécifier le mode d'organisation, le mode d'accès concernant ces fichiers.

FORMAT GENERAL DU SELECT.

SELECT nom-de-fichier ASSIGN TO nom-externe

ORGANIZATION INDEXED

[FILE STATUS IS nom-donnée-1]

RECORD KEY nom-donnée-2

[ACCES MODE { IS SEQUENTIAL / RANDOM / DYNAMIC }]

Les deux clauses ORGANIZATION…., RECORD….. sont obligatoires. Nom-donnée-2 doit être présent dans tout enregistrement et on doit en retrouver la description en FILE-SECTION.

La longueur de cette clef peut aller jusqu'à 60 car. (variable avec chaque système).

SEQUENTIAL, option par défaut est réservé au traitement séquentiel il permet :

- la création séquentielle d'un fichier (en respectant l'ORDRE CROISSANT des CLEFS)

- la lecture séquentielle

- le traitement séquentiel en 1-0 avec modifications ou suppression d'enregistrements.

RANDOM est réservé au traitement aléatoire du fichier séquentiel indexé

DYNAMIC permet à la fois l'accès séquentiel ou aléatoire du fichier

B – DATA DIVISION

La clause LABEL RECORD STANDARD est obligatoire puisque de tels fichiers sont stockés sur disques.

La clause VALUE OF FILE-ID est présente pour les mêmes raisons.

C – PROCEDURE DIVISION

1) OPEN { INPUT / OUTPUT / I-O }

L'ouverture en INPUT doit être faite avant le premier READ de lecture ou START de positionnement, en OUTPUT avant le premier WRITE d'écriture, en I-O s'il s'agit d'un fichier en lecture-écriture utilisant READ, START et REWRITE.

2) READ nom-de-fichier [NEXT] [INTO nom-donnée]

[{ AT END / INVALID } KEY instruction COBOL] END-READ

NEXT est à utiliser dans le cas de lecture séquentielle d'un fichier en DYNAMIC. Dans ce cas, c'est la clause AT END qui doit être utilisée.

INVALID KEY est indispensable en accès RANDOM ou DYNAMIC.

Il faut avoir chargé dans le nom correspondant à RECORD KEY la valeur qui permet de rendre valide le READ. Si la clef fournie n'est pas valide, il y a déclenchement de l'INVALID KEY.

3) START nom-de-fichier [KEY { EQUAL / NOT LESS / GREATER } nom-donnée de la RECORD KEY]

[INVALID KEY instruction COBOL] END-START.

Sert à éviter de lire tout un fichier si l'on ne désire traiter qu'un sous-ensemble des informations qu'il contient. Ce qui revient à sélectionner une clef de départ. Il faut donc charger, là aussi, la valeur choisie dans le nom de la RECORD KEY.

4) { WRITE / REWRITE } nom-d'enregistrement [FROM nom-donnée]
[INVALID KEY instruction COBOL] { END-WRITE / END-REWRITE }

Le verbe WRITE permet :

- en accès séquentiel de créer un fichier

- en accès RANDOM ou DYNAMIC, il faut avoir chargé dans la RECORD KEY la valeur de la clef de l'enregistrement à écrire.

La clause INVALID KEY permet de prévenir des incidents tels que clefs non comprises, non ordonnées ou disque plein (voir rôle de la clause FILE STATUS).

Le verbe REWRITE permet de réécrire un enregistrement déjà existant dans le fichier. En séquentiel la clause INVALID KEY est inutile.

5) DELETE nom-de-fichier RECORD [INVALID KEY
instruction COBOL] END-DELETE

- en séquentiel DELETE permet d'éliminer un enregistrement préalablement lu par un READ.

- en RANDOM. DYNAMIC, on élimine un enregistrement dont la clef est égale au contenu de la RECORD KEY.

6) CLOSE nom-de-fichier rien de spécial !

III – COMPREHENSION DU MECANISME SEQUENTIEL INDEXE

A – organisation interne

A la création, on charge les enregistrements sur disque, DANS L'ORDRE des clefs croissantes mais en séquence. Le système se constitue des tables d'INDEX à plusieurs niveaux.

- éventuellement : le volume (MASTER INDEX chez IBM)

- au niveau des "cylindres"
- au niveau des "pistes"

Ces tables d'index permettent la recherche d'un enregistrement logique par consultations successives. Cela ressemble un peu à la consultation d'un bottin.

Les tables d'index sont constituées des clefs de "fin de volume", de "fin de cylindre" de "fin de pistes".

B – Exemples d'utilisation

(en rapport avec le transparent correspondant)

1 Sert à lire l'enregistrement de clef 470

- recherche dans la table d'index/cylindre ∏ CYL 1

- recherche dans la table d'index/pistes ∏ PISTE 3

- recherche par associativité sur la piste 3 du CYL 1.

2 Sert à ajouter l'élément 425

- recherche dans la table index/cylindres ∏ CYL 1

- recherche dans la piste de la phrase ∏ PISTE 2

- recherche dans la piste de la phrase ∏ c'est complet

- écriture du 444 en débordement

- décalage vers la droite du 430

C – Avantages et Inconvénients

Il y a peu de contraintes pour créer l'organisation séquentielle indexée (clefs d'enregistrements uniques et présentes à la création en ordre croissant).

La programmation est très aisée, le fichier est facile à maintenir.

Les ajouts et les suppressions sont très simples à programmer

Par contre, la recherche d'un élément peut-être relativement lente dans le cas d'un gros fichier (tables d'index).

Le problème de débordements peut faire chuter les temps de recherche.

Il y a aussi nécessité de remettre en état le fichier lorsque les pistes ou cylindres de débordement menacent d'être saturés.

CHAPITRE XIX
LES FICHIERS EN ORGANISATION RELATIVE
LA CLAUSE FILE STATUS

I – INTRODUCTION

L'organisation relative consiste à désigner un enregistrement logique de fichier sur mémoire à accès sélectif par le NUMERO RELATIF d'enregistrement, compté à partir de 1 pour le premier enregistrement du fichier. UN ENREGISTREMENT LOGIQUE EST DONC Défini exclusivement par son numéro d'ordre.

On retrouve ici une technique d'utilisation assez proche de la nature de vecteur de structures.

Malheureusement, si cette organisation est la plus performante des organisations de fichier sur mémoire à accès relatif, elle est difficile à réaliser dans la pratique car il est assez RARE de pouvoir désigner un enregistrement par son rang (sauf si une technique de HASH-CODE (*) est appliquable facilement).

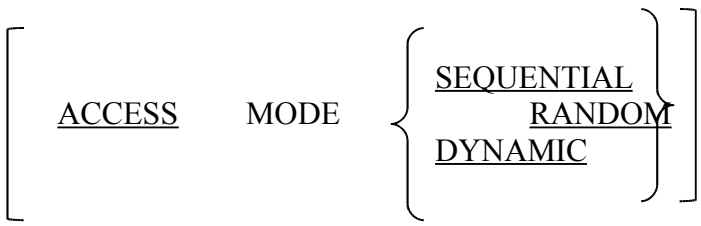
Dans cette organisation, seuls les enregistrements de longueur fixe non bloqués sont autorisés. Le rang de l'enregistrement logique est formé dans une zone de donnée RELATIVE KEY définie en WORKING-STORAGE SECTION sous forme d'un entier non signé.

(*) Technique vue par ailleurs dans le cours d'algorithmique.

II – MODIFICATIONS EN ENVIRONMENT DIVISION

Toujours en FILE-CONTROL de l'INPUT-OUTPUT SECTION, voici le format général des phrases SELECT :

```
SELECT  nom-de-fichier  ASSIGN  TO  nom-externe
        ORGANIZATION RELATIVE
        [FILE STATUS nom-donnée]
        [RELATIVE KEY IS  nom-donnée-1]      *  (1)
```



La clause d'organisation est obligatoire, ainsi que la RELATIVE KEY.

Les mêmes règles d'utilisation vues pour les fichiers en séquentiel indexé s'appliquent aux fichiers en relatif.

III – MODIFICATIONS EN DATA DIVISION

- en FILE SECTION, on retrouve la clause obligatoire.

LABEL RECORD STANDARD

l'attribut VALUE OFFILE-ID { nom-donnée / littéral }

est utilisé sur micro-ordinateur essentiellement.

*** (1) Obligatoire seulement si la clause ACCESS A SELECTIONNE RANDOM ou DYNAMIC**

- EN WORKING-STORAGE SECTION on doit trouver la description de nom-donnée-1 :

77 nom-donnée-1 PIC 9(m)

IV – LES VERBES DE PROCEDURE DIVISION

1) OPEN

OPEN { INPUT / OUTPUT / I-O } nom-de-fichier

Les fichiers en organisation relative doivent être ouverts en :

- INPUT avant le premier READ ou le START de positionnement

- OUTPUT avant le premier WRITE pour créer un nouveau fichier

- I-O s'il s'agit d'un fichier en lecture-écriture utilisant READ, START et REWRITE.

2) READ

a) READ nom-de-fichier [NEXT] [INTO nom-donnée]

[AT END instruction COBOL] END-READ

NEXT est utilisé dans le cas de lecture séquentielle d'un fichier déclaré en DYNAMIC.

b) READ nom-de-fichier [INTO nom-donnée]

[INVALID KEY instruction COBOL] qui est l'ordre de lecture correspondant à l'accès "direct" à un enregistrement. END-READ

3) START

Même syntaxe et même utilisation qu'en séquentiel indexé, mis à part le fait que l'on doit charger la relative key et non la record key.

4) WRITE, REWRITE, DELETE, CLOSE

Voir détails d'utilisation dans le chapitre séquentiel indexé

5) FILE STATUS*

Cette option que l'on a vue apparaître dans les trois types d'organisation de fichier permet de contrôler le bon achèvement d'une entrée-sortie grâce à l'indicateur précisé derrière FILE STATUS

Cet indicateur désigne une zone PIC XX, à déclarer en DATE DIVISION.

1) En séquentiel, il faut interpréter ces 2 caractères de la façon suivante :

00 → opération d'entrée/sortie réussie.
10 → fin du fichier
30 → erreur permanente
34 → espace disque plein
91 → fichier endommagé

2) En séquentiel indexé

Le FILE STATUS peut prendre les valeurs suivantes :

00 → entrée/sortie réussie.
10 → fin de fichier
21 → clef hors séquence
22 → tentative d'écrire un enregistrement dont la clef existe déjà
23 → l'enregistrement cherché n'existe pas
24 → espace disque plein
30 → erreur permanente
91 → fichier endommagé

3) En organisation relative

LE FILE STATUS peut prendre les valeurs suivantes :

00 → entrée/sortie réussie.
10 → fin de fichier
22 → l'enregistrement à écrire (créer) existe déjà
23 → l'enregistrement recherché n'existe pas
24 → espace disque plein
30 → erreur permanente
91 → fichier endommagé.

*** Il faut interpréter les valeurs figurant dans cette page comme utilisables pour le COBOL MICROSOFT. Le principe reste identique en AS/400, la liste des valeurs figurent dans la notice COBOL AS/400 User'sGuide.**

CHAPITRE XX – COMPLEMENTS SUR LES INSTRUCTIONS ARITHMETIQUES NOMS CONDITIONS

I – OPTIONS COMMUNES AUX INSTRUCTIONS ARITHMETIQUES.

A – GIVING nom-donnée

Cette option est identique à l'option INTO de la clause READ, c'est-à-dire que le résultat de l'opération est enregistré dans nom-donnée.

B – ROUNDED

Après une opération (division par exemple) le résultat est rarement entier et la quantité 243,427 devient 243,42 si la PICTURE est 999V99.

Avec ROUNDED, COBOL ajoute 5 au premier chiffre non significatif de droite pour réaliser l'arrondi.

Avec ROUNDED 243,427 devient 243,43 avec la même PICTURE.

C – ON SIZE ERROR instruction

Cette option permet de prévenir les incidents de calcul, par exemple :

- la division par zéro

- un dépassement de capacité de zone

Ex – DIVIDE A INTO B ON SIZE ERROR MOVE ZERO TO B END-DIVIDE.

- ADD 1 TO CPT ON SIZE ERROR…. END-ADD.

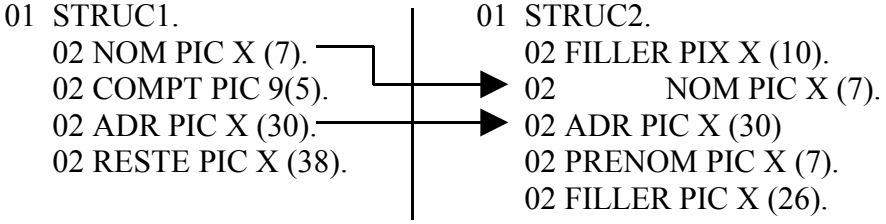
permet d'éviter les "dérapages".

II – ADD, SUBTRACT et MOVE supportent l'option CORRESPONDING ou CORR

Cette option peut-être utilisée sur des noms-de groupe comportant des zones élémentaires de NOMS IDENTIQUES.

(Attention : clauses OCCURS et REDEFINES ignorées)

Exemples :



L'instruction MOVE CORR STRUC1 TO STRUC2 permet de remplir les zones de même nom de STRUC2 par celles de STRUC1.

L'option ne marche avec ADD et SUBTRACT que si les PICTURE correspondantes sont numériques. Pour distinguer deux zones de même nom on les qualifie par rapport au 01.

MOVE "ALFRED" TO NOM OF STRUC1.

III – LE VERBE COMPUTE

Afin de permettre quelques incursions dans le domaine scientifique, on peut utiliser le verbe COMPUTE :

COMPUTE Nom-donnée-1 [ROUNDED] =

{ nom-donnée-2 / litt-numérique-2 / expr. arith. } [ON SIZE ERROR instruction].

Cette instruction n'est à utiliser qu'en cas de véritables complexités

Exemples :

COMPUTE DELTA = (B*B) – (4*A*C)

Est justifié

Par contre

COMPUTE I = 1 + 1
COMPUTE PI = 3.14159

Sont à éviter, puisqu'il existe des instructions moins coûteuses pour arriver au même résultat.

En COBOL, il vaut mieux utiliser les expressions correctement parenthèsées car il est rarement indiqué comment les compilateurs utilisent la hiérarchie :

Niveau		0 =
"	1	( )
"	2	+ ou – unaire
"	3	**
"	4	*,/
"	5	+,-

IV – NIVEAU 88 – LES NOMS-CONDITIONS

Le nombre-niveau 88 sert à spécifier une valeur, une liste de valeur, ou un intervalle qu'un élément peut couvrir. Si l'élément prend une telle valeur, le nombre-condition correspondant est vrai.

VALUE	IS	litt1	[litt2---]
VALUES	ARE	litt1	THRU litt-2

Ex1

```
02      PAYROLL-PERIOD PIC 9.

        88      WEEKLY VALUE 1.
        88      SEMI-MONTHLY VALUE 2.
        88      MONTHLY         VALUE 3.
```

En procédure Division on trouvera

```
IF MONTHLY GO TO DO-MONTHLY au lieu de
IF PAYROLL-PERIOD = 3…..
```

Ex2

```
02      NJOB  PIC 9(4).

        88      PROF VALUE 1 THRU 25.
        88      PREM VALUE 1000 THRU 1500.
        88      DEUX VALUE 2000 THRU 2200.

                        IF PROF GO TO TRAVAIL-PROF.
```

CHAPITRE XXI – LES AUTRES UTILISATIONS DE PERFORM
TRADUCTION DE L'INSTRUCTION CAS

1 – PERFORM Simple (rappel)

PERFORM nom par-1 [THRU nom par-2] END-PERFORM.

C'est l'utilisation (à distance) d'un ou de plusieurs paragraphes rédigés une seule fois et utilisés autant de fois que nécessaire.

Une variante introduit une REPETITION :

PERFORM nom par-1 [THRU nom-par-2] { nom-d / entier } TIMES

END-PERFORM.

II – Le PERFORM….. UNTIL

C'est une instruction COBOL qui permet de traduire la structure itérative TANT QUE de l'algorithmique mais qui, dans l'esprit est en fait équivalente au REPEAT…. du PASCAL.

PERFORM nom-par-1 [THRU nom-par-2] UNTIL condition

END-PERFORM.

Ce que nous interprétons comme : "traiter le paragraphe 1 jusqu'au paragraphe 2 jusqu'à ce que la condition soit satisfaite".

Ex

(morceau de programme visant à calculer une racine carrée)

```
|
|
(A)     (B)
        PERFORM CALCUL UNTIL ECART 0.01          END-PERFORM.
        |
        |
CALCUL.
        MOVE X TO XO.
        COMPUTE X = (XO + (A/XO))/2.
        SUBTRACT XO FROM X GIVING ECART.
        |
        |
        |
```

III – PERFORM….. VARYING

PERFORM paragraphe-1 [THRU] paragraphe-2]

(1er indice)

[VARYING] nom-donnée-1 FROM { nom-donnée-11 | entier-11 }

BY { entier-12 | nom-donnée-12 } UNTIL condition-1

(2ème indice)

[AFTER nom-donnée-2 FROM { nom-donnée-21 | entier-21 }

BY { entier-22 | nom-donnée-22 } UNTIL condition-2]

(3ème indice)

[AFTER nom-donnée-3 FROM { nom-donnée-31 | entier-31 }

BY { entier-32 | nom-donnée-32 } UNTIL condition-3]

END-PERFORM

C'est une instruction assez lourde à utiliser, mais elle permet de répéter un paragraphe ou un groupe de paragraphes n fois, sous contrôle de un ou plusieurs indices.

- nom-donnée-I représentent des variables entières positives

→ PICTURE 9 (n)

- nom-donnée-IJ représentent des identificateurs d'entiers positifs

- condition-I représentent des conditions COBOL permettant de s'arrêter sur une valeur particulière des nom-donnée-I

Exemples

1) PERFORM BOUCLE VARYING I FROM 1 BY 1 UNTIL I > 4 END-PERFORM

On passe dans la BOUCLE pour les valeurs

I = 1,2,3 et 4

```
2) PERFORM   MISEAZERO   VARYING  I  FROM  1
   BY  1  UNTIL  I > 4  AFTER  J  FROM  1
   BY  1  UNTIL  J > 3              END-PERFORM
        |
        |
        |
       MISEAZERO.
            MOVE  ZERO  TO  EL (I,bJ)
```

Cette forme génère beaucoup d'instructions machines (à éviter si possible si la machine est de moyenne puissance).

IV – L'INSTRUCTION CAS : GO TO… DEPENDING

GO TO P1 P2…. PN DEPENDING ON Nom-donnée

Nom-donnée est un identificateur de zone numérique entière dont la valeur doit être comprise entre 1 et n.

Si n = i, on se rend au paragraphe PI.

Ex

```
GO  TO  JANV  FEV  MARS…. DEC  DEPENDING ON MOIS.
```

Avec MOIS décrit en DATA DIVISION par

```
77 MOIS PIC 99
```

On a créé un embranchement à douze directions !

ATTENTION ! Si la valeur qui est dans nom-donnée est hors intervalle, le GO TO… DEPENDING est tout simplement ignoré.

CHAPITRE XXII

LA CLAUSE USAGE

(Cobol MICROSOFT et AS/400)

Attention

Ce chapitre est souvent dépendant des matériels utilisé !

Cette clause spécifie le format de la donnée en mémoire ; elle est donc associée à l'usage d'une zone et suit l'attribut PICTURE.

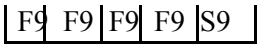
Format Général :

USAGE IS { DISPLAY | COMPUTATIONAL ou COMP | COMPUTATIONAL-0 ou COMP-0 | COMPUTATIONAL-3 ou COMP-3 | COMPUTATIONAL-4 ou COMP-4 }

Le mode DISPLAY correspond à la codification ASCII proprement dite avec un caractère par octet.

Sont donc "USAGE DISPLAY" les données

- alphabétiques A
- alphanumériques X
- numériques externes de la forme



Cette clause est dans ce cas facultative et prise par défaut en absence de tout USAGE.
Elle peut être écrite à tout niveau y compris pour un groupe (où chaque élément se voit appliquer cette représentation interne.
La clause USAGE spécifie donc la manière dont une donnée est stockée dans la mémoire.
Cela N'AFFECTE PAS L'USAGE que l'on peut en faire.
Tous les USAGE de type COMP représentent des données de type NUMERIQUE.

1) COMP : est identique au DISPLAY mais en mode numérique exclusivement.

2) COMPUTATIONAL-0 ou COMP-0 caractérise des données (Micro) numérique en binaire occupant 2 caractères (16 bits). Sur l'IBM-PC ces nombres binaires sont limités à l'intervalle –32768,+32767 et on peut ne pas leur donner une PICTURE.

On peut écrire

```
02 ZONE COMP-0.
```

Ou

```
02 ZONE PIC 99 COMP-0.
```

3) COMPUTATIONAL-4 ou COMP-4 ou BINARY en AS/400 caractérise des données numériques en binaire occupant 4 caractères en mémoire (32 bits). Ces nombres binaires évoluent dans l'intervalle

-2147483648, +2147483647

4) COMPUTATIONAL-3 ou COMP-3 ou COMP sur AS/400 caractérise les données numériques en DECIMAL CONDENSE interne avec 2 chiffres par octet.

Exemple la PICTURE S9(9)V9(4) représentera une zone dont l'image peut être rendue par le schéma suivant

| 99 |99| 99 |99| 99 |9S

En général, la place occupée par une telle donnée EST

P = (n+2) / 2 ou / signifie division entier

S9(5) → 3 octets , S9(8) → 5 octets

5) CHOIX d'une clause USAGE

La plupart du temps on utilise la clause par défaut.

Certains types de données numériques peuvent se voir attribuer un rôle qui exige un attribut COMP-O (sur certaines machines).

Dans les autres cas :

- il peut-être intéressant POUR DES DONNEES subissant BEAUCOUP DE CALCULS de choisir COMP-O ou COMP-4 (peu de temps perdu en conversion).

- pour les tableaux de données numériques de grande dimension, on peut choisir COMP-3 qui permet de diviser la place occupée en machine par un facteur d' à peu près par 2.

(*) Sur l'AS/400, on peut en plus utiliser la clause USAGE avec les attributs suivants :

(1) BINARY ou COMP-4 jusqu'à 4 digits → 2 octets

représentation de 5 à 9 digits → 4 octets

binaire de 10 à 18 digits → 8 octets

(2) PACKED-DECIMAL équivalent à COMP-3 → décimal condensé

CHAPITRE XXIII – LA PROGRAMMATION

"PLEIN ECRAN" (SCREEN SECTION).

(Uniquement pour Microsoft Cobol)

I – INTRODUCTITON

Nous savons déjà dialoguer avec l'écran caractère par caractère ou ligne par ligne.

On peut améliorer le dialogue en imaginant des formats complets incluants des dispositifs standards tels que la double brillance, la sonorisation, la couleur inversée etc…

La SCREEN SECTION comporte des descriptions de données (en affichage ou en saisie) élémentaires ou en groupe. Comme dans la FILE SECTION ou la W-S SECTION ces descriptions sont fournies à l'aide des numéros de niveaux.

II – FORMAT GENERAL

A B

↓ ↓

SCREEN SECTION

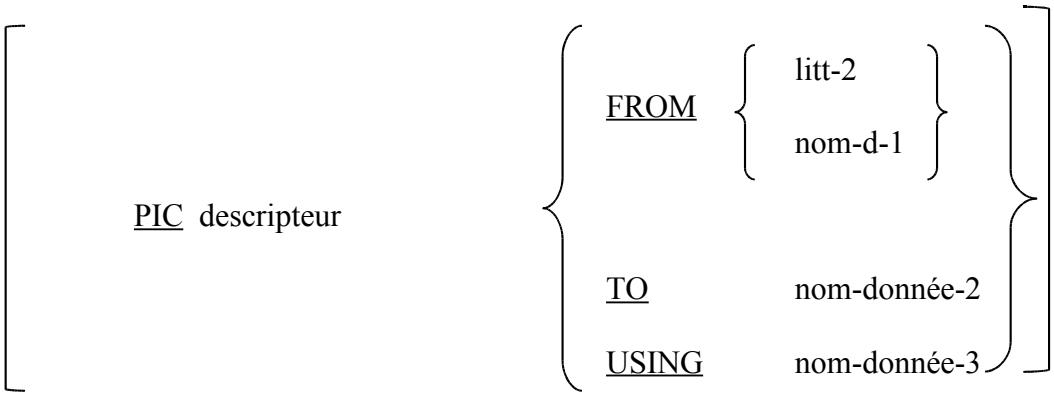
01 Nom-de-grille [AUTO] [SECURE] [REQUIRED]

[FULL]

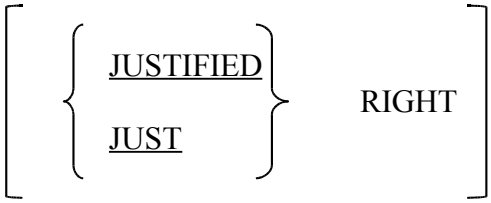
nn [nom-de-zone]

[BLANK SCREEN]
[LINE [PLUS] entier-1]
[COLUMN [PLUS] entier-2]
[BLANK LINE]
[BELL]
[UNDERLINE]
[REVERSE-VIDEO]
[HIGH LIGHT]

[BLINK]
[FOREGROUNG-COLOR entier-3]
[BACKGROUND-COLOR entier-4]
[VALUE littéral-1]



[BLANK WHEN ZERO]



[AUTO] [SECURE] [REQUIRED] [FULL]

Nous allons étudier en détail toutes ces possibilités, mais il est d'abord nécessaire d'indiquer qu'on envoie des données sur l'écran avec l'instruction

DISPLAY nom-de-grille

et que l'on en reçoit par l'instruction

ACCEPT nom-de-grille.

III – LES DECLARATIONS DE GROUPE

Il faut obligatoirement une déclaration d'écran au niveau 01 et il est possible d'associer certains attributs (AUTO, SECURE, REQUIRED, FULL) à tous les éléments du groupe.

IV – LES OPTIONS DE LA SCREEN SECTION

Une zone élémentaire ne peut excéder une ligne d'écran (80 caractères).
Certaines options ne sont valables que pour l'ordre DISPLAY, d'autres que pour ACCEPT.

1) BLANK SCREEN :

Utilisation DISPLAY, l'écran est remis à blanc et le curseur est positionné ligne 1, colonne 1.

2) LINE et COLUMN :

Par définition le groupe de niveau 01 correspond à un positionnement 1,1 du curseur.

LINE n, COLUMN m positionnent sur une ligne et une colonne déterminée.
L'option PLUS indique un déplacement par rapport à la situation présente du curseur.
COLUMN sans LINE déplace le curseur sur la même ligne.
LINE sans COLUMN envoie le curseur au début de la ligne indiquée.

3) BLANK LINE :
Les positions à droite du curseur sont remises à blanc jusqu'au bout de ligne.

4) BELL :

Utilisation ACCEPT déclenche le bruiteur lorsqu'on attend une donnée.

5) HIGHLIGHT, BLINK, REVERSE-VIDEO, UNDERLINE

Déclenchement des effets visuels en utilisation DISPLAY (double intensité, clignotement ou inverse fond et champ, soulignement valable pour écran monochrome).

6) VALUE litt-1

Utilisation DISPLAY, désigne une constante chaîne de caractère qui doit être affichée à l'écran. (En utilisation interdite avec la clause PICTURE).

7) PIC description :

Décrit le format de la donnée affichée à l'écran suivant les règles standard des PICTURE. Toute donnée destinée au, ou venant du programme (options TO, FROM, USING) doit avoir une PICTURE.

FROM { litt-2 / nom-donnée-1 } : utilisation DISPLAY

On affiche le contenu actuel du litt-2 ou de nom-donnée-1 par l'intermédiaire de la grille.

TO nom-donnée-2 : utilisation ACCEPT

Ce qu'on tape au clavier dans ce champ est ensuite rangé dans nom-donnée-2.

USING nom-donnée-3 : indique que la zone peut être MISE A JOUR à l'écran

ATTENTION : nom-donnée- 1,2,3 ne peuvent appartenir à une table !

8) BLANK WHEN ZERO : utilisation DISPLAY même signification qu'en WORKING-STORAGE SECTION.

9) JUSTIFIED
RIGHT
JUST

Les données à afficher ou reçues doivent être cadrées à droite

10) AUTO, SECURE, REQUIRED, FULL : utilisation ACCEPT, elles signifient

a) le curseur saute à la zone suivante lorsque la première est remplie (PICTURE obligatoire).

b) les caractères sont entrés masqués par les astérisques (ex : mots de passe) (PIC obligatoire).

c) signifie qu'au moins 1 caractère doit être entré dans la zone

d) signifie que toute la zone doit être rentrée.

REMARQUE

Attention les attributs du ACCEPT pos-spécification et de la SCREEN SECTION se correspondent de la manière suivante :

AUTO	→	AUTO-SKIP
BELL	→	BEEP
FULL	→	LENGHT-CHECK
JUSTIFIED	→	RIGHT-JUSTIFY
SECURE	→	NO-ECHO
REQUIRED	→	EMPTY-CHECK

V – EXEMPLE DE PROGRAMME

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DEBCOB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
*       SELECT DES FICHIERS
        SELECT FIMPR ASSIGN TO PRINTER.
        SELECT FSEQ ASSIGN TO NOM-EXTERNE
DATA DIVISION.
FILE SECTION.
*       DECLARATION DES FICHIERS.
FD FIMPR LABEL RECORD OMITTED.
01  LIGNE.
        02 FILLER PIC X.
        02 LIGNE PIC X (80).
FD FSECQ LABEL RECORD STANDARD.
01 SEQ.
        02 NUM PIC 99.
        02 NOM PIC X (25).
        02 SUITE PIC X (53).
*****************************************
WORKING-STORAGE SECTION.
* DECLARATIONS DES ZONES DE TRAVAIL
77  I  PIC  99  VALUE ZERO.
77  J  PIC  99 VALUE ZERO.
01  L1 PIC X(80) VALUE " ".
SCREEN SECTION.
```

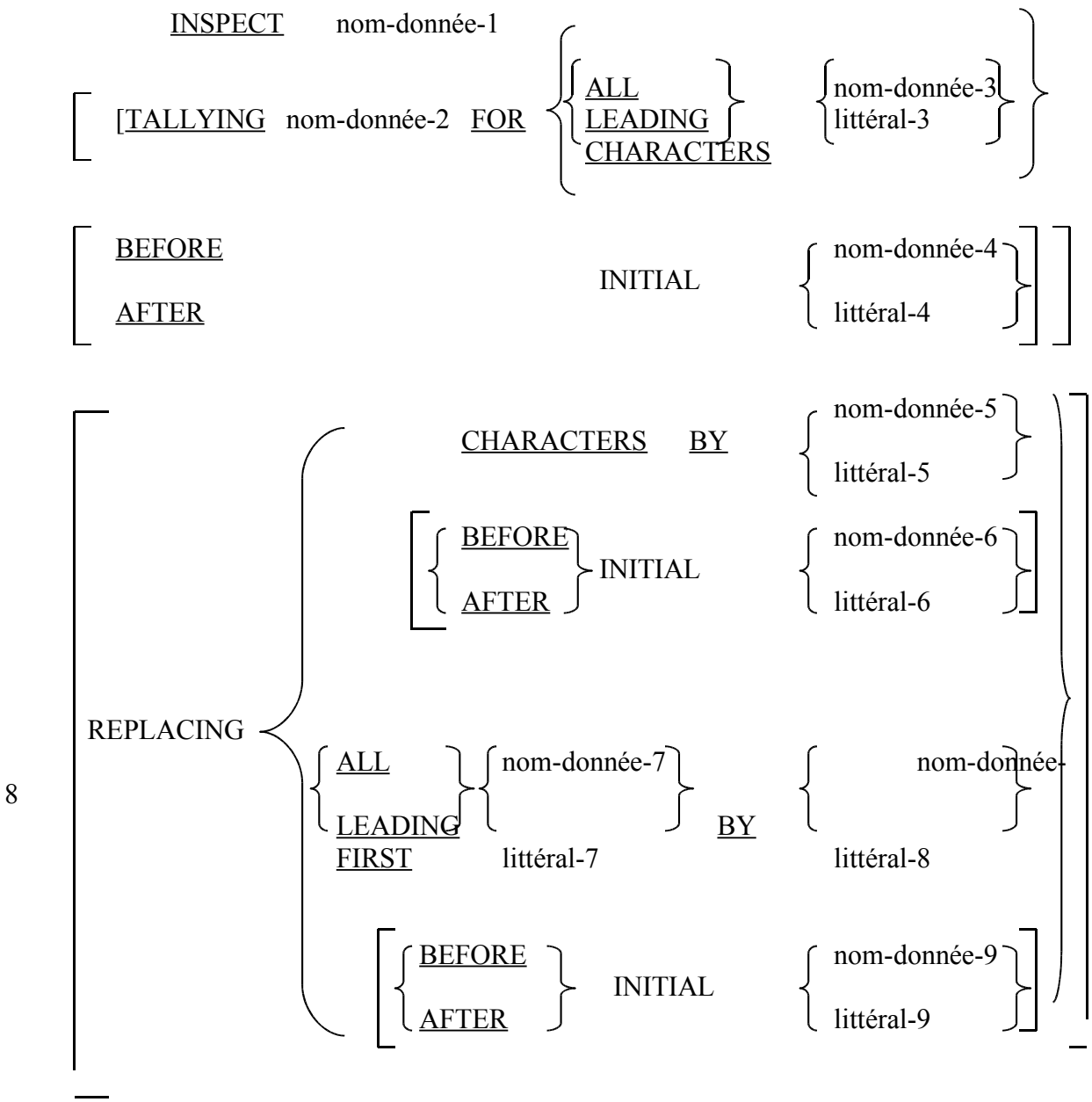
```
01      GRILLE
        02 BLANK SCREEN.
        02 LINE 2 COLUMN 3 VALUE "NUMERO : ?.
        02 LINE 2 COLUMN 15 PIC 99 TO NUM BELL AUTO.
        02 LINE 10 COLUMN 15 VALUE "NOM : ?".
        02 LINE 10 COLUMN 25 PIC X (25) TO NOM.

PROCEDURE DIVISION.
DEBUT
        OPEN OUTPUT FSEQ
        MOVE O TO NUM.
        MOVE SPACES TO NOM.
BOUCLE.
        DISPLAY GRILLE.
        ACCEPT GRILLE.
        MOVE SPACE TO SUITE.
        IF NUM = 99 GO TO FIN.
        WRITE SEQ INVALID DISPLAY "ERREUR" GO TO FIN.
        ADD 1 TO I.
        GO TO BOUCLE.
FBOUCLE.
FIN.
        CLOSE FSEQ.
        DISPLAY.
        STOP RUN.
```

CHAPITRE XXIV

L'INSTRUCTION INSPECT (ANCIEN EXAMINE).

L'instruction INSPECT permet de compter l'occurrence d'un ou plusieurs caractères DISPLAY dans une zône et éventuellement de le (ou les) remplacer par un autre (ou d'autres) caractères.

I – FORMAT GENERAL



Règles

1) Nom-donnée-1, zone à inspecter, doit être en USAGE DISPLAY

2) TALLYING signifie que le résultat du comptage sera placé dans Nom-donnée-2

3) ALL signifie que toutes les occurrences doivent être comptées

Attention

ZONA → | *****/*** |

INSPECT ZONA TALLYING CPT FOR ALL "**"
Formera 3 dans CPT

4) LEADING signifie que seules les premières occurrences contiguës seront comptées.

Ex INSPECT ZONB TALLYING COMPTEUR FOR LEADING "O"

ZONB | O | O | B | D | C | 1 | 0 | 0 | 5 | COMPTEUR = 2

5) CHARACTERS compte tous les caractères, éventuellement avant ou après un caractère ou un groupe de caractères prévus :

Ex INSPECT ZOND TALLYING CPT FOR CHARACTERS BEFORE INITIAL "1"

ZOND → | O | O | B | C | D | 1 | 0 | 0 | CPT = 5

6) REPLACING permet de remplacer les occurrences de configurations de caractères particuliers par d'autres.

Ex INSPECT ZONE REPLACING CHARACTERS BY "O"

ZONE | A | B | C | D | E | F | G | H | devient | OOOOOOOO |

7) ALL, LEADING ont la même signification qu'avec TALLYING

8) FIRST signifie que seule la première occurrence de la configuration désignée par nom-donnée-7 ou litt-7 sera remplacée par nom-donnée-8 ou litt-8 INSPECT ZONG REPLACING FIRST "T" BY "L"

Si ZONG → TOTO on obtient LOTO

9) Si on mélange TALLYING et REPLACING :

INSPECT ZONH TALLYING COMPTEUR FOR CHARACTERS AFTER INITIAL "S" REPLACING ALL "P" BY "T".

Avec ZONH PASSEE on obtient

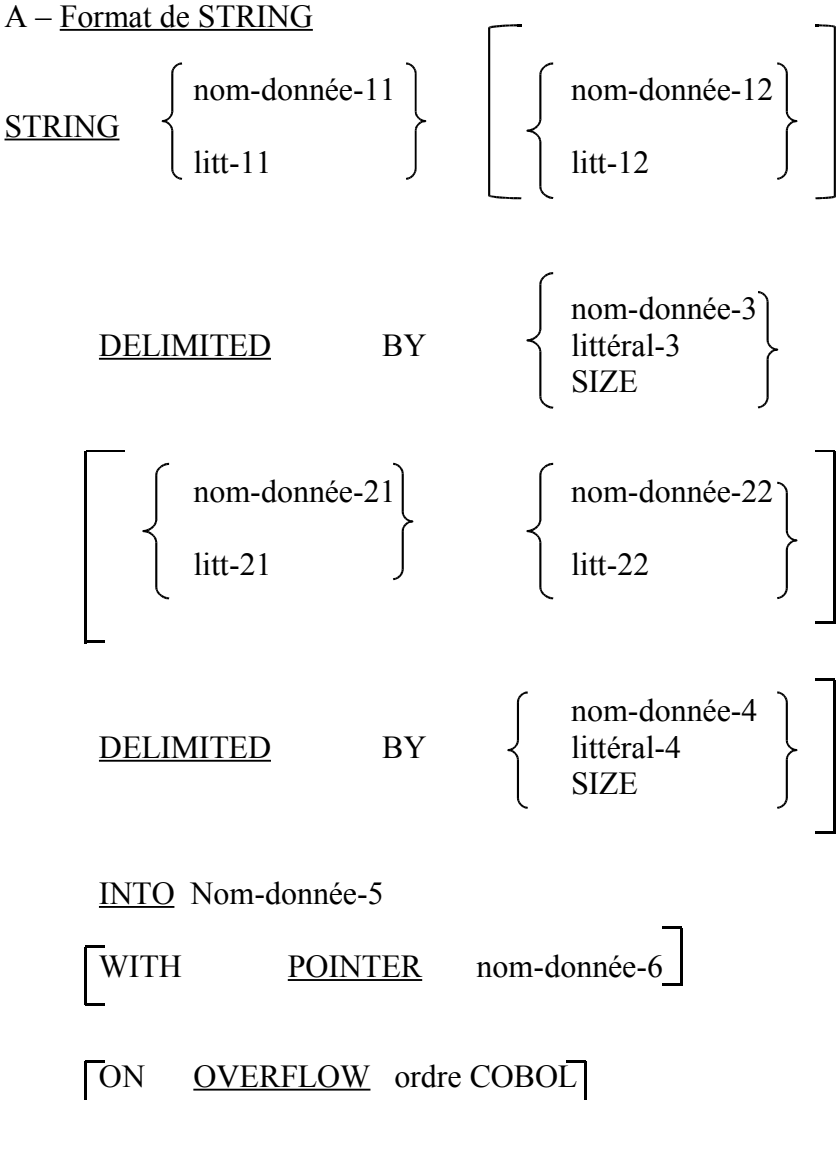
Compteur = 3 et

ZONH → TASSEE

II – MANIPULATIONS DE CHAINES DE CARACTERES

On dispose depuis la norme ANS 74 du verbe STRING qui permet de concaténer plusieurs zones de données dans une seule zone et du verbe UNSTRING qui permet d'éclater une zone de données en plusieurs zones élémentaires.

A – Format de STRING



On cherche à bâtir une chaîne (Nom-donnée-5) en concaténant les zônes de type nom-donnée-li et éventuellement nom-donnée-21. C'est au programmeur de vérifier que la zône réceptrice est au moins égale à la somme des longueurs des zônes émettrices. Il n'y a pas de remplissage à droite par des espaces.

1) DELIMITED BY permet de spécifier une limite de transfert de données.

2) POINTER permet de préciser la position gauche du transfert en zône réceptrice. Nom-donnée-6 est alors incrémenté de 1 pour chaque caractère transféré.

3) L'option OVERFLOW n'agit que conjointement à l'option POINTER dans le cas où le contenu de Nom-donnée-6 est inférieur à 1 ou supérieur au nombre de caractères de la zône réceptrice.

Ex

```
77 ZONE PIX X(15) VALUE SPACES.
   STRING "ANNEE" SPACE "1974" DELIMITED
                BY  SIZE  INTO  ZONE
        résultat  ZONE        ANNEE 1974
```

B – Format de UNSTRING

```
UNSTRING  nom-donnée-1

[  DELIMITED        BY     [ALL]    { nom-donnée-12 }
                                    { littéral-12   }

[ OR  [ALL]    { nom-donnée-13 }  ]  ...
               { littéral-13   }  ]

[ WITH        POINTER          nom-donnée-2  ]

[ TALLYING          IN         nom-donnée-3  ]

[ ON          OVERFLOW ]     ordre-cobol

INTO  nom-donnée-4  [         DELIMITER        IN        nom-donnée-41 ]

[ COUNT             IN         nom-donnée-42 ]
```

Les règles de transfert sont identiques à celles du verbe STRING.

Npm-donnée-1 est la zone émettrice à éclater. La règle d'éclatement est fournie par DELIMITED BY Nom-donnée-2, zone numérique élémentaire, permet de compter le nombre de zones réceptrices créées (il faut initialiser cette zone).

L'option OVERFLOW arrête le déroulement de UNSTRING dans deux cas :

- lorsque le contenu du pointeur Nom-donnée-2 est négatif ou supérieur à la taille de la zone émettrice Nom-donnée-1.

- lorsque toute la zone émettrice n'a pas été examinée et qu'il n'y a plus de zones réceptrices disponibles.

Ex :

```
77 ZONE PIC X (12)
77 NOM PIC X (5).
77 An      PIC X(4).
```

UNSTRING ZONE DELIMITED BY SPACE INTO NOM AN.

Si zone contient | ANNEE 1974 |

Alors NOM contient | ANNEE |

Et An contient | 1974 |

CHAPITRE XXV

LE TRI INTEGRE

COBOL – LE VERBE SORT

COBOL permet d'inclure des programmes généralisés de tri dans un programme COBOL. L'intérêt par rapport au tri utilitaire est essentiellement de pouvoir modifier ou éliminer des enregistrements avant ou après le tri proprement dit.

Le verbe SORT permet d'indiquer que l'on désire trier un fichier (FN1) mais il impose de décrire un fichier entrée (FN2) avant tri et un fichier de sortie (FN3) après tri.

1er Format

SORT Nom-de fichier-tri décrit en SD FN1 (*) USING FN2 GIVING FN3.

2ème Format

SORT FN1 (*) INPUT PROCEDURE nom-de-section GIVING FN3.

3ème Format

SORT FN1 (*) USING FN2 OUTPUT PROCEDURE nom-de-section.

4ème Format

SORT FN1 (*) INPUT PROCEDURE nom-de-section.
OUTPUT PROCEDURE nom-de-section.

(*) ON { ASCENDING / DESCENDING } KEY Arg1 [ON { ASCENDING / DESCENDING } KEY Arg2]

Les quatre formats correspondent à :

1) un tri sans modification des enregistrements en entrée et sortie.

2) un tri avec modification des enregistrements en entrée (INPUT PROCEDURE)

3) un tri avec modification des enregistrements en sortie (OUTPUT PROCEDURE)

4) un tri avec modification des enregistrements en entrée et en sortie

Règles :

1) FN1 a une description spéciale en DATA DIVISION commençant par SD (sort description) au lieu de FD.

2) FN2, FN3 font l'objet de description de fichiers (enregistrements de même longueur).

3) Nom-de-section représente le nom des sections permettant des séquences

- soit avant le TRI

- soit après le TRI

4) Arg1, Arg2 sont des sous-descriptions de l'enregistrement de FN1

5) Le verbe SORT peut apparaître en Procédure Division n'importe où sauf dans une INPUT ou OUTPUT Procédure.

Examinons les divers cas un à un :

A – Cas le plus simple où l'on trie TOUS les articles du fichier :

```
SORT FN1 … USING  FN2  GIVING  FN3.
```

Dans ce cas on n'ouvre pas les fichiers en entrée et en sortie. Le tri intégré gère l'ensemble.

Ex :

```
SORT FN1 … INPUT PROCEDURE ENTREE
   |              THRU FENTREE GIVING FN3
   |
ENTREE SECTION.
        OPEN INPUT FN2.
TQ1
        READ FN2…FTQ1
          |
          |
        RELEASE  ENRFN1   [FROM  ENRFN2]
SAUT.GOTO TQ1.


FTQ1.
        CLOSE FN2
FENTREE SECTION.
        EXIT
```

C – Cas où l'on souhaite modifier ou supprimer des enregistrements en sortie

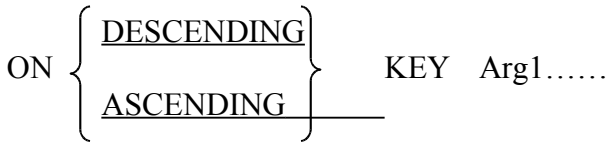
```
SORT FN1…USING FN2 OUTPUT PROCEDURE
                SORTIE THRU FSORTIE.


SORTIE SECTION.
        OPEN OUTPUT FN3.

TQ2
RETURN FN1 AT END GO TO FTQ2.


        WRITE ENR FN3.
    SAUT. GO TO TQ2.
FTQ2.CLOSE FN3.
FSORTIE SECTION.
          EXIT.
```

Il ne faut pas bloquer les enregistrements du SORT.

D – Cas où l'on imbrique B et C :

On utilise les deux sections nouvellement décrites simultanément !

On peut sélectionner plusieurs arguments de tri par la phrase



En général on ne dépasse pas trois niveaux :

- tri majeur

- tri intermédiaire

- tri mineur

Si l'on est amené à trier plusieurs fichiers dans un même programme il faut une description SD pour chaque fichier de tri.

CHAPITRE XXVI

LES SOUS-PROGRAMMES EXTERNES

(HORS - NORMES A.N.S.)

I – INTRODUCTION

Ils sont utilisés souvent pour des parties de programmes particulièrement complexes, ou qui écrites en assembleur, offrent des améliorations de performances.

Les sous-programmes externes sont des programmes écrits en COBOL ou tout autre langage compilé à part du programme principal.

Le programme principal et les sous-programmes sont ensuite reliés par le programme Editeur de lien (LINK EDIT).

En "Microsoft COBOL" les différents modules sont chargés au CALL (chargement dynamique).

II – APPEL D'UN SOUS-PROGRAMME

Cet appel est réalisé par l'instruction CALL

CALL "Nom-de-SP" [USING Nom-donnée-1 [N-d-2]..]

La longueur du nom-de-SP est fixée par la règle des noms dans le système d'exploitation (c'est-à-dire au maximum 6 caractères, le premier alphabétique pour l'IBM-PC).

Ex :

```
CALL "SP1" USING Z1  Z2.
```

La clause USING est utilisée pour transmettre au sous-programme l'adresse des zones de données du programme principal qui servent dans le sous-programme appelé.

III – LE SOUS-PROGRAMME COBOL

Le sous-programme COBOL est construit comme tout programme COBOL avec les quatre divisions usuelles.

Notons les quelques différences suivantes :

1) La PROCEDURE DIVISION est suivie des mots USING Z1 Z2…

2) La LINKAGE SECTION qui établit la correspondance avec les zones de données du programme principal désignées dans la clause USING de l'ordre CALL (description de Z1, Z2…).

3) Il existe de nouveaux mots COBOL redonnant le contrôle au programme principal tels GOBACK ou EXIT PROGRAM (celui-là doit être unique, il correspond au STOP RUN du programme principal).

Attention : dans le cas où le S-P. est utilisé seul, ces 2 mots changent de rôle :

- GOBACK → équivaut à un STOP RUN

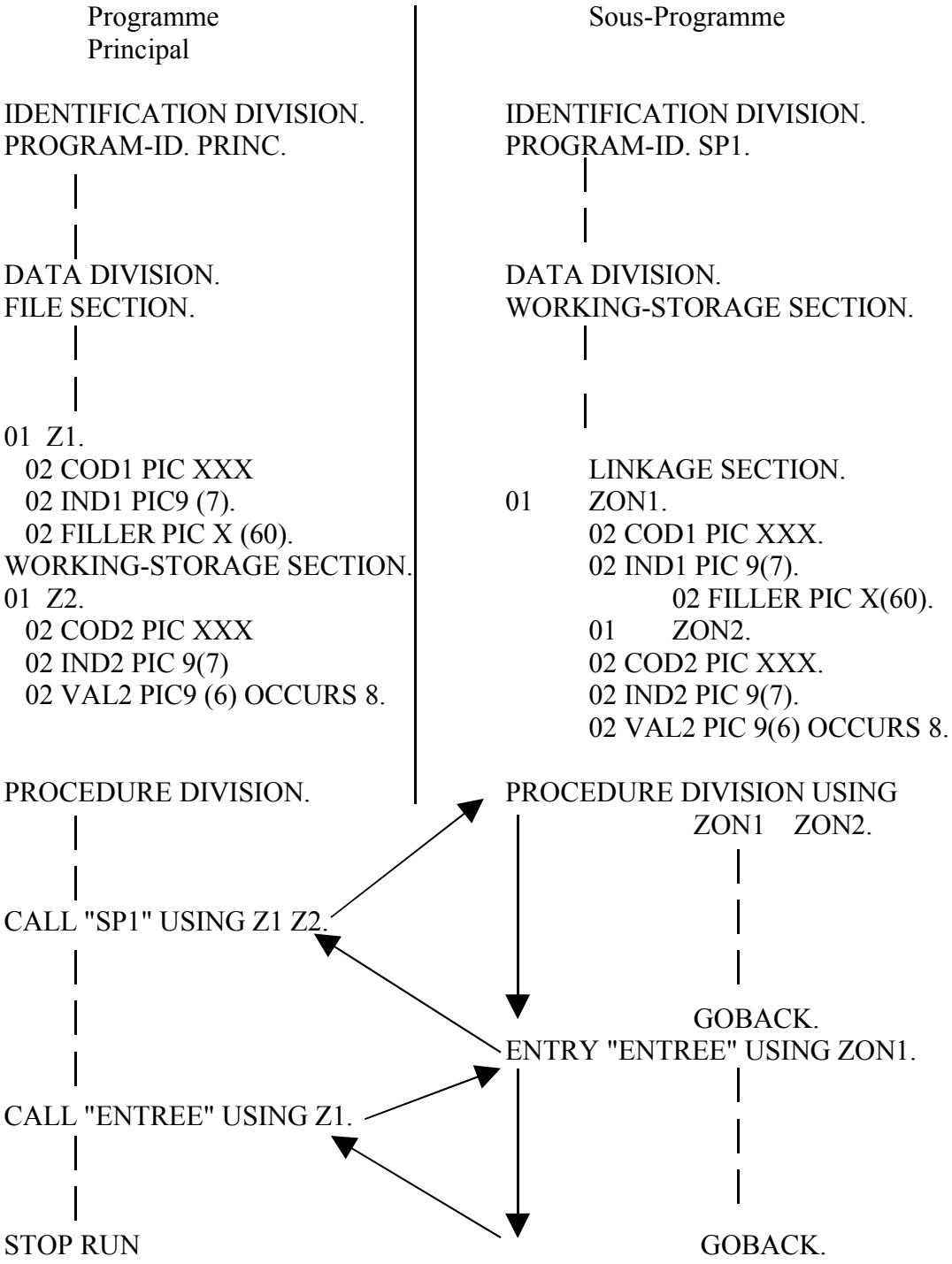
- EXIT PROGRAM est ignoré (=EXIT habituel)

4) Il existe aussi des "points d'entrée" ENTRY ; la syntaxe est proche de "PROCEDURE DIVISION USING Z1 Z2…" : ENTRY "nom-symbolique-entrée" USING Y1 Y2…

Le "nom-symbolique-entrée" joue le même rôle que pour le CALL, mais permet d'éviter certaines parties du sous-programme.

5) La LINKAGE SECTION a comme particularité de ne pas réserver de place en mémoire centrale mais d'attribuer aux zones une adresse VIRTUELLE le, CALL attribue des adresses réelles aux zones de données de la LINKAGE SECTION.

IV – Exemple détaillé :



Les flèches en pointillé indiquent les parties du sous-programme concernées par les 2 CALL.

La syntaxe peut-être légèrement différente d'une machine à l'autre (voir brochure du constructeur).

V – Le chaînage de programme

Le COBOL IBM-PC permet l'appel (ou "chaînage") d'un autre programme principal.

La différence avec l'appel de sous-programme réside dans le fait que le programme qui lance le chaînage est DETRUIT, donc on ne peut y revenir après exécution du programme chaîne.

Format

CHAIN { Littéral / Nom-donnée-1 } [USING n-d-1 [n-d-2]…]

littéral ou Nom-donnée-1 désignent le nom d'un programme (suivant les normes du système d'exploitation de l'IBM-PC).

Exemple

```
CHAIN NOMPROG USING TABLEAU1 TABLEAU2.
```

TABLEAU1, TABLEAU2 désignent des données décrites en DATA DIVISION qui seront transmises au programme chaîne NOMPROG.

CHAPITRE XXVII

AIDE A LA MISE AU POINT DES PROGRAMMES COBOL

(HORS NORMES A.N.S.)

I – LE VERBE COPY

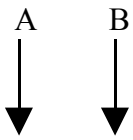
Ce verbe est utilisé pour faire recopier directement par le compilateur une partie de programme de l'environnement division, Data ou Procédure division.

Cette partie a été au préalable enregistrée sous un nom de fichier COBOL sur disquette.

Format : COPY nom-de-fichier

Exemple :

Supposons que l'on ait créé sous le nom ANNUAIRE.COB




```
02  NOM PIC X(9).
02  NUMERO PIC 9(6).
02  SERVICE PIC 9(4).
```

On pourra écrire dans un autre programme :

```
01  REPERTOIRE.
    COPY  ANNUAIRE.
```

Ce qui donnera à la génération :

```
01  REPERTOIRE
    02 NOM PIC X(9).
    02 NUMERO PIC 9(6).
    02 SERVICE PIC 9(4).
```

Certains compilateurs acceptent une instruction COPY plus sophistiquée avec une clause de remplacement (REPLACING n-d Byn-d-2).

II – LE VERBE TRACE

Ce verbe permet d'obtenir la liste de tous les noms de paragraphes activés lors du déroulement du programme :

READY TRACE. (autres matériels : TRACE ON)

(Si l'on veut arrêter la trace)

RESET TRACE (autres matériels : TRACE OFF)

III – LE VERBE EXHIBIT

Ce verbe permet d'éditer la valeur contenue dans une zone donnée

EXHIBIT [NAMED] n-d-1 [n-d-2]…

Les valeurs sont imprimées en caractères DISPLAY

- avec impression du nom de la donnée si on utilise l'option NAMED.

Certains compilateurs acceptent une deuxième option : CHANGED. Dans ce cas la zone n'est affichée que si sa valeur est changée.

CHAPITRE XXVIII

LES DECLARATIVES

I – INTRODUCTION

COBOL a prévu l'éventualité de traitements particuliers de sauvegarde en cas d'erreur d'entrées/sorties (données illisibles sur un fichier par exemple).

Ceux-ci sont réalisés en séquences indépendantes rassemblées dans une partie bien spéciale de la PROCEDURE DIVISION : Les DECLARATIVES.

Cette partie (toujours placée en tête de la PROCEDURE DIVISION) peut être importante au cas où l'on désire décider soi même de ce qu'il faut faire en cas d'incident.

II – FORMAT GENERAL D'ECRITURE DES DECLARATIVES


```
PROCEDURE DIVISION.
DECLARATIVES.

Nom-de-Section     SECTION.  USE….
Nom-paragraphe.
    |
    |    Instruction COBOL
    |
    |

END DECLARATIVES.
```

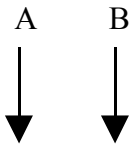
Chaque section assure une tâche particulière vis-à-vis des entrées/sorties et le sujet traité par la section est indiqué par la phrase USE.

Exemple :

```
PROCEDURE DIVISION.
DECLARATIVES.
ERR-LECTURE SECTION.USE AFTER ERROR PROCEDURE ON INPUT
TRAITER-LECT.
      |
      |

END DECLARATIVES.
```

survient une erreur de lecture de fichier, le superviseur du système d'exploitation de l'ordinateur branchera aussitôt le traitement sur la section ERR-LECTURE pour assurer les procédures exceptionnelles prévues par le programmeur.

III – FORMAT DE SECTIONS DECLARATIVES POUR INCIDENTS D'ENTREE/SORTIES



PROCEDURE DIVISION.
DECLARATIVES.

Nom-de-section SECTION.USE AFTER STANDARD ERROR

PROCEDURE ON { nom-de-fichier-1 nom-fichier-2…. | INPUT | OUTPUT | I-O }

- Nom-fichier-1, Nom-fichier-2…. Indiquent que cette section déclarative ne s'applique qu'aux seules erreurs détectées sur ces fichiers.

- INPUT : la section s'applique à tous les fichiers en lecture du programme

- OUTPUT : la section s'applique à tous les fichiers en écriture

- I-O la section s'applique à tous les fichiers en lecture/écriture

Comme l'indique l'instruction USE, le branchement à la section déclarative n'a lieu qu'après que les routines standard de traitement d'erreurs du système d'exploitation de l'ordinateur aient été effectués.

On sort d'une section déclarative automatiquement après la dernière instruction de la section.

COBOL n'autorise pas, dans une section déclarative, à faire référence à des paragraphes extérieurs à cette section.

Réciproquement, les sections déclaratives étant considérées comme des séquences indépendantes, on ne pourra y faire référence depuis la PROCEDURE DIVISION normale d'un programme que par des instructions PERFORM.

CHAPITRE XXIX – UTILISATION D'UN FICHIER

SEQUENTIEL INDEXE ET D'INDEX SECONDAIRES.

I – INTRODUCTION

Le but d'une telle utilisation est de pouvoir accéder à un enregistrement par plusieurs informations différentes de cet enregistrement. D'ordinaire en séquentiel indexé, on doit connaître la valeur de la RECORD KEY pour retrouver un enregistrement.

Il est très possible d'obtenir ce fonctionnement aux moyens de clefs secondaires (ALTERNATE).

II – MODIFICATIONS DANS LE PROGRAMME DE COBOL

La phrase SELECT doit préciser au système la liste des clefs supplémentaires, en informant de l'autorisation de présence de DOUBLONS par l'attribut DUPLICATES.

Format de cette clause :

ALTERNATE RECORD KEY IS

data-name-1 with DUPLICATES
split-key-name-A = d-n-2 d-n-3
WITH DUPLICATES

Il reste que pour utiliser les ordres DELETE, REWRITE ou WRITE il faut utiliser la RECORD KEY habituelle.

Les clefs secondaires peuvent être utilisées uniquement pour RETROUVER un enregistrement.

Si l'on utilise la clause WITH DUPLICATES on accepte les doublons dans le fichier.

Une clef secondaire peut être constituée avec plusieurs champs, même non consécutifs, de l'enregistrement ("splitté").

En procédure division il faut préciser avec quelle clef on travaille :

charger data-name-1
READ fichier KEY IS data-name-1 INVALID

KEY----------

Les clefs secondaires apparaissent donc comme une possibilité supplémentaire d'accéder à un enregistrement et préfigure la notion de base de données.

Exemple :

```
SELECT FICHIER ASSIGN TO DISK ORGANIZATION
IS INDEXED ACCESS MODE IS DYNAMIC
RECORD KEY IS CLEF-PRINC
ALTERNATE RECORD KEY IS CLEF2
ALTERNATE RECORD KEY IS CLEF3 WITH DUPLICATES
ALTERNATE RECORD KEY IS ECLATE = CLEF-PRINC
        PROJET BONUS.
```

Associé à une description

```
FD  FICHIER LABEL RECORD STANDARD
        VALUE OF FILE-ID IS "FICHIER.EXE".
01  ENREG.
    02  CLEF-PRINC         PIC 9(5).
    02  CLEF2              PIC X(20).
    02          CLEF3               PIC X(20).
    02  PROJET             PIC X(12).
    02  BONUS              PIC S9(4)V99.
```

Pour cette description, les enregistrements peuvent être retrouvés par la RECORD KEY (CLEF-PRINC), par deux clefs secondaires (CLEF2, CLEF3), et une clef secondaire "splitée" (ECLATE).

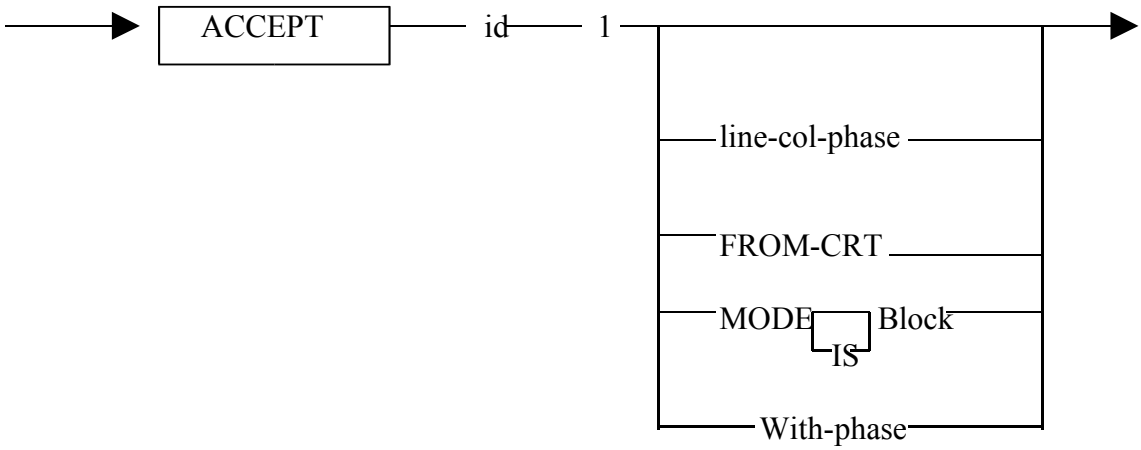
CHAPITRE XXX : ACCEPT et DISPLAY "POSITIONNES"

INTRODUCTION

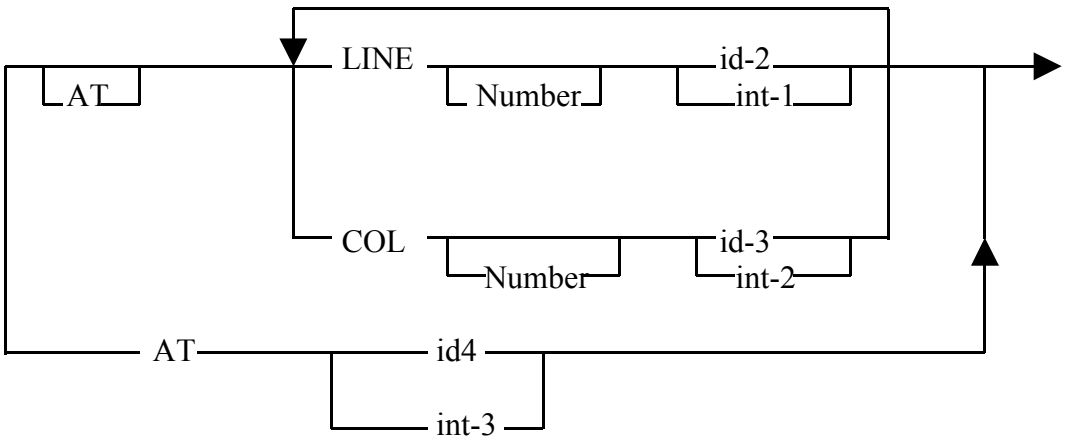
Ces deux VERBES, habituellement réservés à un strict usage d'entrées/sorties de petites dimensions, se voient agrémentés d'extensions permettant d'envisager une utilisation plus fréquente.

On retrouve en gros, tous les attributs, vidéo ou de positionnement, qui existent dans le COBOL Microsoft à l'usage des compatibles P.C.

I – ACCEPT "étendu".



- line-col-phase s'entend par :



- with-phase permet d'utiliser tous les attributs décrits dans un autre chapitre (XXIII, page).

Comme dans SDA il faut toujours laisser une position après et avant un champ utilisé par ACCEPT (responsabilité de programmeur)

On doit d'abord par l'instruction DISPLAY SPACES UPON CRT WITH BLANK SCREEN, effacer l'écran.

L'option "AT…" indique l'adresse absolue dans l'écran où commence l'instruction ACCEPT. S'il n'y a rien l'opération commence Ligne 1, colonne 2.

Si line = 0 et col = 0, l'ACCEPT démarre à la position suivant le précédent ACCEPT.

Si la ligne est absente (ou nulle) et la colonne présente, l'ACCEPT se présente à la ligne suivant le dernier DISPLAY à la colonne indiquée (idem pour ligne présente et colonne absente).

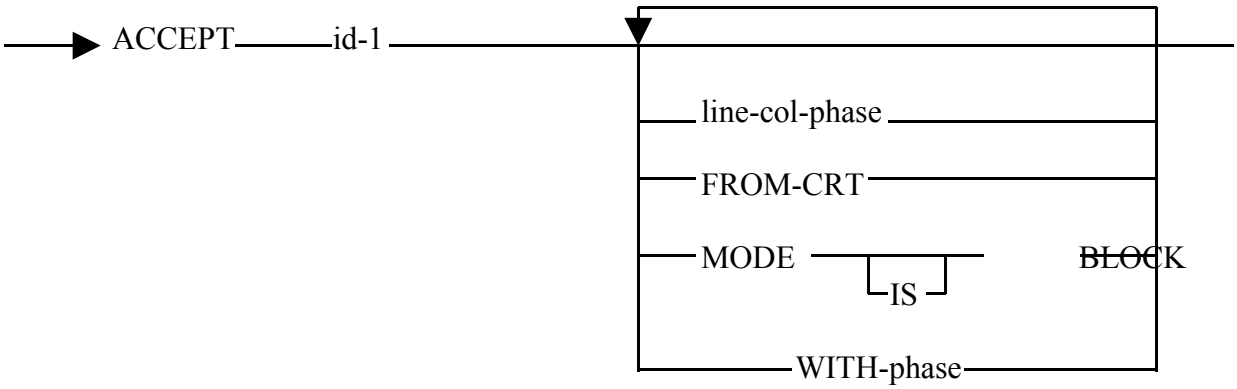
On peut aussi écrire ACCEPT X AT 1214 (ligne = 12, colonne = 14)

- MODE BLOCK indique que la donnée est à traiter comme une donnée élémentaire.

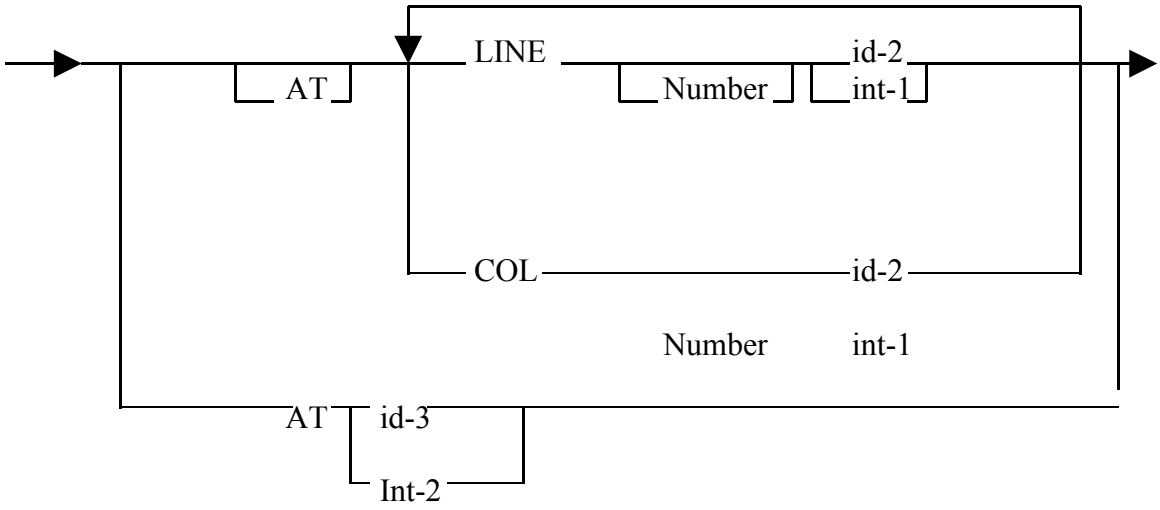
- WITH phase permet à l'utilisateur de spécifier certaines options accompagnant l'opération ACCEPT.

II – DISPLAY "étendu"

- On retrouve strictement la même syntaxe que pour le "ACCEPT étendu" avec un peu moins de possibilités dans la with phase (BEEP, BLINK, HIGHLIGHT, RESERVE-VIDEO, UNDERLINE, BLANK SCREEN, BLANK LINE).



line-col-phase est :



id-3 PIC 9(4) ou PIC 9(6)
int-2 champ numérique à 4 ou 6 octets
On partage dans les 2 cas en 2 parties égales pour ligne et colonne
ACCEPT X AT 1218
Ligne = 12, colonne = 18

ANNEXE 1

Le PERFORM "classique" du COBOL introduit des débranchements peu en rapport avec l'idée actuelle de "programmation structurée". Le standard ANS85 a introduit le PERFORM "en ligne" et les terminateurs d'instructions. Ces terminateurs permettent d'améliorer la lisibilité du programme et surtout sa structuration.

Les principaux terminateurs d'instructions reconnus par le standard ANS sous END-ADD, END-SUBTRACT, END-MULTIPLY, END-DIVIDE, END-IF, END-PERFORM, END-READ, END-WRITE.

Utilisation du END-IF

```
IF          A > B
            MOVE C TO D
ELSE
            MOVE D TO C
            ADD 1 TO I
END-IF.
```

C'est tout de même plus clair qu'avec le seul point terminal !

Utilisation du END-PERFORM

C'est à mon avis, la modification la plus importante, car elle traduit directement le souci de structuration logique des séquences d'un programme :

```
A     B
↓     ↓
DEBUT.
      PERFORM    UNTIL    A >  B
                 READ FICHIER AT END
                              MOVE "V" TO EOF
                              END-READ
                 ADD 1 TO A
                 WRITE LIGNE FROM ENR END-WRITE
      END-PERFORM.
```

On peut, sans difficultés imbriquer d'autres instructions IF, PERFORM en respectant l'usage indiqué ci-dessus.

ANNEXE II

La notion de "SECTION" a été vue dans les deux divisions ENVIRONMENT ET DATA.

Son rôle en PROCEDURE DIVISION permet d'accentuer la possibilité de structurer davantage la programmation COBOL.

L'exemple fourni dans les pages suivantes montre :

1) qu'on peut décomposer (comme en algorithmique) en modules (principal et secondaires) en rajoutant derrière un nom de paragraphe le mot clé SECTION (repérer ici PRINCIPALE, TRAITERDATE, ANNEBISSEX, AFFICHER).

On se rapproche de la notion de PROCEDURE, avec une forte restriction :

- les variables restent GLOBALES.

Les noms de paragraphe utilisés dans une SECTION peuvent être réutilisés dans une autre.

Il est alors interdit de sortir de la section en utilisant des noms de paragraphes appartenant à une autre SECTION.

Il est fortement conseillé de terminer chaque SECTION pour un paragraphe FIN suivi de son instruction EXIT.

La portée d'une SECTION s'applique à tous les paragraphes contenus dans la SECTION jusqu'à

- une prochaine SECTION
- la fin du programme

PROGRAMME

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DATAAAAJJJ.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
01      AAA             JJJ.
        05 AAAA PIC 9(4)
        05 JJJ PIC 999.
01      LISTEMOIS.
        05 FILLER PIC X(18) VALUE "JANVIER FEVRIER".
        05 FILLER PIC X(18) VALUE "MARS     AVRIL".
        05 FILLER PIC X(18) VALUE "MAI      JUIN".
        05 FILLER PIC X(18) VALUE "JUILLET  AOUT".
        05 FILLER PIC X(18) VALUE "SEPTEMBRE OCTOBRE".
        05 FILLER PIC X(18) VALUE "NOVEMBRE DECEMBRE".
01      BIDON REDEFINES LISTEMOIS.
        05 NOMMOIS PIX X(9) OCCURS 12.
01      LISTEJOURS.
        05 FILLER PIC X(24) VALUE "312831303130313130313031303.
01      BIDON2 REDEFINES LISTEJOURS.
        05 NBJOURS PIC 99 OCCURS 12.
77      MOIS PIC 99.
77      BISSEXTILE PIC X.
77      VRAI PIC X VALUE "1".
77       FAUX PIC X VALUE "0".
77       R PIC 999.
77       BIDONS PIC 9999.
77       EDJJ PIC ZZ9.
PROCEDURE DIVISION.
PRINCIPALE SECTION.
DEBUT.
        ACCEPT AAAJJJ.
```

```
                PERFORM TRAITERDATE.
                PERFORM AFFICHER.
FIN.
                STOP RUN.
TRAITERDATE SECTION.
DEBUT.
                PERFORM ANNEEBISSEX.
                IF BISSEXTILE = VRAI MOVE 29 TO NBJOURS (2) END-IF.
                MOVE 1 TO MOIS.
TQ.
                IF NBJOURS (MOIS)NOT < JJJ GO TO FTQ-END-IF.
                SUBTRACT NBJOURS (MOIS) FROM JJJ.
                ADD 1 TO MOIS.
                GO TO TQ
FTQ.
FIN.
                EXIT.
ANNEEBISSEX SECTION.
DEBUT.
                MOVE FAUX TO BISSEXTILE.
                DIVIDE 4 INTO AAAA GIVING BIDON3 REMAINDER R.
                IF R = O
                    DIVIDE 100 INTO AAAA GIVING BIDON3 REMAINDER R
                    IFR = 0
                            DIVIDE 400 INTO AAAA GIVING BIDON3 REMAINDER R
                            IFR = 0 MOVE VRAI TO BISSEXTILE END-IF
                    ELSE MOVE VRAI TO BESSEXTILE
                    END-IF
                END-IF.
FIN.
                EXIT.
AFFICHER SECTION.
                MOVE JJJ TO EDJJ.
                DISPLAY EDJJ " " NOMMOIS(MOIS) " " AAAA.
FIN.
                EXIT.
```