A V E R T I S S E M E N T

Le présent document ne doit pas être considéré comme un manuel de référence du COBOL mais comme un support du cours COBOL.

Certaines parties dans le codage des instructions ont été volontairement omises:

- par souci de clarté de l'exposé
- pour éviter les outils incitant à la rédaction de programmes mal structurés.
- parce qu'elles sont obsolètes
- parce qu'elles sont pas (ou peu) utilisées dans la pratique

Référence: Langage de programmation COBOL normalisé norme française NF Z 65-210
Ce support décrit la syntaxe conforme à ANS85 (X3.23-1985 et X3.23a-1989)

GENERALITES DU LANGAGE COBOL

1. Historique

En 1959, des représentants des secteurs public et privé se réunissent au Pentagone pour étudier les possibilités d'élaborer un langage de programmation adapté à la résolution des problèmes de gestion de l'Armée; ce langage fut appelé COBOL (Common Business Oriented Language : Langage commun orienté vers la gestion). Sa première version fut opérationnelle en décembre 1959.

Issue de ce groupe de travail, la CODASYL (Conférence On Data System Language) produisit une seconde version en 1961.

Depuis, d'autres organisations, nationales ou internationales, ont réalisé de nouvelles versions, en particulier l'ANSI (American National Standard Institute) et l'ISO (International Standard Institute). Plusieurs versions de langage ont finalement été normalisées: COBOL-68, COBOL-74 (la plus utilisée), COBOL-85 (revue en 89). Malgré cela, il subsiste des différences entre les compilateurs proposés par les constructeurs (syntaxe, extensions propres, intégration dans le système), qui définissent des **dialectes** spécifiques.
COBOL continue son évolution et les dernières versions intègrent les concepts de la programmation orientée objet.

2. Intérêt du COBOL

2.1. Indépendance de la machine utilisée

Les programmes peuvent être facilement transposés d'une machine à une autre.

La majorité des ordinateurs disposent d'un compilateur COBOL, quels que soient leur puissance et le système d'exploitation qu'ils utilisent.

Ceci facilite les changements de configuration; toutefois l'indépendance n'est réelle que si l'on respecte la norme, et donc si l'on s'abstient d'utiliser les extensions propres au constructeur.

2.2. Un langage adapté à la résolution des problèmes de gestion

Ces problèmes se caractérisent par des **volumes importants** de données à traiter; le COBOL intègre la majorité des fonctions de manipulation des fichiers traditionnels

2.3. Apprentissage facile

Les phrases de COBOL sont proches du langage courant anglais ce qui facilite la lecture du programme mais entraîne en contrepartie certaines "longueurs".

2.4 Interfaçage avec d'autres outils

De nombreux compilateurs proposent des extensions permettant l'utilisation, depuis un programme COBOL :
- de code développé à partir d'un autre langage,
- de langage d'accès aux bases de données
- d'outils de gestion de transactions

3. Fonctions du COBOL

Depuis 1968, les spécifications de COBOL sont basées sur le concept de MODULES de traitement. La norme définit un noyau et onze modules fonctionnels. Chaque module contient deux niveaux de spécification; les modules M3 à M11 peuvent être vides (niveau 0). Cette méthode permet de comparer les compilateurs offerts par les constructeurs, du plus restreint (niveau 1 du noyau et des modules M1 et M2) au plus complet (le noyau et tous les modules au niveau 2).

Niveaux

1,2 Noyau
- définition du jeu des caractères autorisés
- éléments du langage nécessaires au traitement interne

1,2 M1 Traitement des tables: définition et accès à des tables au moyen d'indices et d'index

1,2 M2 Organisation séquentielle: gestion des fichiers d'organisation séquentielle en accès séquentiel

0,2 M3 Organisation séquentielle indéxée: gestion des fichiers d'organisation séquentielle indexée

0,2 M4 : Organisation relative: gestion des fichiers d'organisation relative

0,2 M5 Tri et fusion: inclusion de tri ou de fusion de fichiers dans un programme

0,1 M6 Edition: aide à la production d'états imprimés

0,2 M7 Segmentation : procédures de recouvrement de sections de la division des traitements

0,2 M8 Bibliothèque : inclusion de textes dans un programme source

0,2 M9 Mise au point : aide à la mise au point et contrôle de l'exécution d'un programme

0,2 M10 Communication inter-programmes :
- sous-programmes
- partage des données entre programmes

0,2 M11 Communications : émission et réception de messages

0,1 M12 Fonctions intrinsèques : calcul, caractères, date

4. Eléments du langage

Tous les éléments surlignés sont spécifiques à la norme ANS85

4.1. Jeu des caractères autorisés

Il comporte les **caractères** suivants:
- 10 chiffres: 0 1 2 9
- 26 lettres: A B C ..... Z a b c ……. z
- le symbole monétaire: $
- 4 signes arithmétiques: + - * /
- 3 symboles de relation: > < =
- 7 séparateurs utilisés pour la ponctuation: , ; . ~~b~~ ( ) " :

(Nous notons ~~b~~ le caractère "espace").

Tous ces caractères sont représentés en mémoire par leur code (ASCII ou EBCDIC); le code ASCII natif utilise 7 chiffres binaires (bits) et est complété par un 8ème bit égal à zéro afin de stocker à l'octet et de pouvoir utiliser une représentation hexadécimale.
Sauf dans des littéraux non-numériques, les lettres minuscules sont équivalentes aux majuscules.

4.2. Les mots

Un mot COBOL est une chaîne continue d'au plus **30** caractères pris dans le sous-ensemble { lettres, chiffres, - }. Un tiret (-) ne peut être ni le premier, ni le dernier caractère d'un mot.

On distingue :
- les mots-utilisateurs
- les mots-réservés
- les noms-système

4.2.1. Les mots-utilisateurs

Créés par le programmeur, ce sont des noms servant à référencer des structures de données ou de traitements. On distingue :

- les **noms de données** : ils servent à désigner une zone de mémoire contenant une information :
- les noms de fichiers : permettent de référencer une structure de fichier
- le nom de programme : identifie le programme
- les **noms de paragraphes et de sections** : délimitent des sous-ensembles du programme
- les **noms d'article** : identifient une structure d'enregistrement logique dans un fichier.
- les **noms d'index** : désignent des index associés à une table
- les **numéros de niveaux** : indiquent la hiérarchie des rubriques à l'intérieur d'une structure complexe.

Tous ces mots propres au programmeur ont une **portée limitée au programme** qui les utilisent (et les programmes contenus en cas de globalisation).

Les numéros de niveaux sont uniquement composés de chiffres et appartiennent à l'ensemble {**01** à **49**, **66**, **77**, **88**}.

A l'exception des noms de paragraphes et de sections et des numéros de niveaux, les mots-utilisateurs doivent comporter **au moins une lettre**.

Exemples

Noms de données : `A1` `TOTAL-FACTURE` `COEFF3`

Nom de paragraphes ou de sections : `S1` `CALCUL` `71` `7-1`

Remarques :

- Les mots cités en exemples de noms de données pourraient aussi être des noms de fichiers ou d'articles ou d'index.

- L'écriture `TOTAL-FACTURE` représente un seul mot à cause du tiret ; TOTAL FACTURE est considéré comme une suite de deux mots.

- Le nom de section `1` est différent du nom `01`

A cette liste, il convient d'ajouter les **constantes (ou littéraux)**

- numériques

- non numérique ou chaînes de caractères

- Les **littéraux numériques** sont limités à 18 chiffres et peuvent contenir un signe (à l'extrême gauche) et une marque décimale (.).

Exemple : `125` `-42.1` `+21`

Certains dialectes acceptant les données en virgule flottante, un littéral numérique de ce type s'exprime par
[±] mantisse E [±] exposant , mantisse d'au plus 16 chiffres, et exposant sur 2 chiffres (domaine de valeurs de 0.54E-78 à 0.72E+76).

Exemples : 15.2E18 -458E-42

- Les **littéraux non numériques** sont des chaînes d'**au plus 160 caractères** quelconques encadrées de **guillemets** (ou apostrophes suivant dialecte).

Exemple: `"*** Liste des Fournisseurs ***"`

4.2.2. Les mots-réservés

Ces mots ont une signification précise pour le compilateur COBOL que le programmeur ne peut pas modifier. Il est donc prudent lorsqu'on choisit un mot utilisateur de s'assurer qu'il ne s'agit pas d'un mot réservé ; à cet effet consulter la liste en Annexe.

Ces mots ont par ailleurs une orthographe fixée qu'il faut impérativement respecter.

On distingue :

- **les mots-clés** : Ils identifient les clauses et les instructions ; la présence du mot clé identifiant une instruction est indispensable chaque fois que cette instruction est utilisée.

  Exemple : `ADD` identifie le verbe arithmétique de l'addition

- **les mots facultatifs** :Ils ne servent qu'à se rapprocher de la syntaxe de la langue anglaise ; leur absence ne modifie pas le sens de la clause ou de l'instruction.

  Exemple : `DATA RECORD IS ARTICLE1`
  peut s'écrire `RECORD ARTICLE1,` `ARTICLE1` étant un mot utilisateur

- **les conjonctions** : pour relier des opérandes dans les clauses ou instructions : virgule ou point-virgule (facultatifs) ; pour former des expressions conditionnelles composées : AND et OR

- **les opérateurs arithmétiques et de relation** : `+ - * / = > < **`

- **les constantes figuratives** : ces mots sont utilisés pour désigner des constantes particulières (`ZERO, SPACE, HIGH-VALUE, LOW-VALUE, QUOTE, ALL` lit) ; ils seront abordés ultérieurement.

- **les registres spéciaux** : zones de mémoire accueillant des valeurs particulières ; ils sont spécifiques des dialectes ; par exemple en OSVS :

`CURRENT-DATE`	date du jour (MM/DD/YY)
`RETURN-CODE`	code-retour émis par le pg
`TALLY`	compteur associé à `EXAMINE`
`TIME-OF-DAY`	heure-système
...	

4.2.3. Les noms-système

Ils servent à communiquer avec le système d'exploitation pour désigner une machine, un langage ou un dispositif (`PRINT`, `SWITCH-1, CONSOLE, TERMINAL, SYSIN, SYSOUT,` ...)

4.3. Instructions et clauses

4.3.1. Les instructions

Une instruction est une combinaison syntaxiquement correcte de mots et de symboles; elle figure dans la **division des traitements** et spécifie une action particulière. Elle commence toujours par un **verbe**, mot réservé qui l'identifie.

Exemple :

```
ADD 1 TO C                      Addition
MOVE RESULTAT TO EDITION        Transfert
```

4.3.2. Les clauses

Une clause est une combinaison syntaxiquement correcte de mots et de symboles ; elle permet de spécifier un **attribut** ou une **disposition** particulière. Contrairement aux instructions, les clauses ont un rôle descriptif statique.

Exemple : `USAGE IS DISPLAY` précise le mode de codification utilisé indépendamment des valeurs et des opérations effectuées.

`LABEL RECORD IS STANDARD` indique que pour identifier ce fichier, on utilise la procédure des étiquettes standards.

4.4. Les phrases

Une phrase est une **suite d'instructions ou de clauses terminée par un point**.

4.5. Les paragraphes

Un paragraphe est une suite d'une ou plusieurs phrases précédée par un **en-tête de paragraphe** qui l'identifie.
L'en-tête de paragraphe est un nom de paragraphe suivi d'un point.
Un paragraphe se termine à la rencontre d'un des éléments suivants :

- en-tête d'un nouveau paragraphe
- en-tête de section
- en-tête de division
- fin du programme

Exemple :

```
PAR0. ADD A TO B
      MOVE A TO I.
PAR1. MOVE B TO C.
S2 SECTION.
```


4.6. Les sections

Une section est un ensemble de zéro, un ou plusieurs paragraphes précédé par un **en-tête de section** au format suivant :

nom de section `SECTION`.

Une section se termine à la rencontre d'un des éléments suivants :
- en-tête d'une nouvelle section
- en-tête de division
- fin de programme

4.7. Les divisions

Une division est un ensemble de zéro, une ou plusieurs sections précédé par un **en-tête de division** au format suivant :

nom de division `DIVISION`.

Le nom de division est un mot clé.

Une division se termine à la rencontre d'un en-tête de division ou de la fin du programme.

5. Structure d'un programme

Un programme COBOL comporte **4 divisions obligatoires** (sauf ANS85), présentées dans l'ordre suivant :

5.1. IDENTIFICATION DIVISION

Elle permet de préciser des renseignements généraux sur le programme tels que : nom du programme, date d'écriture, nom de l'auteur ...

5.2. ENVIRONMENT DIVISION

Elle contient la description de l'ordinateur utilisé (`CONFIGURATION SECTION`) et des informations relatives à la gestion des entrées-sorties (`INPUT-OUTPUT SECTION`).

5.3. DATA DIVISION

Elle contient la description des informations ou données que le programme reçoit, traite et produit ; ces données se répartissent en trois sections :

- La `FILE SECTION` contient toutes les informations relatives aux structures de fichiers (**données externes**).

- La WORKING-STORAGE SECTION rassemble toutes les **données internes** au programme (zones de travail, de calcul, tables, etc ...)
- La LINKAGE SECTION contient les **données communes** à plusieurs programmes.
- La COMMUNICATION SECTION décrit l'interface entre le programme et le sous-système de gestion des messages (terminaux locaux et distants).
- La REPORT SECTION permet la spécification et la description des états.

On trouve aussi, dans les dialectes, les SCREEN SECTION et LOCAL-STORAGE SECTION.

5.4. PROCEDURE DIVISION

Elle contient **toutes** les instructions permettant d'effectuer les traitements portant sur les informations spécifiées dans la DATA DIVISION.

Le programme-source se termine à la fin de la dernière ligne, ou à la rencontre de l'en-tête

```
END PROGRAM nom-programme .
```

6. Conventions

Pour définir le format des instructions et des clauses de COBOL nous appliquerons les conventions suivantes :

6.1. Tous les **mots** écrits en **majuscules** sont des **mots-réservés** ; les mots écrits en minuscules sont des mots-utilisateurs.

6.2. Les **mots**-réservés **soulignés** sont des **mots-clés** ; les mots-réservés non soulignés sont facultatifs.

6.3. Les **accolades** { } rassemblent les **options** de format, écrites les unes en dessous des autres, parmi lesquelles il faut en choisir une.

Exemple :

```
{ PICTURE }
{ PIC     } IS chaîne
```

6.4. Les **crochets** [ ] délimitent une **partie** du format **facultative**

Exemple :

```
ACCEPT nd [ FROM { DATE } ]
                 { TIME }
```


Ce format autorise les trois instructions

```
ACCEPT V1
ACCEPT V1 FROM DATE
ACCEPT V1 FROM TIME
```

V1 étant un mot utilisateur

6.5. Les **points de suspension** … indiquent que le mot précédent ou la partie précédente entre accolades ou crochets, peut être répétée plusieurs fois.

Exemple :

```
MOVE { nd1  } TO nd2 [,nd3] ...
     { lit1 }
```

Ce format autorise en particulier les instructions

```
MOVE 1 TO A, B
MOVE B TO C
```

A, B, C étant des mots utilisateurs

6.6. Les caractères de **ponctuation ,** et **;** apparaissant dans les formats sont facultatifs.

6.7. **Abréviations** utilisées dans ce support

nd : nom de donnée
nt : nom de traitement (paragraphe ou section)
ch : chaîne de caractère
ent : entier
lit : littéral (numérique ou chaîne de caractères)
fich : nom de fichier
enreg : nom d'enregistrement logique

CONCEPTS FONDAMENTAUX

1. Les verbes COBOL

1.1. Catégorie fonctionnelles de verbes

Verbes arithmétiques : `ADD,SUBTRACT, MULTIPLY, DIVIDE, COMPUTE`

Verbes d'entrée-sortie :`READ, WRITE, REWRITE, DELETE, ACCEPT, DISPLAY, START, OPEN, CLOSE, STOP, USE`

Verbes de rupture de séquence : `PERFORM, EXIT, STOP, GO TO, ALTER`

Verbes de communication inter-programmes : `CALL, CANCEL, EXEC`

Verbes de gestion de table : `SET, SEARCH`

Verbes de manipulation de données : `MOVE, INSPECT, STRING, UNSTRING, SORT, MERGE, RELEASE, RETURN`

Directives de compilation : `COPY, INSERT, REPLACE, USE`

Verbes de condition : `IF, EVALUATE,` les verbes d'entrée-sortie utilisés avec l'une des clauses `AT END` ou `INVALID KEY`, les verbes arithmétiques avec une phrase `SIZE ERROR`, `SEARCH avec WHEN`

1.2. Types d'instructions et de phrases

On appelle **instruction conditionnelle** une instruction identifiée par un verbe de condition ; les autres instructions sont dites **impératives**.

Une **phrase impérative** ne contient que des instructions impératives, sinon elle est de type **conditionnel**.

2. Représentation interne des données

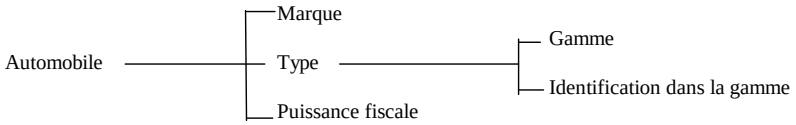
2.1. Caractéristiques d'une donnée

Une information ou **donnée** est la ***valeur*** d'un attribut qui caractérise un objet ou une entité.

Exemple : Si on considère l'objet "une automobile", on peut caractériser cet objet par sa marque, son type, sa puissance (entre autres) ; marque, type et puissance fiscale sont des attributs de "automobile", auxquels on peut donner des valeurs comme RENAULT, CLIO, 6CV.

Un attribut peut être un objet simple (p.ex. la puissance fiscale) ou non ; dans ce cas il peut être à son tour caractérisé par des attributs plus simples.

Exemple : Nous pouvons considérer que le type de la voiture est caractérisé par la gamme de fabrication et par une identification dans la gamme ; nous faisons apparaître deux attributs plus élémentaires distincts : la gamme CLIO et l'identification GL à l'intérieur de cette gamme.



Sur cet exemple, nous avons fait apparaître une structure hiérarchique entre les données. L'objet qui se trouve au sommet de la hiérarchie s'appelle une **entité de base**.

STRUCTURER des données consiste donc à regrouper ces données autour d'une entité de base en les classant suivant une hiérarchie.

Nous appelons **ARTICLE** (ou enregistrement logique) une telle structure de données.

Nous appelons **RUBRIQUE** la zone mémoire permettant de stocker la valeur d'un attribut.

En nous référant au schéma précédent, nous constatons que nous aurons deux sortes de rubriques :

- des **rubriques élémentaires** qui ne contiennent qu'une seule valeur non décomposée (ex. Marque, Gamme, etc..)
- des **rubriques structurées** : elles regroupent un ensemble quelconque de rubriques, élémentaires ou non (ex. Type)

Remarques :

a) La rubrique structurée la plus importante est celle qui est constituée de l'ensemble des rubriques relatives à un article.

b) La **décomposition** en rubriques d'une entité **dépend des fonctions à réaliser** et non des sous-attributs élémentaires qu'on peut mettre en évidence ; ainsi dans l'exemple précédent, il est inutile de décomposer l'attribut "type" en deux sous-attributs si aucun traitement n'utilise ces sous-attributs.

2.2. La hiérarchie des données en COBOL

La description de l'objet "automobile" se présente sous la forme d'une arborescence. Pour décrire cette arborescence de façon linéaire, on attribue à chaque niveau de la décomposition un **NUMERO** et un **ordre** d'écriture tels que pour tout attribut J dans un attribut I :

a) J est écrit après I

b) le numéro de J est strictement supérieur à celui de I

c) tout élément écrit entre I et J a un numéro strictement supérieur à celui de I.

```
01      AUTOMOBILE
            02      MARQUE
            02      TYPE
                    03      GAMME
                    03      IDENT-DANS-GAMME
            02      PUISSANCE
```

Les numéros de niveaux doivent être compris entre 01 (ou 1) et 49 ; ils ne sont **pas obligatoirement consécutifs** ; l'entité de base porte toujours le numéro 1.

2.3. Définition d'une rubrique

Chaque rubrique est identifiée par un nom qui doit être unique (cf. noms de données au chapitre 1).

Les rubriques élémentaires font l'objet d'une description plus détaillée à l'aide de clauses qui précisent leur nature, leur format, etc... et permettent d'avoir une **image théorique de l'attribut**. Ces clauses sont décrites au chapitre 3. Toutefois, notons dès à présent que l'on représente :

- par le symbole **X** une position de rubrique pouvant contenir un caractère quelconque
- par le symbole **9** une position de rubrique ne pouvant contenir qu'un chiffre.

Exemples : `MARQUE XXXXXXX`
qui s'écrit aussi `MARQUE X(7)`
(7 est un facteur de répétition)
La valeur Marque peut contenir une valeur sous forme d'une chaîne de 1 à 7 caractères.

`PUISSANCE 99`
La rubrique PUISSANCE peut contenir une valeur sous forme d'un nombre entier de 2 chiffres, comme 04.

2.4. Types et classes de rubriques

On distingue **cinq types** de rubriques :

- **alphabétique** : la rubrique ne contient que des lettres ou des espaces
- **numérique** : la rubrique ne contient que des chiffres
- **numérique édité** : la rubrique ne contient que des chiffres et des symboles d'édition (. , ~~b~~ , + , - $, etc...)
- **alphanumérique** : la rubrique contient des caractères quelconques
- **alphanumérique édité** : la rubrique contient des caractères quelconques et des symboles d'insertion (~~b~~ , /)

On distingue **trois classes** de rubriques :

- alphabétique
- numérique
- alphanumérique (constitué des 3 derniers types)

Remarque très importante :
Une rubrique structurée est toujours de classe alphanumérique, quel que soit son type.

2.5. Formats internes des données

➲ ***Tous les exemples sont donnés en ASCII***

2.5.1. Formats fondamentaux

On distingue **deux représentations** de base :

- **le format binaire**, ne concerne que les **entiers**. La valeur est codée en complément à 2, sur 2, 4ou 8 octets. Les plages de valeurs sont :
  -32768 à + 32767 pour la représentation sur 16 bits
  -2^{31} à $+2^{31} - 1$ ($\sim 10^9$) pour la représentation sur 32 bits
  -2^{63} à $+2^{63} - 1$

Désigné en COBOL par `BINARY, COMPUTATIONAL,` ou `COMPUTATIONAL-4`

- **la représentation codée**, peut être utilisée pour tous les types ; les valeurs sont codées en utilisant le code ASCII ou EBCDIC. (cf. chap. 1 par 4-1)

Une troisième représentation peut exister : la **virgule flottante**, simple (4 octets) ou double (8 octets).

Désigné par `COMPUTATIONAL-1, COMPUTATIONAL-2.`

2.5.2 Représentation codée

2.5.2.1. Rubrique de type non numérique

Chaque caractère de la rubrique occupe exactement un octet ; sa valeur est représentée par son code ASCII

Exemple : Soit la rubrique MARQUE d'image `X(7)` ; la valeur PEUGEOT est représentée (en hexadécimal)

50	45	55	47	45	4F	54

7 octets

Désigné par `DISPLAY`.

2.5.2.2. Rubriques de type numérique

On distingue deux représentations.

2.5.2.2.1 La représentation décimale étendue (`DISPLAY`)

Chaque chiffre occupe un octet et est représenté par son code ASCII ou EBCDIC

Exemple : Soit la rubrique `PUISSANCE` d'image `99` ; la valeur 06 est représentée (en hexadécimal, ASCII)

30	36

2 octets

Si le nombre est signé, il y a deux manières de préciser le signe :

a) Le signe occupe une position propre
Sa valeur est donnée par le code correspondant au caractère + (soit 2B) ou - (soit 2D) ; elle est placée dans une position séparée soit en tête de zone, soit en fin de zone ; ce choix est fixé par le programmeur.

Exemple :
Soit la rubrique `MONTANT` d'image `S9(5)`, (le symbole S indique une rubrique signée) avec **signe séparé en tête** ; la valeur + 953 est représentée :

2B	30	30	39	35	33

6 octets : 5 chiffres + le signe

En prenant la même rubrique, mais avec une déclaration de **signe séparé en fin**, nous aurions obtenu

30	30	39	35	33	2B

b) Le signe n'occupe pas de position séparée

Dans ce cas, la codification du signe est "superposée", à celle du chiffre de plus faible ou de plus fort poids (par analogie avec les cartes perforées, on parle de "sur-perforation").

Le tableau suivant donne les résultats de signe des chiffres perforés en hors texte

Chiffre de faible poids	Valeurs signées positives		Valeurs signées négatives	
	caractère	code ASCII	caractère	code ASCII
0	{	7B	}	7D
1	A	41	J	4A
2	B	42	K	4B
3	C	43	L	4C
4	D	44	M	4D
5	E	45	N	4E
6	F	46	O	4F
7	G	47	P	50
8	H	48	Q	51
9	I	49	R	52

Exemple : Soit la rubrique MONTANT d'image S9(5) (l'absence de spécification entraîne "signe superposé"); la valeur + 953 est représentée :

30	30	39	35	43

5 octets

On remarque que la valeur du dernier octet est identique au code ASCII de la lettre C.

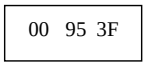
Remarques : Utiliser de préférence la forme (a) car le système s'y ramène dans les opérations arithmétiques.

2.5.2.2.2 La représentation décimale condensée

(`PACKED-DECIMAL ou COMPUTATIONAL-3`**)**

Pour gagner de la place sur les supports et accélérer les traitements, il peut être utile de placer deux codes numériques par octet ; en effet, le quartet de gauche dit hors-texte, est identique pour tous les chiffres et vaut ‘3’ en ASCII et ‘F’ en EBCDIC.

Exemple : Soit la rubrique MONTANT d'image `9(5)` avec représentation décimale condensée ; la valeur 953 est représentée



"F" pour non signée

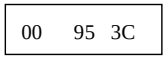
3 octets

Le premier chiffre est rajouté par le système pour compléter le premier octet ; il peut être quelconque et n'intervient pas dans les traitements ; seuls les 5 chiffres de droite, dûment déclarés, seront pris en compte (sauf cas particuliers).

Si la rubrique est signée, la codification du signe occupe le dernier demi-octet de la zone ; elle vaut 'C' pour + et 'D' pour -.

Exemple : Soit la rubrique MONTANT d'image S9(5) avec représentation décimale condensée.

La valeur + 953 est représentée :



3 octets

A noter que sur cet exemple le nombre de positions est pair et donne un nombre entier d'octets.

Remarques :

- Une rubrique de type numérique, non signée, sera toujours considérée comme contenant un nombre positif, quelle que soit la représentation utilisée.

- Sur la plupart des ordinateurs on dispose d'instructions machine permettant d'effectuer des opérations arithmétiques sur des nombres représentés en format binaire ou en format décimal condensé ; les nombres en format décimal étendu doivent être d'abord être convertis dans l'un des deux formats précédents.

2.5.3 Longueur d'une rubrique :

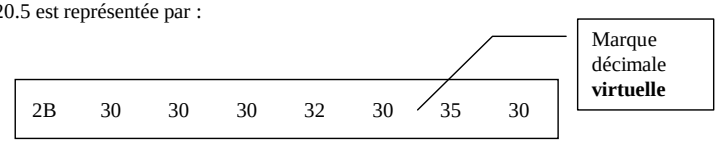
Elle dépend de la représentation interne choisie et de l'image

- Format binaire --> Longueur : 4 octets
- Rubrique de type non numérique, N symboles indiqués dans l'image --> longueur : N octets
- Représentation décimale étendue, sans signe ou avec signe superposé,
  N chiffres indiqués --> Longueur N octets
- Représentation décimale étendue, signe séparé, N chiffres indiqués --> Longueur : N + 1 octets
- Représentation décimale condensée, N symboles indiqués (y compris le symbole S si la rubrique est signée)
  Si N est pair --> Longueur : N/2 octets
  Si N est impair --> Longueur : (N + 1)/ 2 octets

2.5.4 Le séparateur décimal :

Pour séparer la partie entière de la partie fractionnaire d'un nombre on utilise le point (cf Littéraux numériques au chap. 1 § 4.2.1.). La représentation de ce point au niveau du format interne d'une donnée numérique n'est pas utile. Il suffit d'indiquer au système la **position virtuelle** de ce séparateur ; ceci se fait dans la description de l'image de la rubrique à l'aide du symbole `V`.

Exemple :
Soit la rubrique MONTANT d'image `S9(5)V99`, en représentation décimale étendue avec signe séparé en tête ; la valeur + 20.5 est représentée par :



8 octets (dont aucun pour la marque décimale)

Remarques :

- Le symbole V est **unique** au niveau d'une description d'une image ; s'il est omis, il est supposé être à droite et l'image correspond alors à celle d'un nombre entier.
- Si la rubrique doit être visualisée (sur écran ou sur papier), il est nécessaire de faire figurer le point au niveau de l'affichage ; pour cela on utilise des rubriques spéciales de type numérique édité étudiées ultérieurement.

3. Règles d'alignement

Ces règles sont appliquées chaque fois que l'on mémorise une valeur dans une rubrique. Le choix de l'une des deux règles dépend du type de la rubrique élémentaire qui "reçoit" la valeur.

3.1. Règle d'alignement numérique

Elle s'applique si la rubrique **réceptrice** est de type **numérique** ou **numérique éditée**. La valeur est alignée sur la marque décimale ; si la zone est trop grande, elle est complète vers ses extrémités par des zéros ; si elle est trop petite, la valeur est tronquée à l'une ou l'autre des extrémités.

Exemple :
Soit la rubrique R1 d'image `9(2)V9` ; indépendamment de la représentation interne, elle peut contenir un nombre de 3 chiffres dont 2 pour la partie entière.

Valeur transférée	Valeur obtenue
21	21.0
1.5	01.5
1.67	01.6
462.1	62.1
520.02	20.0

3.2. Règle d'alignement non numérique :

Elle s'applique si la rubrique réceptrice est de type **alphabétique** ou **alphanumérique** ou **alphanumérique édité**. La valeur est alignée sur la position d'extrême gauche de la rubrique réceptrice ; si la zone est trop grande, la valeur est complétée vers l'extrémité droite par des caractères "espaces" ; si elle est trop petite, la valeur est tronquée à droite.

Exemple :

Soit la rubrique R2 d'image `X(5)`

Valeur Transférée	Valeur obtenue
"VILLES"	"VILLE"
"BLANC"	"BLANC"
"BEAU"	"BEAU␣"

4. Unicité de référence

Un même mot-utilisateur pet être utilisé pour désigner des données différentes, sous réserve qu'ellea appartiennent à des groupes différents. Pour les référencer sans ambigüité, on **qualifiera** leur nom par celui du groupe, ou du fichier les contenant. La même disposition s'applique aux paragraphes de sections différentes et aux textes-source de bibliothèques différentes.

nd-1 { { IN / OF } nd-2 } ... { { IN / OF } fich-1 }

Exemple

```
01 IDENTITE.
     02  NOM          PIC X(20).
     02  PRENOM       PIC X(15).

01 INDIV.
     02  NOM          PIC X(20).
     02  PRENOM       PIC X(15).
.
.
     MOVE NOM OF IDENTITE TO NOM OF INDIV.
```


5. Règles de ponctuation

4.1. Un ou plusieurs **espaces** consécutifs servent à séparer deux mots.

4.2. La **virgule**, le **point-virgule** et le **point**, utilisés comme séparateurs, doivent être suivis d'un espace et ne peuvent être utilisés que lorsque le format des instructions l'autorise.

4.3. Les **parenthèses** s'utilisent toujours par paires.

4.4. Les **guillemets** délimitent les littéraux non numériques :
- le guillemet "ouvrant" doit être précédé d'un espace
- le guillemet "fermant" doit être suivi d'un espace, d'une virgule, d'un point-virgule ou d'un point.

Les guillemets s'utilisent par paires (sauf lorsqu'un littéral s'écrit sur 2 lignes : cf § 5)

6. Feuille de saisie COBOL

Elle comprend cinq zones, respectivement de gauche à droite :
- zone de numérotation (col. 1 à 6) (inutilisée, ou de contenu quelconque)
- zone indicateur (col. 7)
- zone A (col. 8 à 11) ; la colonne 8 est dite marge A
- zone B (col. 12 à 72) ; la colonne 11 est dite marge B
- zone d'identification (col. 73 à 80) (inutilisée)

Les **en-têtes** de division, section, paragraphes et de description de fichier commencent en zone A. Les instructions et clauses s'écrivent en zone B. Une instruction peut être rédigée sur plusieurs lignes. Il est possible d'inclure des lignes totalement vierges à tout endroit du programme.

La zone indicateur (col. 7) assure trois fonctions selon le caractère qui y est écrit :

- **continuation** d'un littéral non numérique portant jusqu'à la col.72 de la ligne précédente

***** ligne **commentaire** : Le contenu de cette ligne ne sera pas pris en compte par le compilateur ; cette ligne ne sert qu'à améliorer la lisibilité et la compréhension du programme.

/ idem * ; de plus saut de page avant l'impression du commentaire

D ou **d** instrument de **mise au point** : Cette fonction sera étudiée plus tard.

Colonne	0 1 2 3 4 5 6 7 8 123456789012345678901234567890123456789012345678901234567890123456789012345678
Marge	A B
	* CECI EST UNE LIGNE DE COMMENTAIRES
	77 Exemple pic x(75) value "ce littéral non-numérique va jusqu'en colonne72
	- " et se prolonge par la suite"

Exemple de programme COBOL

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PUBLI.

 AUTHOR. GR.
***------------------------------------------------

 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.
 SOURCE-COMPUTER. MWB.
 OBJECT-COMPUTER. MWB.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

          SELECT BASE ASSIGN TO BASABON
              ORGANIZATION RECORD SEQUENTIAL
              ACCESS SEQUENTIAL
              FILE STATUS IS BS.

          SELECT MOUV ASSIGN TO MAJABON
              ORGANIZATION RECORD SEQUENTIAL
              ACCESS SEQUENTIAL
              FILE STATUS IS MS.

          SELECT NOUBASE ASSIGN TO NEWABON
              ORGANIZATION RECORD SEQUENTIAL
              ACCESS SEQUENTIAL
              FILE STATUS IS NS.

          SELECT ERREUR ASSIGN TO ERRABON
              ORGANIZATION LINE SEQUENTIAL
              ACCESS SEQUENTIAL
              FILE STATUS IS ES.
***------------------------------------------------

 DATA DIVISION.

 FILE SECTION.

*-------------------------------------------------------------
 FD  BASE
           LABEL RECORD STANDARD.
 01  B-ENR.
* Code-adresse
      05 B-CD-ADR             PIC 9(6).
* Cle de controle
      05 B-CLE                PIC X.
* Nom-Prenom
      05 B-NM-PRNM            PIC X(25).
* Adresse
      05 B-ADR                PIC X(40).
* Code Postal
      05 B-CD-PSTL            PIC 9(5).
* Ville
      05 B-VL                 PIC X(25).
```



```
*-------------------------------------------------------------
 FD  MOUV
           LABEL RECORD STANDARD.
 01  M-ENR.
      05 M-CD-OPR             PIC X.
      05 M-DONNEES.
        10 M-CD-ADR           PIC 9(6).
        10 M-CD-ADRX          REDEFINES M-CD-ADR  PIC X(6).
        10 M-CLE              PIC X.
        10 M-NM-PRNM          PIC X(25).
        10 M-ADR              PIC X(40).
        10 M-CD-PSTL          PIC 9(5).
        10 M-VL               PIC X(25).
*-------------------------------------------------------------
 FD  NOUBASE
      LABEL RECORD STANDARD.
 01  N-ENR                    PIC X(102).

*-------------------------------------------------------------
 FD  ERREUR
      LABEL RECORD STANDARD.
 01  E-ENR                    PIC X(103).

*-------------------------------------------------------------
 WORKING-STORAGE SECTION.
* Code etat du fichier BASE
 77  BS                       PIC XX.
* Code etat du fichier MOUV
 77  MS                       PIC XX.
* Code etat du fichier NOUBASE
 77  NS                       PIC XX.
* Code etat du fichier ERREUR
 77  ES                       PIC XX.
* Zone de travail
 77  QUOTIENT                 PIC 9(6) COMP.
* Rang de la lettre de controle
 77  RG-LTR                   PIC 99 COMP.
          88  CORRECTE  VALUE 1 THRU 23.
* Rang du message d'erreur
 77  RG-ERR                   PIC 9.

* Definition des cles de controle
 01  LETTRES      PIC X(23) VALUE "ABCDEFGHJKLMNPQRSTVWXY".
 01  TABCLE       REDEFINES LETTRES.
          05  CLE             PIC X OCCURS 23.

* Definition des messages d'erreur
 01  LST-MSG.
          05  FILLER PIC X(30) VALUE "(1) CLE ERRONEE".
          05  FILLER PIC X(30) VALUE SPACES.
          05  FILLER PIC X(30) VALUE SPACES.
          05  FILLER PIC X(30) VALUE "(4) CODE OPER. ERRONE".
          05  FILLER PIC X(30) VALUE "(5) DONNEE INCORRECTE".
          05  FILLER PIC X(30) VALUE SPACES.
          05  FILLER PIC X(30) VALUE SPACES.
          05  FILLER PIC X(30) VALUE SPACES.
 01  TABERR       REDEFINES LST-MSG.
          05  MSGERR          PIC X(30) OCCURS 8.
```

```
***-------------------------------------------------

 PROCEDURE DIVISION.
 DECLARATIVES.

 LECT SECTION. USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
 L0. IF MS = 10 OR BS = 10 NEXT SENTENCE
     ELSE
     DISPLAY "Erreur d'entrée-sortie en lecture" UPON CONSOLE
            DISPLAY "Code d'etat des fichiers:" UPON CONSOLE
     DISPLAY "BASABON = " BS "      MAJABON = " MS UPON CONSOLE
            CLOSE MOUV BASE NOUBASE ERREUR.
            STOP RUN.

 ECR SECTION. USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
 E0.
     DISPLAY "Erreur d'entrée-sortie en ecriture" UPON CONSOLE
            DISPLAY "Code d'etat des fichiers:" UPON CONSOLE
     DISPLAY "NOUBASE = " NS "      ERREUR = " ES UPON CONSOLE
            CLOSE MOUV BASE NOUBASE ERREUR.
            STOP RUN.

 END DECLARATIVES.

 PUBLI-MAIN SECTION.

* Initialisation du traitement
 INIT.
           OPEN    INPUT  BASE MOUV
                   OUTPUT NOUBASE ERREUR.
           PERFORM LIRBAS.
           PERFORM LIRMAJ.

* Traitement majeur
 CORPS.
           PERFORM PRINCIP UNTIL BS NOT = ZERO OR MS NOT = ZERO.
           PERFORM FINBAS UNTIL BS NOT = ZERO.
           PERFORM FINMAJ UNTIL MS NOT = ZERO.

* Finalisation du traitement
 FIN.
           CLOSE BASE MOUV NOUBASE ERREUR.
           DISPLAY "FIN DE PUBLI" UPON CONSOLE.
           STOP RUN.

*-----------------------------------------------------------------
 SPECIALE SECTION.

* Repetitive principale
 PRINCIP. PERFORM CALCLE. IF  CORRECTE PERFORM TRAIT
                           ELSE PERFORM ERR PERFORM LIRMAJ.

* Traitement final de BASE
 FINBAS.  PERFORM COPIE   PERFORM LIRBAS.

* Traitement final de MOUV
 FINMAJ.  IF M-CD-OPR = "C" PERFORM AJOU
```



```
                ELSE MOVE 4 TO RG-ERR  PERFORM ERR.
            PERFORM LIRMAJ.

 LIRBAS.  READ BASE.

 LIRMAJ.  READ MOUV.

* Validation de la cle
 CALCLE.  IF M-CD-ADRX NOT NUMERIC
                            MOVE ZERO TO RG-LTR
                            MOVE 5 TO RG-ERR
             ELSE
               DIVIDE 23 INTO M-CD-ADR     GIVING QUOTIENT
                                           REMAINDER RG-LTR
               SUBTRACT RG-LTR FROM 23 GIVING RG-LTR
               IF CLE(RG-LTR) NOT = M-CLE
                            MOVE ZERO TO RG-LTR
                            MOVE 1 TO RG-ERR.

* Deroulement de BASE
 TRAIT.   IF M-CD-ADR > B-CD-ADR  PERFORM COPIE PERFORM LIRBAS
                ELSE PERFORM NONSUP.

* Ecriture des erreurs
 ERR.     WRITE E-ENR FROM M-ENR. WRITE E-ENR FROM MSGERR(RG-ERR).

* Ecriture de NOUBASE
 COPIE.   WRITE N-ENR FROM B-ENR.

* Ajout d'un mouvement en creation
 AJOU.
*------  Il faudrait controler la validite des rubriques!!
                WRITE N-ENR FROM M-DONNEES.

 NONSUP.  IF M-CD-ADR = B-CD-ADR
                PERFORM EGAL
              ELSE  IF M-CD-OPR = "C"
                    PERFORM AJOU
                         ELSE MOVE 4 TO RG-ERR PERFORM ERR.

          PERFORM LIRMAJ.

* Code-adresse existant dans BASE
 EGAL.    IF M-CD-OPR = "S"
                      NEXT SENTENCE
              ELSE IF M-CD-OPR = "M"
                       PERFORM MODIF
                       PERFORM COPIE
                           ELSE PERFORM COPIE
                              MOVE 4 TO RG-ERR PERFORM ERR.
           PERFORM LIRBAS.

* Chargement des champs a modifier
 MODIF.   IF M-NM-PRNM NOT = SPACES MOVE M-NM-PRNM TO B-NM-PRNM.
          IF M-ADR     NOT = SPACES MOVE M-ADR     TO B-ADR.
          IF M-CD-PSTL NOT = SPACES MOVE M-CD-PSTL TO B-CD-PSTL.
          IF M-VL      NOT = SPACES MOVE M-VL      TO B-VL.
```

LES INSTRUCTIONS DU NOYAU

1. La division IDENTIFICATION

Elle identifie le programme

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  nom-de-programme.
[AUTHOR.  [rubrique-commentaire]]
[INSTALLATION. [rubrique-commentaire]]
[DATE-WRITTEN. [rubrique-commentaire]]
[DATE-COMPILED. [rubrique-commentaire]]
[SECURITY. [rubrique-commentaire]]
[REMARKS. [rubrique-commentaire]]
```

Tous les paragraphes autres que PROGRAM-ID sont obsolètes.

Généralement, seuls les 8 premiers caractères du nom du programme sont évalués.
ANS85 introduit un second format pour PROGRAM-ID

```
PROGRAM-ID.  nom-de-programme [IS [COMMON] [INITIAL] PROGRAM].
```

COMMON: uniquement pour les programmes contenus
INITIAL: programme remis à son état initial à chaque appel.
Cf Communication inter-programme

2. La division ENVIRONMENT

Elle contient la description du **système** utilisé et des informations relatives à la gestion des entrées-sorties.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. ordinateur-origine [WITH DEBUGGING MODE].
OBJECT-COMPUTER. ordinateur-objet.
[SPECIAL-NAMES. [rubrique-des-noms-spéciaux]]
[INPUT-OUTPUT SECTION.
FILE-CONTROL. rubrique-de-gestion-de-fichier ....
[I-O-CONTROL. [rubrique-de-gestion-d'entrée-sortie]]]
```

En ANS85, cette division est facultative.

Le paragraphe SPECIAL-NAMES a le format suivant (partiel).

```
SPECIAL-NAMES.

[définition des mnémoniques du réalisateur]
[ définition de l'alphabet]
[CURRENCY SIGN IS lit-1]
[DECIMAL-POINT IS COMMA]
[NUMERIC SIGN IS TRAILING SEPARATE]
[CONSOLE IS CRT]
```

♦ Le symbole monétaire (CURRENCY SIGN), par défaut $, doit être un caractère **autre** que : un chiffre, une minuscule, un symbole arithmétique ou de ponctuation, les lettres A, B, C, D, P, R, S, V, X, Z.

Exemple : `CURRENCY SIGN IS "F".`

♦ La clause DECIMAL-POINT IS COMMA permet de permuter le rôle du point et de la virgule dans une clause PICTURE et dans les littéraux numériques.

Les clauses [NUMERIC SIGN IS TRAILING SEPARATE] et [CONSOLE IS CRT] relèvent du dialecte Microfocus.

L'INPUT-OUTPUT SECTION est détaillée dans le chapitre des fichiers.

3. La division des données

Elle contient la description de toutes les informations manipulées par le programme.

3.1. Organisation générale :

```
DATA DIVISION.

FILE SECTION.

rubrique-de-description-de-fichier
rubrique-de-description-d'article          ...


WORKING-STORAGE SECTION.

rubrique-de-description-de-niveau-77
rubrique-de-description-d'article          ...


LINKAGE SECTION.

rubrique-de-description-de-niveau-77
rubrique-de-description-d'article          ...
```

Trois sections :

♦ FILE SECTION : description des fichiers (voir chap. 6)

- `WORKING-STORAGE SECTION` : description des données internes au programme.

- `LINKAGE SECTION` : description des paramètres lorsque le programme est un sous-programme (détaillée au : Communications inter-programmes).

Signalons toutefois qu'à quelques exceptions près, la description d'une rubrique se fait de façon identique dans les trois sections. Dans les paragraphes qui suivent, nous nous placerons en priorité au niveau de la `WORKING-STORAGE SECTION`.

Ces sections peuvent être suivies de la `COMMUNICATION SECTION` et de la `REPORT SECTION` si l'on utilise respectivement les modules de communication et d'édition , ainsi que de sections spécifiques aux dialectes (`SCREEN, LOCAL-STORAGE`, ...).

3.2. Description générale d'une rubrique

Elle est de la forme :

numéro-de-niveau { nd | FILLER } [[clause] ...] .

- le numéro de niveau est compris entre 1 à 49 ou vaut 77, 66 ou 88.

- nd : nom de donnée, identifie la rubrique.

- le mot clé `FILLER` remplace le nom d'une rubrique qui n'est jamais référencée exeplicitement par les instructions du programme ; il peut être utilisé dans plusieurs descriptions de rubriques du même programme.

La description d'une rubrique structurée se réduit au couple (n° de niveau, nom) ; les rubriques élémentaires doivent être détaillées par des clauses (sauf les index).

Exemple :

```
02 DATE-NAISSANCE.
  03 JOUR ....       .
  03 MOIS ....       .
  03 ANNEE ...       .
```


3.3. La clause PICTURE

Elle sert à définir le format externe ou image d'une rubrique élémentaire (nombre de chiffres, position de la marque décimale, etc...)

{ PICTURE | PIC } `IS` Format

Le format est constitué d'une suite de symboles décrits ci- dessous (liste partielle) :

- `9` : représente un chiffre
- `V` : indique la position **virtuelle** du séparateur décimal ; unique dans un format
- `S` : indique une donnée signée ; unique dans un format, il doit être le premier symbole du format
- `X` : représente un caractère quelconque
- `.` : matérialise le séparateur décimal : le format interne de la rubrique doit être le format décimal étendu ; son utilisation exclut celle de V et de S.

- Les 3 premiers symboles permettent de décrire des données de type numérique : le symbole X permet de décrire des données de type alphanumérique (chaînes de caractères) ; la présence du symbole **.** dans un format indique une donnée de type numérique édité.

- La description du format peut comporter jusqu'à **30** symboles ;

- L'emploi de facteurs de répétition est conseillé : `X(5)` au lieu de `XXXXX`

- Cette clause est obligatoire pour toutes les rubriques élémentaires. Elle est interdite pour les index.

Exemple :

1) `PIC X(5)` chaînes de 5 caractères

2) `PIC 9(3)` entier de 3 chiffres non signé

3) `PIC 99V9` nombre décimal non signé ; 2 chiffres pour la partie entière, 1 chiffre pour la partie décimale

4) `PIC S9(5)` entier de 5 chiffres, signé

5) `PIC 99.9` nombre décimal en format numérique édité ; le séparateur décimal est matérialisé par un point.

Remarquons que la rubrique de l'exemple 5 est de classe alphanumérique (du fait de la présence du point) alors que les rubriques des exemples 2 à 4 sont de type numérique.

Il existe d'autres symboles pour décrire un format numérique édité (*Cf Module Edition*)

Capacité maximum d'une rubrique :

- 18 chiffres, signe exclu, pour une rubrique numérique.
- 31 caractères pour une rubrique numérique éditée
- 32768 caractères pour une rubrique alphanumérique

Modification de Référence

En ANS85, il est possible de référencer une sous-chaîne d'une donnée alphanumérique:

nd(*début*:[*longueur*]) désigne la sous-chaîne de nd débutant à la position *début* et comportant *longueur* caractères.

Le rang du premier caractère est 1.

début et *longueur* est une expression numérique quelconque
Si *longueur* n'est pas précisée, elle porte jusqu'au dernier caractère de nd.

Si nd est de type autre qu'alphanumérique, on le considère comme redéfini dans une donnée de ce type de la même taille que nd.

Exemple

```
77  DONNEE PIC X(10)  VALUE "abcdefghij".

DONNEE(5:2) renvoie  "ef"          tout comme DONNEE(I:I-3)  si I vaut 5
DONNEE(5:)  renvoie  "efghij"
```


3.4. La clause USAGE

Elle permet de préciser le format interne d'une rubrique.

```
USAGE  IS  { DISPLAY
             COMPUTATIONAL
             BINARY
             COMPUTATIONAL-3
             PACKED-DECIMAL }   ou autres, suivant dialectes
```

- `DISPLAY` : représentation codée de type non numérique ou décimale étendue (cf. §2.5.2.2.) ; valeur prise par défaut
- `COMPUTATIONAL-3` ou `COMP-3` ou `PACKED-DECIMAL`: représentations décimales condensées signées.
- `COMPUTATIONAL-1 (COMPUTATIONAL-2)` = virgule flottante simple (double) précision
- `COMPUTATIONAL` ou `COMP` ou `BINARY` : format binaire sur 32 bits (moins de 10 chiffres dans la PICTURE)

  Cette clause peut s'appliquer à une rubrique élémentaire ou structurée ; dans ce cas, elle est valable pour toutes les rubriques élémentaires qui composent la rubrique structurée.

3.5. La clause REDEFINES

Elle permet de redéfinir la **structure** d'une rubrique (et non sa ***valeur***).

n°-de-niveau nd1 `REDEFINES` nd2

Règles du langage :

- cette clause doit suivre immédiatement nd1.
- nd1 et nd2 doivent être de même niveau
- nd2 ne peut contenir la clause `REDEFINES` mais nd2 peut être subordonné à une rubrique qui contient cette clause.
- cette clause est interdite pour les rubriques de niveau `01` de la `FILE SECTION`.
- nd1 peut être remplacé par FILLER

Principe :

La redéfinition de la rubrique nd2 commence à nd1 et se termine lorsqu'un numéro de niveau inférieur ou égal à celui de nd1 est rencontré.

```
02    D1.
      04 NOM      PIC X(20).
      04 PRENOM   PIC X(20).
02    D2 REDEFINES   D1.
      03 ADRESSE    PIC X(20).
      03 C-POSTAL   PIC X(5).
      03 LOCALITE   PIC X(15).
```

En cas de redéfinitions multiples, c'est toujours la rubrique d'origine qui est citée en nd2.

```
03    DATE-1.
        04  JOUR-1    PIC 99.
        04  MOIS-1    PIC 99.
        04  AN-1      PIC 99.
03    DATE-2 REDEFINES   DATE-1 PIC 9(6).
03    DATE-3 REDEFINES   DATE-1.
        04  FILLER  PIC X.
        04  JOUR-3  PIC 9(3).
        04  AN-3    PIC 99.
```

Sauf si les rubriques nd1 et nd2 sont de niveau 01, les longueurs de la rubrique de redéfinition et de la rubrique d'origine doivent être égales.

3.6. La clause VALUE

Elle permet de préciser la valeur initiale d'une rubrique ; une rubrique déclarée sans valeur initiale contient une valeur quelconque non définie.

VALUE IS littéral

Le littéral doit être du même type que la rubrique.

Cette clause est interdite :

- dans la FILE SECTION
- à l'intérieur d'une rubrique de redéfinition.

Exemple

```
02  D1  PIC  X(5)  VALUE "DEBUT".
03  N1  PIC  99V9 VALUE 5.2.
```

Une seconde utilisation de la clause VALUE permet la définition de noms-conditions.

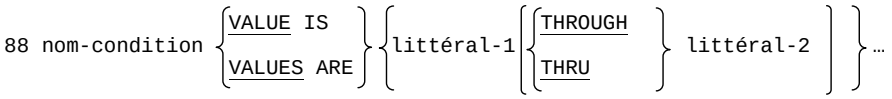
Un **nom-condition** est une référence symbolique assignée à une ou plusieurs **valeurs** spécifiques d'une variable donnée, ou à un inverseur.

Il sera défini soit dans le paragraphe SPECIAL-NAMES, soit en DATA DIVISION. Dans ce dernier cas, il est repéré par un numéro de niveau 88.

Exemple :
```
01  MOUVEMENT.
      02  INDIC      PIC 9(6).
      02  TYP        PIC 9.
                    88  LECTURE  VALUE 1.
                    88  MODIF    VALUE 2.
                    88  SUPPRES  VALUE 4.

    IF LECTURE.................
           ELSE IF MODIF...........
```

L'intérêt des noms-conditions est donc d'améliorer la lisibilité du pg ; ils permettent aussi de simplifier l'énoncé de conditions, car un seul nom-condition peut être assigné à plusieurs valeurs en particulier grâce à la locution THRU :

Format général :



Exemples :
```
VALUES ARE 1  2   16.
VALUES ARE 4  THRU 12.
VALUES ARE 5  THRU 9  13  17.
```

Un nom-condition doit suivre la description de la rubrique à laquelle il est associé ; cette dernière peut être quelconque à l'exception de :

- un autre nom-condition
- une rubrique de niveau 66 (clause RENAMES)
- un index
- un groupe contenant des données avec JUSTIFIED, SYNCHRONIZED, et USAGE (autre que DISPLAY).

Si la référence à une variable conditionnelle nécessite l'indexation ou l'indiçage, la référence à l'un quelconque de ses noms-conditions la nécessite de la même façon.

Les règles d'alignement appliquées sont celles décrites au chapitre 2 § 3.

Utilisation des constantes figuratives

Ce sont des constantes identifiées par des mots clés ; elles ont des valeurs particulières.

ZERO
ZEROS représente soit "0", soit un ou plusieurs caractères "0" selon le contexte

ZEROES

SPACE SPACES	représente un ou plusieurs caractères espace
HIGH-VALUE HIGH-VALUES	représente un ou plusieurs caractères de rang le plus élevé dans l'ordre de classement ASCII. La valeur la plus élevée est FF hexadécimale.
LOW-VALUE LOW-VALUES	représente un ou plusieurs caractères de rang le plus bas dans l'ordre de classement ASCII. La valeur la plus basse est 00 hexadécimal.
QUOTE QUOTES	représente un ou plusieurs caractères guillemets (") ; le mot QUOTE ou QUOTES ne peut être utilisé à la place d'un guillemet dans un programme origine pour borner un littéral non- numérique. Ainsi QUOTE ABD QUOTE est une façon incorrecte d'écrire le littéral non numérique "ABD".
ALL *littéral*	représente une ou plusieurs fois le *littéral* non- numérique.

- La première (ZERO) peut s'appliquer à des rubriques de type numérique ou non numérique ; les autres sont de type non numérique.
- Elles s'écrivent sans guillemet.
- Contrairement aux autres constantes, leur longueur dépend de la longueur des rubriques qu'elles servent à initialiser.

Exemples :

```
D1   PIC   X(5)   VALUE   "0".        0bbbb
D1   PIC   X(5)   VALUE ZERO.         00000
D1   PIC   X(5)   VALUE ALL "-"       -----
```

3.7. La clause BLANK

Elle permet de remplacer la valeur nulle d'une rubrique de type numérique ou numérique édité par des espaces.
Le format interne de la rubrique doit être DISPLAY.

```
BLANK   WHEN   { ZERO   }
               { ZEROS  }
               { ZEROES }
```

La rubrique sera considérée comme étant du type numérique édité.

Exemple : `D1  PIC  9(5)  BLANK ZERO.`

Si D1 est affecté d'une valeur nulle, la rubrique sera remplie par 5 espaces.

3.8. La clause SYNCHRONIZED

Elle permet de préciser l'**alignement** d'une rubrique élémentaire sur les bornes d'adressage spécifiques de la mémoire de l'ordinateur à savoir :

- une adresse multiple de 2 pour un mot
- une adresse multiple de 4 pour un double mot.

```
{ SYNCHRONIZED } { LEFT  }
{ SYNC        } { RIGHT }
```

Elle ne peut s'utiliser que pour des rubriques élémentaires de format interne binaire.

Les options LEFT et RIGHT sont équivalentes.

Les rubriques de niveau 1 et 77 commencent à des limites de mot.

3.9. La clause JUSTIFIED

Elle permet, associée à une zone de réception alphabétique ou alphanumérique, de force l'alignement à droite, plutôt qu'à gauche (alignement standard).

```
{ JUSTIFIED } RIGHT
{ JUST      }
```

3.10. La clause SIGN

Elle permet de préciser la représentation interne du signe pour une rubrique numérique de format décimal étendu. (cf.§ 2.5.2.2-2).

```
[SIGN  IS]   { LEADING  } [SEPARATE  CHARACTER]
             { TRAILING }
```

- Elle peut être spécifiée pour un groupe.
- le format externe (PICTURE) doit comporter le symbole S
- la clause indique que le signe occupe dans la représentation interne une position séparée à gauche (LEADING) ou à droite (TRAILING).
- l'absence de cette clause pour une rubrique numérique signée, de format interne décimal étendu, entraîne une représentation du signe superposée avec celle du chiffre de droite.

Exemple : `D1  PIC  S999V9  SIGN TRAILING SEPARATE`

3.11. La clause RENAMES

Elle permet un regroupement alternatif des rubriques.

66 nd-1 RENAMES nd-2 { THROUGH / THRU } nd-3

- Interdite aux niveaux 01, 66, 77, 88.

Exemple:
```
01  EMPLOYE.
     02  NOM          PIC X(20).
     02  PRENOM       PIC X(15).
     02  RUE          PIC X(30).
     02  CP           PIC 9(5).
     02  VILLE        PIC X(25).
     66 IDENTITE  RENAMES NOM THRU PRENOM.
     66  ADRESSE   RENAMES RUE THRU VILLE.
```

3.12. Exemple de synthèse

```
77   N  USAGE COMP.
77   TOTAL-TTC  PIC 9(5).99 BLANK ZERO.
01   DONNEE-1.
    05  CODE-ART-1PIC X(5)    VALUE SPACE.
    05  PRIX-UNIT-1     PIC 9(4)V99 COMP-6.
    05  QUANTITE-1      PIC S9(3)   COMP-3 VALUE +0.
    05  QUANTITE-2REDEFINES   QUANTITE-1 PIC S9V99   COMP-3.
    05  COEF-1          PIC S99V99  SIGN TRAILING  SEPARATE.
```

4. La division PROCEDURE

Elle contient la description des traitements à effectuer sur les informations spécifiées dans la division des données.

4.1. Expressions arithmétiques

Une expression arithmétique est une combinaison syntaxiquement correcte de symboles pris dans l'ensemble suivant :

- les noms de données identifiant des rubriques élémentaires de type numérique
- les constantes de type numérique
- les opérateurs arithmétiques

- les parenthèses

Les opérateurs arithmétiques sont, dans l'ordre décroissant des priorités :

- les opérateurs unaires + et -
- l'exponentiation **
- la multiplication * et la division /
- l'addition + et la soustraction -

Les principales règles de syntaxe à respecter sont au nombre de deux :

1. deux opérandes (variable ou constante) sont toujours reliés par un opérateur arithmétique ; toutes les opérations doivent être spécifiées explicitement.

2. deux opérateurs arithmétiques ne se suivent jamais sauf si le second est un opérateur unaire.
   Exemples :
   ```
   (A + B  * C ) / 3.5
   - A / - (B + 1)
   ```

   Remarques :

   1) Le même symbole / sert à spécifier la division réelle et la division entière ; le choix est déterminé par le type des objets manipulés : / indique une division entière **si et seulement** si tous les opérandes (y compris celui qui sera affecté de la valeur finale) sont de type entier.

   2) Pour évaluer la valeur d'une expression arithmétique, le processeur conserve les valeurs intermédiaires avec une précision maximale ; seul le résultat final sera, tronqué ou arrondi en fonction du format externe de la rubrique affectée.

   3) Les opérandes n'ont pas besoin d'être du même format interne ; les conversions de format sont faites automatiquement.

4.2. Valeurs arrondies

Le résultat d'un calcul peut comporter davantage de chiffres que de positions prévues au niveau de la rubrique affectée.

Rappelons que l'affectation d'une valeur numérique se fait conformément à la règle d'alignement numérique énoncée au chap. 2 paragraphe 3 ; s'il y a débordement à gauche la valeur sera tronquée ; s'il y a débordement à droite, la valeur sera tronquée ou arrondie selon que le programmeur aura précisé ou non la locution ROUNDED. Cette locution est à spécifier pour chaque instruction dont le résultat doit être arrondi.

L'arrondi est déterminé de la manière suivante :

- si le premier chiffre tronqué vaut de 0 à 4, les chiffres restants ne sont pas modifiés.

- si ce chiffre vaut de 5 à 9, on rajoute une unité au dernier chiffre restant.

  Exemple : Soit A PIC 99V99

  Le résultat 5.1772 donne sans arrondi 5.17

avec arrondi 5.18
Le résultat 5.1729 donne avec ou sans arrondi 5.17
Le résultat 5.195 donne sans arrondi 5.19
avec arrondi 5.20

Remarque : Dans l'évaluation d'une expression arithmétique, l'arrondi ne porte que sur le **résultat final**.

4.3. Expressions conditionnelles

Une expression conditionnelle est une expression dont le résultat est VRAI ou FAUX.

4.3.1. Expressions conditionnelles simples

4.3.1.1. Relation

Une relation permet d'exprimer la comparaison de deux objets ; ces objets sont des données constantes ou variables ; l'un des deux doit être obligatoirement une donnée variable.

Une relation s'écrit :

{ nd1 / constante 1 / exp-arith 1 } Opérateur de relation { nd2 / constante 2 / exp-arith 2 }

Opérateurs de relations	Signification	Négation
IS [NOT] GREATER THAN IS [NOT] >	strictement supérieur	inférieur ou égal
IS [NOT] LESS THAN IS [NOT] <	strictement inférieur	supérieur ou égal
IS [NOT] EQUAL TO IS [NOT] =	égal	différent

IS GREATER THAN OR EQUAL TO
IS >=

IS LESS THAN OR EQUAL TO
IS <=

Exemples

```
A  <  10
NOM    =  NOM-REF
```



```
NOM    =  "DUPONT"
A - B   =  RESUL + 15
```

On distingue deux algorithmes de comparaison :

1) Comparaison numérique :

Elle compare les valeurs algébriques de deux données de type numérique ; la longueur de ces données n'est pas significative. Une donnée non signée est considérée comme positive.

2) Comparaison non numérique :

Elle opère sur deux données de type non numérique ; par extension, l'une des deux peut être numérique à condition qu'elle soit entière et déclarée en mode DISPLAY. L'algorithme compare les deux données caractère par caractère, de gauche à droite, jusqu'à trouver une différence ou la fin des rubriques. Le résultat de la comparaison est déterminé comme suit :

- en cas de différence, c'est la donnée contenant le caractère de plus fort poids (cf. table du code ASCII) qui est considérée comme étant la plus grande
- sinon, les deux données ont des valeurs égales.

Si les deux données sont de longueurs différentes, la plus courte est considérée comme étant complétée à droite d'espaces jusqu'à concurrence de la plus longue.

Exemples

1) Soient
```
A    PIC    X(5).
B    PIC    X(5).
```

Valeur de A	Valeur de B	Résultat
TOTO␣	TOTO␣	Egalité
TOTO␣	TOT␣␣	A > B
TOTO␣	TO␣v␣	A > B

2) Soit
```
A    PIC    X(5).
C    PIC    X(3).
```

Valeur de A	Valeur de C	Résultat
TOT␣␣	TOT	Egalité
TOTO␣	TOT	A > C
TOTO␣	XY␣	A < C

4.3.1.2. Test de classe

Un test de classe permet d'exprimer le fait que la valeur d'une donnée est de classe numérique ou alphanumérique.

- Un test de classe numérique ne peut s'appliquer qu'à une rubrique élémentaire déclarée en USAGE DISPLAY.
- Un test de classe alphabétique ne peut pas s'appliquer à une rubrique de type numérique.

Il s'écrit :

nd IS [NOT] { NUMERIC / ALPHABETIC }

Remarque :

Pour effectuer un test de classe numérique, le processeur tient compte de la déclaration du signe au niveau du format de la rubrique ; ainsi une valeur non signée dans une rubrique signée donne un résultat faux ; il en est de même pour une valeur signée dans une rubrique non signée.

Exemples : Soit `A PIC X(5).`

- Le test `A ALPHABETIC` donne un résultat vrai si et seulement si les 5caractères sont des lettres ou des espaces.
- Le test `A NUMERIC` donne un résultat vrai si et seulement si les 5 caractères sont des chiffres.

4.3.1.3. Condition de signe

Elle détermine si la valeur algébrique d'une expression arithmétique est inférieure, supérieure ou égale à zéro.

Expression arithmétique IS [NOT] { NEGATIVE / POSITIVE / ZERO }

4.3.1.4. Condition de nom-condition

Elle s'exprime par le simple énoncé du nom-condition :

nom-condition

Elle est vraie si la valeur courante de la variable conditionnelle associée appartient (ou est égale) à celle attribuée au nom-condition.

4.3.2. Expressions conditionnelles complexes

Une condition simple `NEGATIVE` est une condition simple précédée de l'opérateur logique NOT. Elle a pour effet d'attribuer la valeur booléenne opposée à celle de la condition simple.

Ex. : `IF NOT A = B ….`

Une condition COMPOSEE résulte de la conjonction de conditions par les opérateurs AND ou OR :

condition { AND / OR } condition

où condition peut être :

- une condition simple
- une condition simple négative
- une condition composée
- une condition composée négative (i.e. une condition composée entre parenthèses et précédée de NOT)
- toute combinaison de ce qui précède

Ex : `A > B AND NOT (A = C) OR A NOT < D AND B = C`

Il est possible d'agréger une condition composée dans le cas où des conditions simples successives contiennent un même sujet et/ou un même opérateur logique, par l'omission du sujet ET de l'opérateur.

Ex. : `A > B AND NOT < C OR D` signifie
`((A > B) AND (A NOT < C)) OR (A NOT < D)`

4.4. Organisation de la division PROCEDURE

La structure de cette division est de l'une des deux formes ci-dessous :

a) `PROCEDURE DIVISION.`

nom de section `SECTION`.

nom de paragraphe. phrase ... ...

b) `PROCEDURE DIVISION.`

nom de paragraphe. phrase ... ...

Ceci signifie que :

- le découpage de la division en paragraphes ou en sections et paragraphes est **obligatoire**.
- le découpage doit être **complet** :
  - toute phrase appartient à un paragraphe explicitement déclaré.
  - si la forme (a) est choisie, tout paragraphe appartient à une section explicitement déclarée.

Exemple

```
PROCEDURE DIVISION.
S0 SECTION.
P1.
P2.
S1 SECTION.
P3
END PROGRAM.
```

4.5. ACCEPT

Permet d'initialiser une rubrique à l'aide d'une valeur particulière (date ou heure) ou d'une valeur lue dans le fichier d'entrée de l'utilisateur.

Format : `ACCEPT` nd [`FROM` { `DATE` | `DAY` | `TIME` | `DAY-OF-WEEK` }]

nd est une rubrique en mode DISPLAY d'au plus 71 caractères ; seules les positions correspondant à des caractères effectifs sont initialisées ; les autres restent inchangées ; il est donc vivement conseillé d'initialiser correctement la rubrique nd avant l'exécution de l'ordre ACCEPT.

De nombreuses variantes existent, suivant les implantations, permettant une gestion avancée de messages à l'écran.

Exemple :
```
01  IDENTITE.
    02  NOM PIC X(10) VALUE SPACE.
    02  PRENOM PIC X(10) VALUE SPACE.
    .
    .
ACCEPT IDENTITE
```

Si la valeur lue est DUPONT␣␣␣␣JULES, la rubrique IDENTITE contient après exécution de l'ordre

DUPONT␣␣␣␣JULES␣␣␣␣␣
espaces provenant de la clause VALUE

Les options `DATE` et `TIME` permettent de lire respectivement la date et l'heure enregistrées au niveau du système de l'ordinateur.

- Pour l'option `DATE`, nd doit être une rubrique élémentaire de 6 chiffres, non signée, `PIC 9(6) DISPLAY` ; la valeur obtenue est du format AAMMJJ (Année Mois Jour)
- `DAY` fournit la date au format AAJJJ au format PIC 9(5)
- Pour l'option `TIME`, nd doit être une rubrique élémentaire de 8 chiffres, non signée `PIC 9(8) DISPLAY` ; la valeur obtenue est du format hhmmsscc (heure minutes secondes 1/100è seconde)
- `DAY-OF-WEEK` est du format `PIC 9` et renvoie le rang du jour (1=lundi, 7=dimanche)

4.6. DISPLAY

Permet d'envoyer sur un fichier de sortie un faible volume d'informations construit par juxtaposition des valeurs des objets cités :

`DISPLAY` { nd1 | lit1 } [{ nd2 | lit2 }] .... [`WITH NO ADVANCING`]

L'impression n'incluant aucune conversion de format, les valeurs numériques doivent être codées en ASCII.

Ex. : `DISPLAY "NOM : " NM " PRENOM : " PRNM.`

Même remarque que pour `ACCEPT`

4.7. Verbes arithmétiques

4.7.1. Généralités

- Dans toutes les instructions de ce paragraphe, les opérandes identifiés par nd désignent des rubriques numériques ; cependant, les zones de réception des calculs spécifiées après `GIVING`, n'intervenant pas dans les calculs, peuvent être de type numérique édité.

- Tous les littéraux "lit" sont de type numérique
- La spécification d'un arrondi se fait par la locution ROUNDED.
- La phrase ON SIZE ERROR permet de spécifier un traitement à accomplir en cas de dépassement de capacité lors des calculs ou de division par 0. Les rubriques de réception associées à cette phrase ne seront pas modifiées en cas d'erreur. La phrase NOT ON SIZE ERROR en est le pendant.
- La locution CORRESPONDING (ou CORR) peut être utilisée dans le cas où les opérandes nd1 et nd2 désignent des rubriques de GROUPES. Une rubrique de nd1 et une rubrique de nd2 correspondent si :

  elles sont désignées par le même nom (autre que FILLER) et ont les mêmes qualificateurs jusqu'à nd1 et nd2 non inclus
  ce sont des données élémentaires
  nd1 et nd2 ne doivent pas contenir de niveau 66, 77, 88
  une donnée subordonnée à nd1 et nd2 et contenant une clause REDEFINES, RENAMES, OCCURS ou INDEX est ignorée de même que les données qui lui sont subordonnées.

Sous ces conditions, l'opération portera individuellement sur chaque couple de données correspondantes.

4.7.2. Addition

Format 1 : ADD { nd1 nd2 / lit1 lit2 } TO nd-m [ROUNDED] [nd-n [ROUNDED]]…

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]
[END-ADD]

Format 2 : ADD { nd1 / lit1 } … TO { nd2 / lit2 } GIVING nd-m [ROUNDED] …

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]

[END-ADD]

Format 3 : ADD { CORRESPONDING / CORR } nd1 TO nd2 [ROUNDED]

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]
[END-ADD]

Exemple : `ADD N M 3 TO P` P=N + M + 3 + P
`ADD N M 3 GIVING P` P=N + M + 3

4.7.3. Soustraction

Format 1 : SUBTRACT { nd1 / lit1 } ... FROM {nd-m [ROUNDED]} …

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]
[END-SUBTRACT]

Format 2 : SUBTRACT { nd1 / lit1 } ... FROM { nd2 / lit-m }
GIVING nd-n [ROUNDED]

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]
[END-SUBTRACT]

Format 3 : SUBTRACT { CORRESPONDING / CORR } nd-1 FROM nd-2 [ROUNDED]

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]
[END-SUBTRACT]

4.7.4. Multiplication

Format 1 : MULTIPLY { nd1 / lit1 } BY {nd-2 [ROUNDED]} ...

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]
[END-MULTIPLY]

Format 2 : MULTIPLY { nd1 / lit1 } BY { nd2 / lit2 } GIVING {nd-3 [ROUNDED]} …

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]
[END-MULTIPLY]

4.7.5. Division

Format 1 : DIVIDE { nd1 / lit1 } INTO {nd-2 [ROUNDED] } ...

[ON SIZE ERROR ph-impér.]
[NOT ON SIZE ERROR ph-impér.]

[END-DIVIDE]

Format 2 :

```
DIVIDE { nd1 | lit1 } INTO { nd2 | lit2 } GIVING {nd-3 [ROUNDED]} …
       [ON SIZE ERROR ph-impér.]
       [NOT ON SIZE ERROR ph-impér.]
[END-DIVIDE]
```

Format 3 :

```
DIVIDE { nd1 | lit1 } BY { nd2 | lit2 } GIVING {nd-3 [ROUNDED]} …
       [ON SIZE ERROR ph-impér.]
       [NOT ON SIZE ERROR ph-impér.]
[END-DIVIDE]
```

Format 4 :

```
DIVIDE { nd1 | lit1 } INTO { nd2 | lit2 } GIVING nd-3 [ROUNDED]
       REMAINDER nd-4

       [ON SIZE ERROR ph-impér.]
       [NOT ON SIZE ERROR ph-impér.]
[END-DIVIDE]
```

Format 5 :

```
DIVIDE { nd1 | lit1 } BY { nd2 | lit2 } GIVING nd-3 [ROUNDED]
       REMAINDER nd-4

       [ON SIZE ERROR ph-impér.]
       [NOT ON SIZE ERROR ph-impér.]
[END-DIVIDE]
```

Remarques :
1. L'option BY implique GIVING
2. L'option REMAINDER permet de recueillir le reste de la division dans nd-4 ; elle interdit l'usage de plusieurs zones de réception.

4.7.6. Calcul

La valeur d'une expression arithmétique complexe peut être obtenue par l'instruction :

```
COMPUTE {nd-1 [ROUNDED]} ... = expression-arithmétique
        [ON SIZE ERROR ph-impér.]
        [NOT ON SIZE ERROR ph-impér.]
[END-COMPUTE].
```

Ex. :
```
COMPUTE TOTAL ROUNDED = PUHT * QTE + TRSP * (1 - TX)
```


4.8. MOVE

L'instruction MOVE permet de recopier dans une (ou plusieurs)rubrique(s) dites réceptrices, le contenu d'une (ou plusieurs) rubrique(s) dites émettrices.

Format 1 :
```
MOVE { nd1 | lit1 } TO {nd-2} ...
```

Format 2 :
```
MOVE { CORRESPONDING | CORR } nd1 TO nd2
```

- L'opération inclut si nécessaire les conversions de format interne.

- Dans le format 2, au moins une des données du couple de données concerné par l'opération doit être élémentaire.

  Ex. :
```
05 DT-AMER.
      10 AA PIC 99.
       10 MM PIC 99.
       10 JJ PIC 99.

05 DT-JOUR.
      10 JJ PIC 99.
       10 FILLER PIC X VALUE "/".
      10 MM PIC 99.
       10 FILLER PIC X VALUE "/".
      10 AA PIC 99.

   MOVE CORR DT-AMER TO DT-JOUR.
```

- Il y a deux types de transferts :
  - transfert **numérique**
  - transfert **non numérique**

  qui respectent les règles d'alignement décrites au chapitre 2 § 3 ; il ne peut y avoir de conversion de format interne dans un transfert non numérique.

- Le type du transfert est déterminé par le type de l'émetteur et du récepteur ; le tableau ci-dessous résume les transferts possibles

	Zone de Réception					
Zone d'Emission	Num. entier	Num.fract.	Alphabét.	Alphanum.	Alph. Edité	Num. Edité
Num. entier	N	N	I	X	X	N
Num. fractionnaire	N	N	I	I	I	N
Alphabétique	I	I	X	X	X	I
Alphanumérique	X*	X*	X*	X	X	X*
Alphanum. Edité	I	I	X	X	X	I
Num. Edité	I	I	I	I	X	X

X : transfert non numérique
X* : transfertautorisé seulement si tous les caractères émis sont autorisés dans la zone de réception
N : transfert numérique
I : transfert interdit

Une rubrique structurée se comporte comme une rubrique élémentaire alphanumérique.

Transfert du signe :

- Emetteur signé/récepteur non signé : le récepteur contient la valeur absolue de l'émetteur
- Emetteur non signé/récepteur signé : la valeur de l'émetteur est considérée comme positive
- Emetteur numérique entier/récepteur alphanumérique : le signe n'est pas transféré.

4.9. IF

Elle permet d'exprimer une alternative entre deux traitements.

Format : IF expression conditionnelle THEN { phrase-1 / NEXT SENTENCE }

{ ELSE / OTHERWISE } { phrase-2 / NEXT SENTENCE }

[END-IF]

Exemple
```
IF A NOT NUMERIC OR A NOT > 0
                DISPLAY "ERREUR***"
        ELSE ADD A TO SOMME.
```

- Si l'expression conditionnelle donne un résultat vrai, on effectue le traitement décrit dans "phrase-1" sinon on effectue celui qui est décrit dans "phrase-2".

Remarques :

- ne pas oublier le point à la fin de "phrase-2" ou END-IF. Le mot clé ELSE indique la fin de "phrase-1".
- phrase-1 et/ou phrase-2 se composent d'instructions impératives et/ou d'une instruction IF.
- IF et ELSE s'emploient par paires ; tout mot ELSE est à associer avec la première condition qui le précède et qui n'est pas déjà reliée à un mot ELSE.

Exemple :
```
IF ....
    IF ....
    ELSE ....
            ELSE   IF ....
 ELSE .... .
```


4.10. EVALUATE

En COBOL 85, l'instruction EVALUATE permet de généraliser la notion de structure alternative, et représente une implémentation de la liste de cas.

EVALUATE { nd-1 / lit-1 / expression-1 / TRUE / FALSE } [ALSO { nd-2 / lit-2 / expression-2 / TRUE / FALSE }] ...

WHEN { ANY / condition-1 / TRUE / FALSE / [NOT] { nd-3 / lit-3 / expression-3 } [{ THROUGH / THRU } { nd-4 / lit-4 / expression-4 }] }

ALSO { ANY / condition-2 / TRUE / FALSE / [NOT] { nd-5 / lit-5 / expression-5 } [{ THROUGH / THRU } { nd-6 / lit-6 / expression-6 }] }

phrase-impérative-1 …

[WHEN OTHER phrase-impérative-2]

[END-EVALUATE]

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

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

Le processus est répété en cas d'inégalité pour trouver le premier WHEN satifaisant l'ensemble des conditions. Si aucune phrase suivant WHEN n'a été sélectionnée, et si WHEN OTHER a été spécifié, la phrase-impérative-2 est exécutée, et l'instruction EVALUATE est terminée.
Il est possible de préciser CONTINUE en tant que phrase-impérative-1 : aucune action n'est effectuée et l'on passe à l'instruction suivant EVALUATE.

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

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

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

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

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

A	B
1	1
1	0
0	1
0	0

Table de vérité XOR.

Exemple 4
```
EVALUATE TRUE ALSO TRUE
     WHEN A = 1 ALSO B = 1 PERFORM TRAITEMENT-1
     WHEN ANY ALSO B = 2         PERFORM TRAITEMENT-2
     WHEN OTHER            PERFORM TRAITEMENT-3
END-EVALUATE.
```

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

4.11. PERFORM

Elle traduit l'action de "faire" un traitement 0, 1 ou plusieurs fois

Format 1 : `PERFORM` nt1 { `THROUGH` | `THRU` } nt2
[ph-impér. `END-PERFORM`]

Format 2 : `PERFORM` nt1 { `THROUGH` | `THRU` } nt2 { nd | entier-1 } `TIMES`
[ph-impér. `END-PERFORM`]

Format 3 : `PERFORM` nt1 { `THRU` | `THROUGH` } nt2 [`WITH TEST` { `BEFORE` | `AFTER` }]
`UNTIL` condition-1
[ph-impér. `END-PERFORM`]

- Terminologie :
  - nt désigne un paragraphe ou une section
  - nd doit référencer un entier non signé

- L'instruction `PERFORM` permet d'exécuter un bloc de traitement décrit sous forme :
  - d'un paragraphe
  - d'une section
  - d'un ensemble de paragraphes et/ou de sections contigus dont le premier s'appelle nt1 et le dernier nt2.

- L'exécution du bloc de traitement commence à la première instruction du bloc et se termine obligatoirement à la dernière instruction du bloc.

- L'instruction `PERFORM` peut être écrite avant ou après le traitement qu'elle fait exécuter.

- Dans les formats 3 et 4 (cf ci-après), **par défaut** ou si l'on précise `TEST BEFORE` la condition est évaluée **avant** d'exécuter le bloc ; avec `TEST AFTER`, après exécution du bloc.

Exemple
```
S0 SECTION.
P0.
P1.

S1 SECTION.
P2.
P3.

S2 SECTION.


P4. PERFORM S0                (1)
   PERFORM P0 THRU P1         (2)
   PERFORM P1 THRU P2         (3)
   PERFORM S0 THRU S1         (4)
   PERFORM P0 THRU P3         (5)
```

Les lignes (1) et (2) sont équivalentes ; il en est de même des lignes (4) et (5).

- Le format 2 permet de traduire un traitement itératif, le nombre de répétitions étant connu par avance.

Exemples :
```
PERFORM S0   3   TIMES
PERFORM P1  N1   TIMES
                Où  N1 PIC  99.
```

A noter que seule la valeur de `N1` définie **au moment** de l'exécution de l'ordre `PERFORM` est prise en compte ; toute modification ultérieure de `N1`, au cours des itérations, est sans effet sur le nombre de répétitions effectuées.

- Le format 3 permet de traduire un traitement itératif à répéter jusqu'à ce qu'une condition soit satisfaite.

Exemple : `PERFORM SI UNTIL J > 50`

Le cycle des opérations standard est le suivant :

1) Evaluation de l'expression conditionnelle (sauf si `TEST AFTER`)
2) Si le résultat de l'expression conditionnelle est vrai
   Arrêt des itérations sinon exécuter S1 reprendre au point 1)

Remarque:

Si au départ J contient une valeur supérieure à 50, il n'y a pas d'exécution de S1.

♦ Instructions PERFORM imbriquées :

Le bloc de traitement exécuté à partir d'un ordre PERFORM peut contenir des ordres PERFORM à condition que ceux-ci réfèrent des blocs de traitement **totalement inclus ou disjoints** du bloc d'origine.

Exemple :
```
PERFORM  P0 THRU P4
P0.
P1.
P2.
P3.    PERFORM P1 THRU P2 2 TIMES        (1)
P4.    PERFORM P5 UNTIL NOM = SPACE
P5.
```

Le bloc P1-P2 est inclus dans le bloc P0-24
Le bloc P5 est disjoint du bloc P0-P4.
Il est interdit de remplacer par exemple la ligne (1) par PERFORM P4 THRU P5

Instruction PERFORM avec variable de comptage

Format 4 :
```
PERFORM  nt1 { THRU | THROUGH }  nt2 [ WITH TEST { BEFORE | AFTER } ]
    VARYING { nd-2 | index-1 }  FROM { nd-3 | index-2 | ent-1 }
        BY { nd-4 | ent-2 }  UNTIL  condition-1
    [ AFTER { nd-5 | index-3 }  FROM { nd-6 | index-4 | ent-3 }
        BY { nd-7 | ent-4 }  UNTIL  condition-2 ] ...
[ph-impér. END-PERFORM]
```

Jusqu'à 6 phrases AFTER

Il inclut la gestion des indices ou des index jusqu'à 3 (7) niveaux d'imbrication.

Exemple : Soit la procédure CUMUL-NOTE permettant de cumuler une note (appartenant à une table de dimension 2) à une variable de cumul partiel (SOMME). E et N sont deux indices indiquant respectivement le rang de l'élève et le rang de la note.

```
        MOVE 0 TO SOMME
        PERFORM CUMUL-NOTES VARYING E FROM 1 BY 1
                            UNTIL E > 25
                            AFTER N FROM 1 BY 1
                               UNTIL N > 10.

CUMUL-NOTES.
        ADD NOTE-ELEVE (E,N) TO SOMME.
```

4.12. EXIT

Format 1 : `EXIT .`

Format 2 : `EXIT PROGRAM.`

Cette instruction doit être la seule instruction de la seule phrase du paragraphe où elle est écrite.

Son rôle : - EXIT : définir un paragraphe vide
- EXIT PROGRAM : définir un paragraphe vide et déclencher le retour au programme appelant; cette option ne peut être utilisée que dans un sous-programme.

Exemple :

Dans un tableau INDIVIDUS de dimension 1 comportant 25 noms de personnes, déterminer le rang de l'individu DUPONT.

```
        PERFORM RECHERCHE VARYING E FROM 1 BY 1
                          UNTIL NOM (E) = "DUPONT"
                                       OR E > 25.
RECHERCHE. EXIT.
```

A la fin de l'itération, E contient le rang de DUPONT ou la valeur 26 si cette personne ne figure pas dans la table.

4.13. STOP

Permet d'arrêter temporairement ou définitivement l'exécution d'un programme.

Format : `STOP { RUN | lit }`

- STOP RUN provoque l'arrêt définitif du programme ;
- Dans le format 2, le littéral est affiché sur le terminal de l'opérateur et le programme est suspendu. A ce point, l'utilisateur peut relancer le programme par `Enter`, ou l'abandonner.

4.14 GO

Permet d'effectuer dans un programme une rupture de séquence inconditionnelle. L'usage de cette instruction doit être réservé exclusivement à la traduction de certaines structures algorithmiques bien précises.

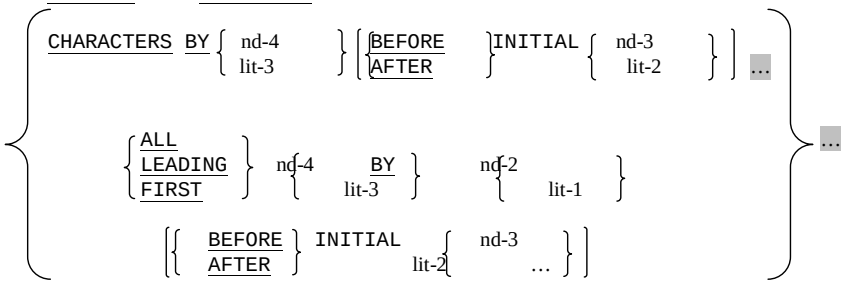
Format 1 : `GO TO` nt
Format 2 : `GO TO` nt1 [nt2] ..., ntn `DEPENDING ON` nd

Le deuxième format (aiguillage) permet de traduire un choix entre n traitements en fonction de la valeur de nd; nd réfère une rubrique élémentaire à valeur entière comprise entre 1 et n ; si nd vaut k, il y a branchement au $k^{ième}$ traitement de la liste ; si k # [1,n], l'instruction est sans effet.

4.15 INSPECT

Permet, à l'intérieur d'une rubrique de type non numérique, de remplacer (Format 1) les occurrences d'un ou plusieurs caractères, ou de les compter (Format2).

Format 1



- nd1 réfère la rubrique étudiée élémentaire ou de groupe , en mode `DISPLAY` obligatoirement.
- nd2 à nd4 doivent être des rubriques élémentaires.
- Algorithme de remplacement : L'algorithme examine le contenu de la rubrique de gauche à droite ; il y a deux possibilités :

a) Option `CHARACTERS` : on remplace chaque caractère rencontré par la valeur précisée nd2 ou lit1. Cette clause ne peut apparaître qu'une fois.

b) Autres options : on remplace chaque sous-chaîne de valeur nd4 ou lit3 par la valeur précisée par nd2 ou lit1 ; la sous-chaîne de remplacement doit avoir la même longueur que la sous-chaîne d'origine. Il est possible de préciser plusieurs options de ce type permettant de procéder au remplacement des occurrences de plusieurs sous-chaînes distinctes.

Exemples
```
INSPECT ZONE REPLACING CHARACTERS BY "*"
```
Tous les caractères de ZONE sont remplacés par des *
```
INSPECT ZONE REPLACING ALL "ABC" BY "XXX"
                       ALL "CDE" BY "YYY"
```
Valeur d'origine de ZONE : B C D E Z A B C D E A B C C D E
Valeur finale de ZONE : B Y Y Y Z X X X D E X X X Y Y Y
Les opérations sont traitées suivant l'ordre d'écriture ; ainsi la 2e occurrence de CDE n'est pas prise en compte, la lettre C ayant déjà été traitée par le remplacement de la 1ère occurrence de ABC.

Les mots clés `ALL`, `LEADING` et `FIRST` permettent de sélectionner ou non certaines occurrences de la sous-chaîne à remplacer :

- `ALL` : toutes les sous-chaînes ayant la valeur nd4 ou lit3 sont traitées

- `LEADING` : **toutes** les occurrences **contiguës** de la sous-chaîne de valeur nd4 ou lit3 sont traitées à condition que la première se trouve au début de la zone à traiter.

Exemple : `INSPECT ZONE REPLACING LEADING "AB" BY "ZZ"`
Valeur d'origine de ZONE : XYABABXAB
Valeur finale : identique

si on rajoute AFTER "Y", valeur finale : XYZZZZXAB

♦ FIRST : seule **la première** sous-chaîne de valeur nd4 ou lit3 est traitée.

Délimitation de la zone à étudier :

L'option AFTER/BEFORE, valable pour les deux algorithmes de remplacement, permet de faire débuter (respect. d'arrêter) le traitement à la rencontre de la première occurrence de la valeur spécifiée par nd3 ou lit2 (obligatoirement de longueur 1 caractère pour l'algorithme (a)).

Format 2

```
INSPECT   nd-1 TALLYING

nd-2 FOR  { ALL        }  { nd-3 }
          { LEADING    }  { lit-1 }
          { CHARACTERS }

[ { BEFORE } INITIAL { nd-4 } ]
[ { AFTER  }         { lit-2 } ]   …
```

Avec TALLYING, nd-2 doit référencer une rubrique numérique, précédemment initialisée. Cette rubrique est incrémentée de un pour chaque occurence de sous-chaîne (ou chaque caractère) suivant des modalités analogues à celles de REPLACING.

Format 3

Résulte de la juxtaposition du format 2 (TALLYING) suivi du format 1 (REPLACING).

Exemple :
```
INSPECT RUB TALLYING COMPTEUR FOR ALL "L" REPLACING
   LEADING  "A" BY "E"  AFTER INITIAL "L".
```

Format 4

Il permet de remplacer, dans nd1, chaque caractère de nd2 par le caractère de même rang de nd3 (équivaut à une série de INSPECT REPLACING pour chaque caractère).

```
INSPECT   nd-1 CONVERTING

{ nd-2 } TO { nd-3 }
{ lit-1 }   { lit-2 }

[ { BEFORE } INITIAL { nd-4 } ]
[ { AFTER  }         { lit-3 } ]   …
```

Exemple :
```
77 ZON PIC X(10)  VALUE "ABCAEFACA".
INSPECT ZON CONVERTING "ABC" TO "XYZ" BEFORE "F".
```
ZON contiendra "XYZXEFACA"

4.16.1. Concaténation

L'instruction STRING permet la juxtaposition des contenus partiels ou complets de 2 ou plusieurs rubriques en une seule.

```
STRING { émetteur-1 ... DELIMITED BY { délimiteur-1 } } …
       {                             { SIZE        } }

  INTO récepteur [WITH POINTER pointeur]

[ON OVERFLOW phrase-impérative 1]
    [NOT ON OVERFLOW phrase-impérative 2]
    [END-STRING]
```

♦ Chaque rubrique d'émission doit être d'usage DISPLAY ; si elle est numérique, elle doit être entière. Ce peut être aussi un littéral non numérique (ou une constante figurative, considérée comme un seul caractère).

♦ La rubrique de réception doit être **alphanumérique** élémentaire.

♦ La clause DELIMITED est **obligatoire** ; les délimiteurs (1 ou plusieurs caractères) suivent les règles de syntaxe des émetteurs ; un délimiteur n'est PAS TRANSMIS dans la zone de réception. Si SIZE est spécifié, le contenu complet de l'émetteur (ou des émetteurs) précédant DELIMITED est transféré dans le récepteur.

♦ Le pointeur est une rubrique entière de longueur suffisante pour contenir la taille + 1 du récepteur; il contient à chaque instant le rang du prochain caractère à recevoir ; s'il est spécifié, il doit être initialisé à une valeur supérieure ou égale à 1(défaut = 1).

Aucun des caractères du récepteur, non concernés par le transfert (en début ou en fin de zone) n'est modifié par l'instruction.

♦ La locution OVERFLOW sera validée si au cours des transferts, le nombre de caractères émis excède la longueur du récepteur (à défaut, le pg se poursuit en séquence).

Remarque : On peut considérer que les transferts se font caractère par caractère, en commençant, pour chaque émetteur pris en séquence, par le caractère le plus à gauche et jusqu'à atteindre :
- soit la fin de l'émetteur,
- soit le(s) caractère(s) du délimiteur associé (jamais transmis).

Exemples :
```
STRING JJ "/" MM "/" AA DELIMITED BY SIZE INTO DATE-COM.
STRING NOM DELIMITED BY "   " SPACE DELIMITED SIZE PRENOM
DELIMITED SPACE INTO IDENTITE.
```

4.16.2. Séparation

L'instruction UNSTRING provoque la séparation de données contigües d'une rubrique émettrice et leur transfert dans des rubriques réceptrices multiples.

```
UNSTRING émetteur    [DELIMITED  BY [ALL] délimiteur-1
                [OR [ALL]  délimiteur-2] ...]

INTO      {récepteur-1 [DELIMITER IN délimiteur-3] [COUNT IN compteur-1]}
  [WITH POINTER pointeur] [TALLYING IN compteur-3]
    [ON OVERFLOW phrase-impérative 1]
       [NOT ON OVERFLOW  phrase-impérative 2]
       [END-UNSTRING]
```

- L'émetteur et les délimiteurs doivent être des rubriques alphanumériques ; les délimiteurs peuvent àtre des littéraux non numériques.
- Les récepteurs peuvent être alphabétiques, alphanumériques ou numériques (d'usage DISPLAY).
- Les compteurs et pointeurs doivent être numériques entiers.
- Les locutions DELIMITER et COUNT ne peuvent figurer que si DELIMITED BY est spécifié.

Règles d'application

- délimiteur-3 et -4 sont les rubriques de réception des délimiteurs, et accueillent ces derniers selon les règles de transfert non-numérique. Si la condition de délimitation est la fin de l'émetteur (i.e. délimiteur non rencontré) ces rubriques sont à blanc.
- Chaque compteur contient le nombre de caractères transmis dans le récepteur associé, délimiteur exclu.
- Le pointeur indique le rang du premier caractère de l'émetteur participant à l'opération (défaut = 1), puis est incrémenté au fur et à mesure.
- Le compteur-3 enregistre le nombre de rubriques de réception mises en jeu par l'opération (initialisation à charge du programmeur).
- Lorsque ALL est spécifié, plusieurs apparitions contigües du délimiteur associé sont traitées comme une seule occurrence ; sinon, deux apparitions contigües entraînent une mise à blanc ou à zéro du récepteur associé, selon sa définition.
- Lorsqu'un délimiteur contient plusieurs caractères, ils doivent être présents en totalité, et dans l'ordre, dans l'émetteur pour être reconnus comme délimiteur.
- Si plusieurs délimiteurs sont spécifiés dans la locution DELIMITED BY, reliés par OR, l'émetteur est analysé pour chacun d'eux dans l'ordre donné, jusqu'à correspondance.

- Si DELIMITED BY n'est pas spécifié, le transfert s'effectue jusqu'a remplir le récepteur courant.
- Après chaque extraction d'une sous-chaîne de l'émetteur, l'analyse reprend au caractère suivant le délimiteur.
- L'analyse se poursuit jusqu'à la fin de l'émetteur ou épuisement de récepteurs.
- A la fin de l'opération, le pointeur est égal à sa valeur initiale plus le nombre de caractères examinés dans l'émetteur.

La locution ON OVERFLOW sera validée :

- si le pointeur est initialisé à une valeur inférieure à 1 ou supérieure à la taille de l'émetteur
- si toutes les zones de réception ont été utilisées et il reste des caractères à examiner dans l'émetteur.
- Si une condition de débordement apparaît et OVERFLOW n'est pas spécifié, l'exécution se poursuit en séquence.
- Si un délimiteur ou un récepteur est indicé ou indexé, l'indice ou l'index est évalué immédiatement avant le transfert de donnée dans la rubrique correspondante.

Exemples :
```
UNSTRING DATE-COM DELIMITED BY "/" INTO JJ MM AA

UNSTRING IDENTITE DELIMITED BY SPACE INTO NOM
COUNT IN CN PRENOM COUNT IN CP POINTER CT
```

4.17. INITIALIZE

En ANS85, cette instruction permet d'initialiser, certains **types** de données.

```
INITIALIZE   { nd-1} …

[ REPLACING { ALPHABETIC
              ALPHANUMERIC
              NUMERIC
              ALPHANUMERIC-EDITED
              NUMERIC-EDITED }   DATA BY { nd-2
                                           lit-1 } … ]
```

- nd-1 est le récepteur, à initialiser ; nd-2 ou lit-1 sont les émetteurs servant à l'initialisation.
- nd-1 ne peut être un index, ni contenir de clause RENAMES ou DEPENDING ON

- en l’absence de clause `REPLACING`, les données élémentaires des récepteurs sont initialisée à **b** pour les types non-numériques et à **0** pour les types numériques.

- Si `REPLACING` est spécifié, l'instruction opère comme un ensemble de `MOVE` entre l'émetteur et le(s) récepteur(s), au niveau élémentaire s'il s'agit de groupes. L'opération n'est effectuée que pour les données correspondant au type cité. Un type ne peut apparaître qu'une seule fois dans un `INITIALIZE`.

Exemple
```
01  COMPTE.
    02  NUM    PIC 9(5).
    02  LIBEL  PIC X(20).
    02  SOLDE  PIC S9(6)V99  PACKED-DECIMAL.
    …

INITIALIZE COMPTE.

INITIALIZE COMPTE  REPLACING ALPHANUMERIC BY ALL "X".
```

LE TRAITEMENT DES TABLES

1 Principes

Une table (ou tableau) est une collection ordonnée d'objets du même type ; tout objet de la table peut être sélectionné indépendamment des autres en citant ses "coordonnées".

Exemple : Table des intitulés des jours de la semaine LUNDI MARDI MERCREDI JEUDI VENDREDI SAMEDI DIMANCHE. L'intitulé du 3e jour est MERCREDI.

Une table est de **dimension** n, n > 0 ; l'exemple précédent montre une table de 7 chaînes de caractères et de dimension 1.

- Le langage COBOL comporte des clauses permettant de décrire des tables de **dimension ≤3 (7)** ; les objets d'une table peuvent être de type élémentaire ou structuré.

Exemple : Tarif des consommations ; c'est une table de dimension 1, composée d'objets du type "consommation" défini par :
. nom-consommation (chaîne de 15 caractères)
. prix-consommation (réel)

- Le langage COBOL inclut les manipulation de tables, c'est-à-dire qu'il permet de sélectionner des objets appartenant à des tables. Deux techniques peuvent être utilisées :
  - L'indiçage : consiste à préciser dans l'ordre les valeurs des différents sélecteurs sous forme d'indices; les indices étant du type entier, leurs valeurs peuvent être calculées à l'aide des instructions arithmétiques classiques.
  - L'indexation consiste à préciser dans l'ordre les valeurs des différents sélecteurs sous forme de pointeurs appelés index ; la valeur d'un index correspond en fait à une forme d'adressage de la mémoire centrale ; elle ne peut donc être calculée par les instructions arithmétiques classiques et nécessite l'emploi d'instructions spécifiques. Par ailleurs, un index est rattaché à un sélecteur d'une table et ne peut être utilisé à d'autres fins.

On peut cependant définir une variable index en `DATA DIVISION`, par la clause :

```
[USAGE IS]  INDEX
```

Toute autre clause est interdite.

Ex. : `77 IND1 USAGE INDEX.`

Ceci peut être utile pour sauvegarder des valeurs index.

La **sélection** d'un objet d'une table s'exprime par : nom de l'objet (liste d'expressions d'indices ou d'index). Le nombre d'expressions d'indice ou d'index correspond à la dimension de la table.

Une **expression d'indices ou d'index** est de la forme :

nd-1 ({ entier-1 / nd-2 [{±}entier-2] / index-1 [{±}entier-3] } …)

Exemples :

```
A (3)
B (I, 5)
B (I, J)          ou I, J est du type entier
B (IND + 1, 5)    IND est du type index
```

Remarques :

1) Les virgules séparant les expressions d'indices ou d'index sont facultatives et peuvent être remplacées par des espaces.
2) Un objet d'une table étant nécessairement à occurrences multiples, il est indispensable de préciser l'occurrence de l'objet sur laquelle porte le traitement ; cette sélection doit être répétée pour chaque instruction faisant référence à cet objet.

2 Définition d'une table - OCCURS

Format 1 : `OCCURS` ent-2 `TIMES`

{ `ASCENDING` / `DESCENDING` } `KEY IS` nd2 [nd3] ...

[`INDEXED BY` index1 [index2] ...] ...

Format 2 : `OCCURS` ent-1 `TO` ent-2 `TIMES DEPENDING ON` nd1

{ `ASCENDING` / `DESCENDING` } `KEY IS` nd2 [nd3] ...

[`INDEXED BY` index1 [index2] ...] ...

- ent-1 et ent-2 sont des entiers désignant respectivement le nombre minimum et le nombre maximum d'objets dans la table (ent-1<ent-2)
- nd1 doit décrire un entier positif ; elle contient le nombre exact d'occurrences de l'objet dans la table courante (nd1 Ú 1) et sera placée dans la partie fixe de l'article ou hors de celui-ci.
- nd2, nd3, ... doivent être des rubriques subordonnées à celle contenant OCCURS ; cependant, nd2 peut désigner cette rubrique (si elle est seule) ; aucune des rubriques citées ne peut contenir OCCURS

- `INDEXED` est obligatoire si l'objet (ou ses rubriques subordonnées) doit être référencé par indexation ; index1, index2, ... ne peuvent àtre définis ailleurs dans le programme.

- une rubrique contenant le format 2 ne peut être suivie dans l'enregistrement que par des rubriques subordonnées.

- `OCCURS` ne peut être spécifié aux niveaux `01, 77, 66` ou `88`

- nd1 ne peut pas être subordonnée à la rubrique sujet de `OCCURS`

- La locution `DEPENDING`, autorisant la gestion de tables de longueur variable (i.e. ayant un nombre variable d'éléments) permet d'optimiser l'occupation des supports externes, et de limiter les recherches en table quand celle-ci n'est pas totalement remplie dans tous les articles.

Exemple :

```
01 ABONNE.
     05 NOM PIC X(20).
     05 NB-OUV PIC 99.
     05 OUVRAGE OCCURS 1 TO 20 DEPENDING ON NB-OUV.
          10 CODOUV PIC 9(5).
          10 DATEMP PIC 9(6).
```

La locution `ASCENDING` (ou `DESCENDING`) `KEY` permet de préciser les rubriques de l'entrée servant de critères de classement dans le cas d'une table ordonnée. Elle est indispensable pour une recherche dichotomique (`SEARCH ALL`). L'ordre de classement est déterminé selon les règles de comparaison classique ; les critères sont à citer dans l'ordre d'importance décroissante (i.e. majeur en tête)

Exemple :

Table des élèves d'une classe, chaque élève ayant 10 résultats sous forme (note, coefficient)

```
01 TABLE-ELEVES.
     02 ELEVE OCCURS 25.
          03 NOM-ELEVE PIC X(20).
          03 RESULTAT-ELEVE OCCURS 10.
               04 NOTE-ELEVE PIC 99V9.
               04 COEFF-ELEVE PIC 9.
```

Cette description nous donne :

- au niveau 2, une table de 25 "élèves"
- au niveau 3, une table de dimension 2, de 25 fois 10 résultats

Ainsi :
`ELEVE (3)` réfère l'ensemble des informations concernant le 3ème élève.
`NOM-ELEVE (3)` désigne le nom du 3ème élève.
`RESULTAT-ELEVE (3, 6)` désigne le 6ème résultat du 3ème élève
`COEFF-ELEVE (3, 6)` désigne le coefficient du 6ème résultat du 3ème élève.

Utilisation conjointe de la clause OCCURS et des autres clauses :

- Aucune rubrique rattachée à un objet appartenant à une table ne peut être initialisée par une clause `VALUE` en ANS74. En ANS85, chaque occurrence est initialisée par la valeur spécifiée.

- La clause `REDEFINES` est autorisée :
  - pour redéfinir la structure d'un élément d'une table.
  - lorsque la table est incluse dans la redéfinition.

Il est par contre **interdit de redéfinir** une rubrique dont la structure comporte une table.

Exemples :

1)
```
01 A.
     02 TABLE1.
          03 ELEMENT PIC X OCCURS 10.
     02 B REDEFINES TABLE1 PIC X(10).
```
est **interdit**

2)
```
01 A.
     02 B PIC X(10).
     02 TABLE2 REDEFINES B.
          03 ELEMENT PIC X OCCURS 10.
```
est autorisé

3)
```
01 A.
     02 ELEMENT OCCURS 10.
          03 A1 PIC XX.
          03 A2 REDEFINES A1.
               04 E1 PIC 9.
               04 E2 PIC 9.
```
est autorisé

3. SET

Elle permet d'initialiser, d'incrémenter ou de décrémenter une variable index.

Format 1 : `SET` { nd-1 [nd-2]... / index-1 [index-2]... } `TO` { nd-3 / index-3 / ent-1 }

Format 2 : `SET` index-4 [index-5]... { `UP BY` / `DOWN BY` } { nd-4 / ent-2 }

- L'instruction SET...TO permet d'initialiser une suite d'index à une valeur exprimée sous forme d'entier (nd-3 ou ent-1) ou d'index (index-3) ; inversement, elle permet d'initialiser une suite d'indices (nd-1, nd-2,...) à une valeur exprimée sous forme d'index (index-3).

Attention : Il est interdit d'initialiser un indice, par l'instruction SET, à une valeur exprimée sous forme d'un entier ; une telle affectation s'exprime par une instruction MOVE (cf. chap. 3 §4.8)

Exemple : Si I et J sont du type entier et IND du type index, on peut écrire SET I J TO IND : initialisation des indices I et J au rang correspondant à la valeur de l'index.
IND SET IND TO 3 : positionnement de l'index IND sur la 3e occurence dans la table à laquelle il est rattaché.

- L'instruction SET...UP BY sert à incrémenter un ou plusieurs index d'un déplacement correspondant à n occurences, n étant exprimé par nd-4 ou ent-2.

  Exemple: SET IND UP BY 2

  L'instruction SET...DOWN BY sert à décrémenter des index

  Exemple : SET IND DOWN BY 3

Remarques :

1) Le format 2 de l'instruction SET ne peut être utilisé pour incrémenter ou décrémenter des indices ; une telle opération s'exprimerait par les instruction ADD ou SUBTRACT.

2) Dans les tests de relation, il est possible de faire référence à des index ; leurs valeurs sont "remplacées" par des valeurs entières correspondant aux rangs des occurences sur lesquelles ils pointent ; on est ainsi ramené à des comparaisons numériques.

4. Recherche en table - SEARCH

L'instruction SEARCH est utilisée pour rechercher dans une table un élément qui satisfait à une condition donnée.

4.1. Recherche séquentielle

```
SEARCH élément [ VARYING { rubrique } ]   [AT END phrase impérative]
                       { index   }

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

[END-SEARCH]
```

- L'élément de table cité après SEARCH ne doit être ni indicé ni indexé.

- Si une rubrique sert à la progression, elle doit être numérique entière (indice) ou d'usage INDEX.

La recherche débutera à partir de l'élément pointé par le contenu de l'index (ou de la rubrique) si ce dernier est inférieur au rang maximum autorisé. Les conditions sont évaluées dans l'ordre cité, et la recherche se terminera dès que l'une d'entre elles sera satisfaite ; l'index point alors l'entrée pour lequel la condition est remplie. Si la recherche a été vaine (fin de table atteinte sans satisfaire aucune condition), la locution AT END sera validée et la phrase impérative correspondante exécutée ; à défaut le contrôle passe à l'instruction suivant SEARCH.

- A la fin de l'exécution d'une phrase impérative ne comportant pas GO TO, le contrôle passe à l'instruction suivante.

- Si VARYING n'est pas spécifié, l'index utilisé par défaut est le premier cité dans la locution INDEXED BY de la définition de l'élément (idem s'il s'agit d'une rubrique après VARYING) ; les autres ne sont pas touchés.

N.B. : L'élément cité dans SEARCH peut être subordonné à une rubrique contenant OCCURS (cas de table à 2 ou 3 dimensions).

Exemple :
```
SET IND1 TO 1
SEARCH ELEMENT WHEN RUB (IND1) = VALEUR PERFORM TRAIT
               WHEN RUB (IND1) = SPACE GO TO ERREUR
```

4.2. Recherche dichotomique

```
SEARCH ALL élément [AT END phrase-impérative]

WHEN condition { phrase-impérative }
               { NEXT SENTENCE     }
[END-SEARCH]
```

où la condition est de la forme :

```
critère-1 { =        } { rubrique-1 } [ AND critère-2 { =        } { rubrique-2 } ]...
          { EQUAL TO } { exp-arith-1 } [             { EQUAL TO } { exp-arith-2 } ]
```

- Les critères doivent être cités dans la locution KEY et indexés avec le premier index ; ce peut être également un nom-condition associé à une valeur unique du critère.

- Les rubriques spécifiées ne doivent pas être des critères de l'élément concerné, ni être indexés par le premier index de celui-ci ; il en va de même pour les opérandes des expressions arithmétiques. On peut utiliser des littéraux.

- Lorsqu'un critère est utilisé, tous ceux le **précédant** dans la locution KEY doivent être également référencés.
- Le contenu de l'index sera **indéterminé** en cas de recherche infructueuse (critères non trouvés ou non discriminatoires).

Exemple :

```
.
.
05 RESID OCCURS 1000 ASCENDING DEPT COMU INDEXED BY IND IND1.
     10 DEPT PIC 99.
     10 COMU PIC 999.
     10 RUE PIC X(30).
.
.
.
         SEARCH ALL RESID WHEN DEPT (IND) = 67 AND COMU (IND) = 200
                 PERFORM STRBG-OUEST.
```

GESTION DES FICHIERS EN COBOL

1. Organisation et accès supportés

Le langage COBOL permet la manipulation de fichiers d'**organisation**

- s**équentielle** (SEQUENTIAL)
- **séquentielle indexée** (INDEXED)
- **relative** (RELATIVE)

L'accès à un fichier organisé séquentiellement est nécessairement séquentiel. Les enregistrements sont atteints dans l'ordre physique où ils sont rencontrés.

Pour les autres organisations de fichiers, l'**accès** pourra être :

- **séquentiel**, soit dans l'ordre logique des enregistrements (SEQUENTIAL)
- **sélectif** (ou aléatoire) selon une clé (RANDOM)
- **mixte**, c'est à dire séquentiel et aléatoire, à l'intérieur d'un même programme (DYNAMIC), sans avoir à ré-ouvrir le fichier.

L'organisation d'un fichier est fixée lors de sa **création** ; le mode d'accès est précisé dans chaque programme manipulant le fichier en question.

2. Input-output section

Cette section fait partie de l' ENVIRONMENT DIVISION et se compose :

- du paragraphe FILE-CONTROL comprenant une phrase SELECT par fichier utilisé ; cette phrase permet de préciser en particulier les paramètres organisation et accès pour ce fichier et de faire le lien avec le SGF par l'intermédiaire d'un mnémonique.
- du paragraphe I-O-CONTROL (facultatif) permettant de préciser les zones d'entrée-sortie (buffers) communes à plusieurs fichiers.

2.1. Le paragraphe FILE-CONTROL

2.1.1. Syntaxe générale

Format 1 (pour fichiers d'organisation séquentielle)

```
SELECT [OPTIONAL] fich-1 ASSIGN TO fspec1 [fspec2] ...
 RESERVE entier-1 { AREA  }
                  { AREAS }
[ORGANIZATION IS  SEQUENTIAL]
[ACCESS MODE IS { LINE   }       SEQUENTIAL]
                { RECORD }
( PADDING CHARACTER IS { nd-1  } )
                       { lit-1 }
( RECORD  DELIMITER IS {STANDARD-1} )
                       {chaîne    }
 [FILE STATUS IS nd-2].
```

Format 2 (pour fichiers d'organisation relative)

```
SELECT [OPTIONAL] fich-2 ASSIGN TO  fspec1 [fspec2] ...
( RESERVE entier-1 { AREA  } )
                   { AREAS }
ORGANIZATION IS RELATIVE
(                                                          )
( ACCESS MODE IS { SEQUENTIAL [ RELATIVE KEY IS nd-2]      } )
(                {{RANDOM   }                              } )
(                {{DYNAMIC  } RELATIVE KEY IS nd-2         } )
(                                                          )
[FILE STATUS IS nd-3].
```

Format 3 (pour fichiers d'organisation séquentielle indexée)

```
SELECT [OPTIONAL] fich-3 ASSIGN TO fspec1 [fspec2] ...
( RESERVE entier-1 { AREA  } )
                   { AREAS }
```

```
ORGANIZATION IS INDEXED

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

RECORD KEY IS nd-4

[ALTERNATE RECORD KEY IS nd-5 [WITH DUPLICATES]]...

[FILE STATUS IS nd-3].
```

- Par **défaut**, organisation et accès sont considérés comme séquentiels.
- A chaque phrase SELECT doit correspondre un paragraphe FD.
- Les spécifications de fichiers permettent l'assignation d'un nom externe au nom de fichier interne, et sont spécifiques de l'implantation.
- OPTIONAL indique que le fichier peut être absent ; dans ce cas, la condition de fin-de-fichier est validée dès la première lecture.
- RESERVE spécifie, à titre documentaire, le nb de buffers d'E/S alloués au fichier.
- PADDING indique le caractère à utiliser pour compléter les blocs incomplets.
- RECORD DELIMITER traite des enregistrements de longueur variable.
- FILE STATUS définit la zone de WORKING-STORAGE SECTION accueillant le code de retour de l'opération d'E/S (cf ci-après).
- Les autres clauses sont définies par la suite.

2.1.3. Exemple de FILE-CONTROL

```
ENVIRONMENT DIVISION.
 *
 *
CONFIGURATION SECTION.
SOURCE-COMPUTER.xxx.
OBJECT-COMPUTER.xxx.
 *
 *
INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT ARTICLES-S ASSIGN TO ARTICLES-S-EX
      ORGANIZATION IS SEQUENTIAL ACCESS MODE IS SEQUENTIAL.
      SELECT ARTICLES-R ASSIGN TO ARTICLE-R-EX
      ORGANIZATION IS RELATIVE
      ACCESS MODE IS RANDOM RELATIVE KEY IS RK.
      SELECT MOUVEMENTS ASSIGN TO MVTS-EX
      ORGANIZATION IS SEQUENTIAL MODE IS SEQUENTIAL.
```

Ce programme manipule

- un fichier ARTICLES-S, d'organisation séquentielle, sur disque, de numéro logique 1
- un fichier ARTICLE-R, d'organisation relative, de numéro logique 2
- un fichier MOUVEMENTS, d'organisation séquentielle, de numéro logique 3

2.2. Le paragraphe facultatif I-O CONTROL

Il spécifie, le cas échéant, la zone mémoire d'E/S qui doit être partagée par différents fichiers :

```
[SAME [RECORD] AREA FOR fichier1 ,fichier 2 ...]...
```

Ceci équivaut à une redéfinition de zone.

3. Data Division

Il faudra y inclure une FILE SECTION permettant de définir la structure de chaque fichier et celle des enregistrements correspondants.

3.1. Syntaxe générale

Pour toutes les organisations

```
DATA DIVISION.
FILE SECTION.

[ FD fich BLOCK CONTAINS [ent-1 TO] ent- 2   { RECORDS }  ]
                                             { CHARACTERS }

[ DATA      { RECORD  } { IS  }   nd-1, [nd-2]... ]
            { RECORDS } { ARE }

[ LABEL  { RECORD IS  }  { STANDARD }  ]
         { RECORDS ARE }  { OMITTED  }

  [RECORD CONTAINS [ent-2 TO ] ent-3 CHARACTERS]

  description d'article  .....
```

Toutes les clause sont facultatives en ANS85

- La clause BLOCK n'est utile que pour les fichiers sur bande et spécifie la taille d'un bloc.
- Les noms de données nd-1, nd-2, ... correspondent aux noms attribués à chaque description d'article. Ils correspondent à des redéfinitions de la zone d'entrée-sortie du fichier.
- Seule la clause LABEL peut être obligatoire et précise la présence (STANDARD) ou l'absence (OMITTED) d'étiquettes pour le fichier, selon l'organisation et le support retenus. Elle n'est significative que pour les fichiers sur bande.

4. Procedure Division

4.1. Ouverture et fermeture d'un fichier

Tout fichier manipulé dans un programme doit être ouvert auparavant et fermé en fin de traitement.

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

CLOSE { fichier1 WITH { NO REWIND } } ...
                      { LOCK      }
```

- L'ouverture rend disponible le fichier, prépare en mémoire centrale une zone enregistrement associée au programme, vérifie les étiquettes en INPUT ou les crée en OUTPUT.
- Un pointeur d'enregistrement courant est positionné pour désigner le premier enregistrement à traiter pour les fichiers en lecture (INPUT) ou en mise à jour (I-0). Ce premier enregistrement est déterminé en fonction de l'organisation du fichier (premier enregistrement physique pour les fichiers séquentiels, enregistrement de clé donnée, valide, pour les autres organisations).
- L'ouverture en mode OUTPUT positionne le pointeur d'enregistrement courant sur l'espace correspondant au premier enregistrement du fichier.
- L'ouverture en mode EXTEND positionne le pointeur d'enregistrement courant sur l'espace disponible se trouvant après le dernier enregistrement du fichier et permet de ce fait l'adjonction de nouveaux enregistrements en fin de fichier (uniquement pour fichiers d'organisation séquentielle).
- Pour les rajouts d'enregistrements il faudra ouvrir le fichier :
  - soit en mode I-O ou OUTPUT pour les fichiers non séquentiels, et en accès non séquentiel,
  - soit en mode EXTEND, pour les fichiers séquentiels.
- L'instruction CLOSE est indispensable pour :
  - éviter que le fichier ne reste à l'issue du programme dans un état indéfini qui pourrait empêcher de la rouvrir (fichier perdu)
  - éviter la perte des derniers enregistrements écrits qui se trouveraient encore dans le buffer d'entrée-sortie (buffer incomplet).

Mode d'Accès	Instruction	INPUT	OUTPUT	I-O	EXTEND
Séquentiel	READ	X		X	
	WRITE		X		X
	REWRITE			X	
Random	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
Dynamic	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

4.2.1. READ

L'instruction provoque la lecture d'un enregistrement ; il sera utilisé sous la forme

Format 1(accès séquentiel)
```
READ  fich [NEXT] RECORD  [INTO nd-1]
[AT END phrase impérative1]

[NOT AT END phrase impérative2]
[END-READ]
```

Format 2 (accès sélectif)
```
READ  fich RECORD  [INTO nd-1]  [KEY IS nd]
[INVALID KEY phrase impérative-1]

[INVALID KEY phrase impérative-2]
[END-READ]
```

L'option KEY permet d'indiquer la clé à utiliser pour la recherche, dans le cas d'emploi de clès alternées ; elle sera aussi valable, en accès dynamique pour les lectures séquentielles consécutives, jusqu'à établissement d'une nouvelle référence. (fichiers indexés seulement).

L'option NEXT est utilisée pour une lecture séquentielle en accès dynamique.

L'instruction READ rend disponible l'enregistrement du fichier désigné par le pointeur. il est accessible dans la zone d'entrée décrite en DATA DIVISION sous la rubrique FD.

Le fichier doit être ouvert en mode INPUT ou I-O. En cas de fin de fichier ou de clé invalide, la phrase introduite par les clauses AT END ou INVALID KEY est exécutée. Dans le cas contraire, (lecture réussie) le programme se poursuit normalement en séquence.

Après une lecture réussie, le pointeur d'enregistrement désigne le prochain enregistrement valide pour les fichiers en accès séquentiel. Dans les autres cas, le pointeur est positionné sur l'enregistrement identifié par la clé. La longueur des enregistrements du fichier réel doit être identique, à celle de la zone d'entrée associée au fichier (cas d'un fichier avec des enregistrements de longeur fixe) ; elle doit être dans tous les cas inférieure ou égale à celle de la zone d'entrée (cas d'un fichier avec des enregistrements de longueur variable).

4.2.2. WRITE

L'instruction provoque l'écriture d'un enregistrement dans un fichier ouvert en OUTPUT, I-O ou EXTEND. Elle sera utilisée sous la forme

Format 1(en accès séquentiel)
```
WRITE  enreg  [FROM nd-1]
[END-WRITE]
```

Format 2 (en accès sélectif)
```
WRITE  enreg  [FROM nd-1]
[INVALID KEY phrase impérative-1]

[NOT INVALID KEY phrase impérative-2]
[END-WRITE]
```

L'instruction WRITE délivre un enregistrement au système de gestion de fichier. Cet enregistrement n'est alors **plus** disponible dans la zone de sortie associée au fichier (paragraphe FD de la DATA DIVISION), à moins que l'exécution de l'instruction WRITE n'échoue (violation de limite de fichier ou clé invalide par exemple).

Même remarque que pour la lecture : la longueur des enregistrements du fichier réel doit être compatible avec la description du fichier dans programme COBOL.
Pour les fichiers d'organisation indexée, une écriture en accès séquentiel suppose que les enregistrements sont triés dans l'ordre croissant des clés.

4.2.3. Options READ... INTO... et WRITE ... FROM ...

Elles permettent de combiner en une seule instruction les instructions de lecture ou d'écriture classqiue et les transferts entre le buffer d'entrée-sortie et une zone de travail (définie par exemple en WORKING-STORAGE) et inversement.

Ainsi :
```
                READ fichier INTO donnée ...
équivaut à      READ fichier ...
                MOVE enreg. TO donnée
```

- enreg. étant une description d'article au niveau FD du fichier correspondant
- donnée est une zone quelconque de la DATA DIVISION.

De même :
```
                WRITE enreg. FROM donnée ...
équivaut à      MOVE donnée TO enreg.
```

```
WRITE enreg ....
```

4.2.4. REWRITE

L'instruction permet la réécriture d'un enregistrement modifié dans un fichier ouvert en I-O.

Format1 (accès séquentiel) `REWRITE enreg [FROM nd-1]`

Format 2 (accès sélectif) `REWRITE enreg [FROM nd-1] [INVALID KEY ph-imp1]`

`[NOT INVALID KEY ph-imp2]`
`[END-REWRITE]`

L'enregistrement modifié est le dernier lu en accès séquentiel, celui pointé par la clé pour les autres cas. Cet ordre n'est pas utilisable pour les fichiers sur bande.

4.2.5. DELETE

L'instruction permet la suppression d'un enregistrement dans un fichier d'organisation relative ou séquentielle indexée. Elle s'écrit :

```
DELETE fichier RECORD [INVALID KEY ph-imp1]

[NOT INVALID KEY ph-imp2]
[END-DELETE]
```

L'enregistrement supprimé est le dernier lu en accès séquentiel, celui pointé par la clé pour les autres cas.

4.2.6. START

L'instruction permet le positionnement dans un fichier d'organisation relative ou séquentielle indexée, en vue d'une lecture séquentielle à partir de l'enregistrement pointé.Elle s'écrit :

```
START fichier ( KEY  IS EQUAL TO                    nd-1 )
                     IS =
                     IS GREATER THAN
                     IS >
                     IS NOT LESS THAN
                     IS NOT <
    [INVALID KEY ph-imp1]

    [NOT INVALID KEY ph-imp2]
    [END-START]
```

Cette instruction ne concerne que les fichiers relatifs ou indexés, en accès séquentiel ou dynamique, ouverts en mode INPUT ou I-O. Si KEY IS n'est pas spécifié, l'option par défaut est KEY IS EQUAL TO nd-1.

Le système de gestion de fichiers compare les clés des enregistrements avec la valeur fournie par nd1 et positionne le pointeur d'enregistrement courant sur le premier enregistrement valide dont la clé satisfait la

comparaison. En cas d'échec, la condition INVALID KEY est validée et la phrase impérative correspondante est exécutée (le pointeur est alors dans une position indéfinie). nd-1 est le nom de clé donné en INPUT-OUTPUT SECTION (clause RELATIVE KEY ou RECORD KEY).

Exemple :

```
START FICH KEY IS > RKEY INVALID KEY.
```

ou encore

```
START FICH2 INVALID KEY.
```

5. Erreurs d'entrée-sortie et code d'état du fichier

Toute opération sur un fichier (ouverture, lecture, écriture, suppression, etc...) entraîne la mise à jour par le SGF, d'un code d'état du fichier indiquant la réussite ou la nature de l'échec de l'opération en question. Ce code est accessible au programmeur et lui permet de gérer lui-m^me les erreurs d'Entreés/sorties, d'interrompre ou de poursuivre d'exécution du programme, d'afficher des messages, etc... La valeur de ce code est rangée par le SGF dans la rubrique désignée par la clause `FILE STATUS` de la phrase `SELECT`.

5.1. FILE STATUS

S'il est utilisé, le `FILE STATUS` doit être

- **défini** dans `l'INPUT-OUTPUT SECTION` (phrase `SELECT` correspondant au fichier ; clause `FILE STATUS`)
- **décrit** dans la `DATA DIVISION` (`PICTURE XX`)
- **analysé** selon la grille jointe

5.2. Gestion des erreurs d'entrées / sortie

Le traitement des erreurs d'E/S s'effectuera à l'aide d'un ou plusieurs outils suivants :

- analyse du code-état associé au fichier
- utilisation des locutions standards `AT END` et `INVALID KEY`
- utilisation de sections de déclaratives

Les **déclaratives** sont constituées d'un ensemble de **sections**, formant un bloc autonome.
Elles sont regroupées en début de `PROCEDURE DIVISION.` A chaque section est associée une phrase `USE`, spécifiant le rôle (cas d'emploi) de la déclarative. L'ensemble des déclaratives est délimité par un en-tête et un pied:

```
PROCEDURE DIVISION.
DECLARATIVES.
.........
.........
END DECLARATIVES.
```

Constituant une entité séparée de la `PROCEDURE DIVISION`, il ne peut y avoir (sauf cas particulier) de référence à des blocs standards de celle-ci depuis une section de `DECLARATIVES` (et réciproquement).

Une section de déclaratives traitant les erreurs d'E/S sera identifiée par une phrase `USE` au format suivant :

```
USE AFTER STANDARD { ERROR     } PROCEDURE ON { I-O    }
                   { EXCEPTION }              { INPUT  }
                                              { OUTPUT }
                                              { EXTEND }
```


- Une phrase USE par section, après l'en-tête.
- L'organisation et l'accès des fichiers concernés (implicitement ou explicitement) peuvent être différents.
- Une référence explicite à un fichier prévaut sur la référence par le mode d'ouverture.
- `EXCEPTION` et `ERROR` sont équivalents.

A la suite d'une erreur d'E/S/, le système de gestion des fichiers provoquera éventuellement l'exécution dela section déclarative associée au fichier concerné ; après exécution de cette section, le contrôle reviendra à l'instruction suivant celle qui a déclenché l'appel.

Le traitement général d'une erreur d'E/S est donc le suivant :

1. Le SGF renseigne le code-état associé au fichier (s'il a été défini dans le programme)

2. Si l'instruction d'E/S a provoqué une **erreur standard** (i.e. validant une locution `AT END` ou `INVALID KEY`) :

   2.a) Si la locution est fournie, la phrase impérative correspondant est exécutée ; puis le programme se poursuit en séquence .

   2.b) sinon, si une section déclarative existe pour ce fichier, elle est exécutée, puis le programme se poursuit en séquence.

   2.c) Si aucun des traitements en a) ou en b) n'est prévu, le programme sera soit **interrompu**, (soit poursuivi en séquence, suivant le système utilisé).

3. Si l'erreur d'E/S est NON-STANDARD :
   le traitement est le même qu'en 2, à partir de 2.b)

6. Codes d'Etat des Opérations d'E/S

Type	S1	S2	Seq	Ind	Rel	Signification
OK	0	0				Entree-Sortie réussie.
↓	0	2				Entrée-Sortie réussie, mais le système a détecté une clé dupliquée.
↓	0	4				Lecture faite, mais l'enregistrement n'a pas la longueur déclarée dans le programme
↓	0	5				Tentative d'`OPEN` d'un fichier `OPTIONAL` qui n'est pas présent. En mode `I-O` ou `EXTEND`, le fichier est créé.
↓	0	7				A l'occasion d'une instruction `CLOSE` ou `OPEN` avec une des clauses `NO REWIND`, `REEL/UNIT` ou `REMOVAL`, on constate que le fichier n'est pas sur bande magnétique.
AT END	1	0				Fin de fichier en lecture.
↓	1	4				Pour les fichiers relatifs, le nombre d'enregistrements du fichier dépasse la capacité de la clé relative.
INVALID KEY	2	1				Problème de séquence en traitement séquentiel. Par exemple, on a lu un enregistrement et on veut le réécrire avec une clé différente.
↓	2	2				On veut écrire un enregistrement alors qu'il en existe déjà un avec une clé identique.
↓	2	3				Tentative d'accès à un enregistrement qui n'existe pas.
↓	2	4				Tentative de `WRITE` hors des limites d'un fichier relatif ou indexé.
Erreur Permanente	3	0				Erreur permanente.
↓	3	5				`OPEN` sur un fichier, non `OPTIONAL`, qui n'est pas présent.
↓	3	7				`OPEN` sur un fichier supposé en accès direct qui, en fait, ne l'est pas.
↓	3	8				`OPEN` sur un fichier au préalable fermé par `CLOSE WITH` LOCK .
↓	3	9				`OPEN` impossible suite à un conflit entre la description Cobol et la réalité.
Erreur Logique	4	1				Tenœtive `d'OPEN` sur un fichier déjà ouvert.
↓	4	2				Tenmtive de `CLOSE` sur un fichier non ouvert.
↓	4	3				Tentative de `DELETE` ou `REWRITE` sur un enregistrement qui n'a pu être lu par un `READ` précédent.
↓	4	4				Tentative de `WRITE` ou de `REWRITE` d'un enregistrement dont la dimension ne correspond pas à la description de fichier
↓	4	6				Tentative de lecture séquentielle d'un enregistrement alors que la lecture précédente n'a as réussi.
↓	4	7				Tentative de `READ` ou `START` (fichiers relatifs ou indexés) sur un fichier non ouvert en `INPUT` ou en `I-O`.
↓	4	8				`WRITE` sur un fichier non ouvert en `OUTPUT`, en `I-O` ou en `EXTEND`.
	4	9				Tentative de `DELETE` ou `REWRITE` sur un fichier non ouvert en `I-0`.
Erreur Run-time	9	1				Fichier endommagé.
↓	9	X				Erreur irréparable - fichier endommagé.

7. Le Module Séquentiel

7.1 Les opérations d'entrées / sorties sur un fichier d'organisation séquentielle

L'**accès** à un fichier d'organisation séquentielle ne peut être que **séquentiel**.

A l'ouverture, le pointeur d'enregistrement se positionne sur le premier enregistrement disponible du fichier. Les opérations possibles sont :

- la lecture pour un fichier ouvert en mode `INPUT` ou `I-O`
- l'écriture pour un fichier ouvert en mode `OUTPUT` ou `EXTEND`
- la réécriture pour un fichier ouvert en mode `I-O`

7.1.1. LECTURE

- Elle concerne l'enregistrement indiqué par le pointeur d'enregistrement courant. L'ordre de lecture s'écrit suivant le Format 1.
- Le nom de fichier est défini dans le paragraphe FD de la `DATA DIVISION`.
- La locution facultative `AT END` permet l'exécution d'instructions (calculs, branchements, etc...) au cas où le pointeur d'enregistrement pointe sur la fin de fichier. Dans ce cas le code d'état du fichier prend la valeur "10" et la lecture échoue. Le repositionnement en début de fichier s'obtient en fermant puis rouvrant ce dernier. (`CLOSE`... suivi de `OPEN`...).
- A l'issue d'une lecture réussie, le pointeur d'enregistrement pointe sur l'enregistrement suivant.

7.1.2. ECRITURE

- Elle ne peut se faire que dans un fichier ouvert en mode `OUTPUT` ou `EXTEND`, au Format 1.
- enreg désigne une description d'article de ce fichier (voir paragraphe FD).
- Un nouvel enregistrement, dont la valeur est celle de la zone de sortie associée au fichier, est créé à la suite des enregistrements existants.

7.1.3. RE-ECRITURE

Elle est utilisée pour modifier un enregistrement existant. Plus précisément, elle permet de réécrire sur le fichier, après modification, le dernier enregistrement obtenu par une lecture réussie.

- Format 1
- Le fichier doit être ouvert en mode `I-O`.
- La longueur de l'enregistrement réécrit doit être égale à celle de l'enregistrement modifié (donc lu).

7.2. CREATION DE FICHIERS D'EDITION

On ne modifie jamais un fichier d'édition qui sera de ce fait toujours un fichier d'organisation séquentielle ouvert en mode OUTPUT. Afin de répondre aux impératifs de la mise en page, le programmeur dispose au niveau de l'ordre WRITE d'une option complémentaire lui permettant de gérer l'avance du papier (sauts de lignes et de page).
Par ailleurs, il est nécessaire de préciser pour un fichier d'édition un mode particulier au niveau de la phrase SELECT, clause ORGANIZATION du paragraphe FILE-CONTROL.

7.2.1. Syntaxe générale de l'ordre

```
WRITE enreg  [ FROM nd-1]

{ BEFORE }  ADVANCING  { lit-1     } { LINE  }
{ AFTER  }             { nd-2      } { LINES }
                       { ent-1     }
                       { mnémonique}
                       { PAGE      }

[ AT { END-OF-PAGE } ph-impérative-1 ]
     { EOP         }

[ NOT AT { END-OF-PAGE } ph-impérative-2 ]
         { EOP         }

[END-WRITE]
```

7.2.2. Exemples

```
WRITE ART-STCK AFTER ADVANCING PAGE;
```
écriture en début de page suivante
```
WRITE ART-STCK AFTER ADVANCING 2 LINES.
WRITE ART-STCK AFTER ADVANCING N LINES.
```
(la valeur de N ayant été précisée précédemment).
écriture après saut de 2 (ou N) lignes

```
WRITE ART-STCK BEFORE 3 LINES.
```
saut de 3 lignes après l'écriture.

Remarque importante concernant les fichiers d'organisation séquentielle
On ne peut pas supprimer un enregistrement dans un fichier d'organisation séquentielle.

8. Le Module Indexé

8.1. Description d'un fichier d'organisation séquentielle indexée

- Chaque enregistrement est repéré par une clé d'enregistrement faisant partie intégrante de cet enregistrement. Cette clé précisée par la clause RECORD KEY est une rubrique alphanumérique qui occupe une position fixe dans l'enregistrement.

La déclaration d'un tel fichier en COBOL se fera de la façon suivante, par exemple :

```
SELECT HISTO     ASSIGN TO FLCH
                 ORGANIZATION IS INDEXED
                 ACCESS IS DYNAMIC
                 RECORD KEY IS CLE-HISTO
                 STATUS IS ST-HISTO.
```

La description de la clé CLE-HISTO est faite dans la DATA DIVISION dans la description du fichier HISTO, par exemple

```
FD HISTO RECORD STANDARD
      DATA RECORD ENRE-HISTO.
01 ENR-HISTO.
      02 CLE-HISTO.
           03 DATE-MOUV-HISTO    PIC X(6).
           03 NOCPTE-HISTO       PIC X(6).
           03 NO-ORDRE-HISTO     PIC X(3).
      02 ...
         .....
```

8.2. L'accès séquentiel

Les enregistrements sont atteints dans l'ordre croissant des clés d'enregistrements.

8.2.1. ECRITURE

- WRITE Format 2

- Elle permet d'écrire un nouvel enregistrement à la suite des enregistrements déjà créés, le fichier étant ouvert en mode OUTPUT. La condition INVALID KEY est validée si la clé d'enregistrement est inférieure ou égale à la dernière valeur de clé écrite ou si l'espace alloué au fichier est rempli.

8.2.2. LECTURE

- READ Format 1

- Elle délivre l'enregistrement de clé immédiatement supérieure à celui qui vient d'être traité, le fichier étant ouvert en mode INPUT ou I-O.

8.2.3. SUPPRESSION

- DELETE fichier RECORD

- Elle permet la suppression du **dernier enregistrement lu**, le fichier étant ouvert en mode I-O.

Exemple :

```
READ HISTO
     .
     .
DELETE HISTO.
```

8.2.4. REECRITURE

- Format 2

- Même procédure que pour les fichiers séquentiels ; la clause INVALID KEY est validée si la clé d'enregistrement ne correspond pas à celle du dernier article lu.

8.3. L'accès sélectif (RANDOM)

Les enregistrements sont recherchés par comparaison de la clé d'enregistrement avec les valeurs de clés des enregistrements existant dans le fichier.

8.3.1. ECRITURE

- Format 2

- Le fichier étant ouvert en mode I-O ou OUTPUT, l'enregistrement est inséré logiquement dans le fichier en tenant compte de l'ordre croissant des clés. Si un enregistrement valide de même clé existe déjà, la clause INVALID KEY est validée.

8.3.2. LECTURE

- Format 2

- L'enregistrement dont la clé correspond à la valeur courante de la rubrique clé d'enregistrement est délivré dans la zone d'entrée associée au fichier. Si cet enregistrement n'existe pas, la clause INVALID KEY est valide. Le fichier est ouvert en mode I-O ou INPUT.

8.3.3. SUPPRESSION

Elle provoque la suppression logique dans le fichier de l'enregistrement identifié par la clé. La clause INVALID KEY est validée si l'enregistrement n'existe pas. Le fichier est ouvert en mode I-O.

8.3.4. REECRITURE

- Format 2

- Elle concerne un fichier ouvert en mode I-O provoque le remplacement de l'enregistrement identifié par la clé par les nouvelles données. La clause INVALID KEY est validée si un tel enregistrement n'existe pas.

8.4. L'accès dynamique (DYNAMIC)

8.4.1. Instructions classiques

- L'utilisation d'un fichier en accès dynamique permet "indifféremment" l'accès sélectif et l'accès séquentiel au cours du même programme.

- Les instructions WRITE, REWRITE, DELETE correspondent à l'utilisation **en accès sélectif** (affectation ou suppression selon la clé).

- La **lecture sélective** se fait par l'instruction au Format 2 après avoir mis à jour la clé.

- La **lecture séquentielle** se traduit par l'instruction

  READ fichier NEXT RECORD [INTO nd-2] [AT END phrase]

Dans ce cas le système recherche l'enregistrement **suivant l'ordre des clés**.

- L'accès dynamique permettra ainsi, par exemple, de lire séquentiellement un fichier à partir d'un enregistrement correspondant à une certaine clé. Pour réaliser ce positionnement on utilise l'instruction START décrite au paragraphe 4.2.6.

8.4.2. POSITIONNEMENT (START)

- Le système compare la valeur figurant dans nd-1 avec les valeurs correspondantes dans les enregistrements du fichier et sélectionne le premier enregistrement satisfaisant à cette comparaison. Pour être accessible celui-ci devra faire l'objet d'une lecture.

- nd-1 peut être la rubrique introduite par la clause RECORD KEY ou toute rubrique incluse dans cette dernière, partant de sa 1ère position gauche.

- Si la clause KEY est omise, la comparaison porte sur la **clé d'enregistrement complète et l'égalité** des valeurs est recherchée.

- Le fichier est ouvert en INPUT ou I-O et l'accès est séquentiel ou dynamique.

Exemple :

```
MOVE CLE-LUE  TO CLE-HISTO.
START HISTO KEY NOT < CLE-HISTO INVALID KEY.
….
READ HISTO NEXT.
```

8.5. Clés secondaires

Une clé secondaire (ou **alternée**) permet d'accéder aux articles d'un ficher indexé suivant un second

critère; ce sera donc une rubrique alphanumérique présente dans chaque article. Un index secondaire spécifiant cette clé aura été préalablement créé et chargé.

8.5.1. Déclaration d'une clé secondaire

- Elle se fait par la clause optionnelle

{ ALTERNATE RECORD KEY IS nd-5 [WITH DUPLICATES] } …

de la phrase SELECT, où nd-5 désigne la clé secondaire.

- La locution DUPLICATES sera spécifiée si la valeur de l'indicatif supplémentaire peut ne pas être unique dans le fichier (synonymes).

- On peut spécifier plusieurs clauses ALTERNATES (i.e. plusieurs clés secondaires), chaque clé étant décrite dans l'article du fichier.

8.5.2. Utilisation d'une clé secondaire

Une clé alternée ne peut être spécifiée que dans les instructions de lecture sélective (READ) et de positionnement (START).

8.5.2.1. Lecture sélective

READ fichier RECORD [INTO nd-1] [KEY IS clé] [INVALID KEY ph.imp]

Clé désignant la clé de référence pour la lecture (par défaut, c'est la clé primaire). Si le mode d'accès est dynamique, la clé spécifiée servira aussi de référence pour les lectures séquentielles ultérieures jusqu'à établissement d'une nouvelle clé de référence.

8.5.2.2. Positionnement

La rubrique nd-1 dans la locution KEY de l'instruction START peut désigner une clé secondaire (ou une partie gauche de celle-ci). Elle devient alors clé de référence pour toutes les lectures séquentielles ultérieures.

9. Le Module Relatif

9.1. Description d'un fichier d'organisation relative

Chaque enregistrement est repéré par sa position relative par rapport au début du fichier. Cette position est fournie par un **entier positif** défini par la clause RELATIVE KEY IS nd au niveau de la phrase SELECT de ce fichier.

```
SELECT FIART ASSIGN TO FLCH
ORGANIZATION IS RELATIVE
ACCESS MODE IS RANDOM RELATIVE KEY IS CLE1.
```

La description de la donnée CLE1 se fait dans la DATA DIVISION, il s'agit d'un **entier non signé**. Par exemple :

```
* RELATIVE KEY DE FIART :
 77 CLE1 PIC 99 value 1.
```

Remarque :

Cette rubrique n'a pas le droit d'appartenir à une description d'article du fichier concerné. Toutefois, rien ne s'oppose à ce que la donnée correspondante à la valeur de la clé soit incorporée dans l'enregistrement par copie dans une rubrique portant un nom différent. On prendra garde cependant à conserver la concordance des valeurs lors des mises à jour d'enregistrements. La clé relative d'un enregistrement ne peut bien sur être modifiée lors d'une mise à jour.

9.2. Les différents accès à un fichier relatif

Les fichiers d'organisation relative autorisent comme les fichiers indexés les trois accès :

- séquentiel (SEQUENTIAL)
- sélectif (RANDOM)
- dynamique (DYNAMIC)

L'utilisation des différentes instructions d'entrée et de sortie est rigoureusement semblable à celle décrite dans le module indexé, auquel le lecteur se référera. Il suffit d'y remplacer la notion de clé d'enregistrement par celle de clé relative. Il en est de même pour l'instruction START qui portera sur la clé relative.

La gestion de cette clé pour les lectures et les écritures non séquentielles relève du programmeur qui choisira une méthode de calcul de la clé adaptée au contexte de travail (voir les modes d'adressage relatif des fichiers).

On notera qu'après une lecture séquentielle réussie, le système de gestion des fichiers met à jour la rubrique, si la locution (facultative en séquentiel) RELATIVE KEY est spécifiée, en y chargeant la valeur correspondant à l'enregistrement qui vient d'être lu. Il en est de même pour l'écriture d'enregistrements en accès séquentiel. On peut ainsi constituer une table des clés relatives des différents articles écrits au fur et à mesure de la création du fichier.

LES FORMATS EDITES

1.Principe de l'édition

L'édition consiste à transformer en mémoire centrale une valeur numérique ou alphanumérique pour en améliorer la présentation en vue de sa visualisation sur un support externe (écran, listing).

Pour cela, deux techniques peuvent être utilisées :

- **l'insertion** : on insère dans la suite des chiffres des caractères tels que le point, le signe (+ ou -), le symbole monétaire, etc...
- la **suppression** et le **remplacement** de **zéros** non significatifs.

Le recours à ces moyens s'exprime par des symboles spéciaux qui figurent dans la description de l'image de la rubrique dite **éditée** ou d'édition (clause `PICTURE`).

Pour réaliser l'édition d'une valeur, il suffit alors de la transférer dans une rubrique d'édition ; ceci peut se faire par une instruction `MOVE` (voir règles chap. 3 paragraphe 4.8) ; si la valeur est le résultat d'un calcul, la rubrique d'édition peut être précisée par l'option `GIVING` de la dernière instruction arithmétique.

Attention !

- Une rubrique d'édition est toujours décrite en `USAGE DISPLAY`.

  La valeur obtenue après tranformation n'est en général plus de type numérique pur : de ce fait elle ne peut intervenir dans des calculs ultérieurs.

2. Les symboles d'insertion

2.1. Le point décimal

Le point **.** remplace le symbole `V` ; il assure l'alignement sur la marque décimale mais en plus génère un caractère ".".

Exemple : `E1 PIC 99.99` décrit une rubrique de 5 caractères

`MOVE 2.51 TO E1` donne 02.51

2.2. Les symboles d'insertion simple

Ce sont la virgule (**,**), l'espace (**B**), le zéro (**0**) et le slash (**/**).

Chaque symbole provoque la génération du caractère correspondant à condition de ne pas figurer dans une partie de suppression de zéros non significatifs (cf. § 3 de ce chapitre).

Exemple : `E2 PIC 9B999.99` décrit une rubrique de 8 caractères
`MOVE 1500 TO E2` donne 1b500.00

Attention !
Ne pas confondre les symboles **,** et **.** ; la virgule n'assure **pas** l'alignement sur la marque décimale.

2.3. Les symboles d'insertion fixe

Ce sont le symbole monétaire (**$**) et les caractères d'édition du signe (**+**, **-**, **CR**, **DB**). Ils ne doivent apparaître qu'une seule fois par format (`PICTURE`).

- Le symbole $ doit être le **premier** symbole du format ; il peut toutefois être précédé d'un symbole + ou d'un symbole -.
- Les symboles + et - se placent en première ou en dernière position du format ; le symbole + provoque l'édition du signe sous forme d'un caractère + (valeur positive) et d'un caractère - (valeur négative) ; le symbole - provoque l'édition d'un espace si la valeur est positive et d'un caractère - si la valeur est négative.

Exemple :

```
E3   PIC   +$99      +$31
E4   PIC   $99       $31
E5   PIC   99-       31b

MOVE 31 TO E3  E4  E5 donne les résultats respectifs
```

- Les symboles CR et DB, exclusifs des symboles + et -, se placent en **dernière position** du format ; ils provoquent la génération des chaînes respectives "CR" et "DB" si la valeur est **négative** ; sinon ils sont remplacés par **deux** espaces.

Exemple :

```
E6   PIC   99DB

MOVE  +31  TO   E6 donne  31bb
MOVE  -31  TO  E6  donne  31DB
```

2.4. Les symboles d'insertion flottante :

Ce sont le symbole monétaire (**$**) et les caractères d'édition du signe (**+** et **-**) ; l'insertion flottante est indiquée par la présence d'au moins **deux** occurrences consécutives de l'un des trois symboles cités. Ces trois symboles s'excluent mutuellement dans ce cas.

- La chaîne des symboles "flottants" ne peut être précédée que de symboles d'insertion simple ou fixe et éventuellement du point décimal ; elle ne peut être entrecoupée que par des symboles d'insertion simple ou par le point décimal ; dans ce dernier cas, la partie fractionnaire de la valeur doit être décrite en totalité par le màme symbole "flottant".

- Une occurrence d'un de ces symboles représente **un chiffre** ; les zéros qui figureraient à gauche sont remplacés par des espaces sauf celui qui précède le premier chiffre non nul : celui-ci sera remplacé par $ ou + ou - (règles identiques à celle de l'insertion fixe).

Exemple 1 :

```
E7     PIC     $$$99          décrit une rubrique de 5 caractères

MOVE  500   TO  E7   donne  ~~b~~$500
MOVE 1000   TO  E7   donne  $1000
MOVE 1      TO  E7   donne  ~~bb~~$01
```

Dans le dernier cas, l'édition des chiffres est "forcée" à la rencontre du premier symbole 9, d'où le zéro.

Exemple 2 :

```
E8  PIC  $(4).$$   décrit une rubrique de 7 caractères

MOVE    50.1  TO  E8  donne  ~~b~~$50.10
MOVE    0.01  TO  E8  donne  ~~bbb~~$.01
```

Dans le dernier cas, l'édition est forcée par le point décimal et par le fait que la partie fractionnaire est non nulle ; le chiffre 0 édité a un sens et ne peut être supprimé.

```
MOVE  0  TO  E8        donne 7 espaces
```

A noter que $(4).$9 est un format incorrect.

- Si la chaîne des symboles "flottants" contient des symboles d'insertion simple, ceux-ci sont remplacés par le caractère qui leur correspond à condition de ne pas figurer dans une partie de remplacement des zéros de gauche ; sinon ils sont systématiquement remplacés par des espaces.

Exemple 3 :

```
E9  PIC  ++,++9.99    décrit une rubrique de 9 caractères

MOVE  1000  TO  E9    donne    +1,000.00
MOVE -50  TO  E9      donne   ~~bbb~~ -50.00
MOVE  0  TO  E9       donne    ~~bbbb~~+0.00
```


3. Les symboles de suppression et de remplacement

Le principe est le même que celui que nous venons d'étudier pour l'insertion flottante excepté le fait que tous les zéros concernés sont remplacés par le même caractère.

3.1. Remplacement par des espaces

Il s'agit du symbole **Z** ; il représente un chiffre.

Exemple 1 :

```
E10  PIC Z(5)          décrit une rubrique de 5 caractères
MOVE 500  TO  E10      donne  bb500
```

Exemple 2 :

```
E11  PIC Z(5).ZZ       décrit une rubrique de 8 caractères
MOVE 500 TO E11        donne  ~~bb~~500.00
MOVE 0.05 TO E11       donne  ~~bbbbb~~.05
MOVE 0  TO E11         donne 8 espaces
```

Exemple 3 :

```
E12  PIC  ZZ,ZZZ       décrit une rubrique de 6 caractères

MOVE 1000 TO E12       donne   ~~b~~1,000
MOVE 20 TO E12         donne    ~~bbbb~~20

MOVE  0 TO E12         donne     6 espaces
```

3.2. Remplacement par des astérisques

Il s'agit du symbole ***** ; son utilisation est identique à celle du symbole Z.

Exemple 1 :

```
E13  PIC  *(5)         décrit une rubrique de 5 caractères
MOVE 100 TO E13        donne  **100
MOVE 0 TO E13          donne  *****
```

Exemple 2 :

```
E14  PIC  *(5).**      décrit une rubrique de 8 caractères
MOVE  0  TO  E14       donne  *******
```

4. Remarques générales

- la longueur d'une rubrique d'édition est constante et correspond au nombre de symboles du format.
- parmi les dispositifs étudiés, certains s'excluent mutuellement ainsi, pour un format donné :
  - les symboles Z et * et d'insertion s'excluent.
  - on ne peut faire qu'une seule spécification du signe (par + ou - en insertion fixe à gauche ou à droite, par CR ou DB, par + ou - en insertion flottante).
- insertion fixe et insertion flottante du symbole monétaire s'excluent.
- Le panachage à l'intérieur d'un même format des symboles d'édition et du symbole 9 est possible, à condition de respecter les deux règles suivantes :

1) Les symboles d'édition doivent figurer en début de format, les symboles 9 en fin de format.
2) La partie fractionnaire de la valeur éditée doit être décrite en totalité à l'aide du même symbole.

5. Rubrique numérique éditée

Une rubrique numérique éditée peut être décrite à l'aide de tous les symboles étudiés.

Il faut toutefois se souvenir que la valeur qu'elle contient est assimiliée à une chaîne de caractères ; ce fait a une incidence sur l'utilisation des instructions arithmétiques, de l'instruction MOVE et de la clause VALUE
Exemple 1 :

```
E15 PIC *(4)9.99   VALUE   "**21.50".
```

littéral non numérique

L'utilisation conjointe de la clause BLANK est possible à condition que le format ne contienne pas de symbole *.

6. Rubrique alphanumérique éditée

Une rubrique alphanumérique éditée ne peut être décrite que par les symboles A, X, 9 et les symboles d'insertion simple B, 0, /

Exemple : `E16  PIC  XXBXXBXX`

`MOVE "ABCDEF" TO E15` donne AB␣CD␣EF

7. Modification du sens de certains symboles d'édition

Les règles standards appliquées sont celles du système numérique anglo-saxon. Il est possible de réaliser deux adaptations pour se rapprocher du système français.

Elles se font dans le paragraphe `SPECIAL-NAMES` de l'`ENVIRONMENT DIVISION.`

7.1. Changement de symbole monétaire

`CURRENCY SIGN IS` lit

lit est un littéral non numérique d'un caractère, différent des chiffres, des caractères spéciaux, de l'espace et des lettres A, B, C, D, P, R, S, V, X, Z.

`CURRENCY IS "F"` oblige le programmeur à remplacer toutes les occurrences du caractère $ dans les formats numériques édités par la lettre F.

Exemple : `E17  PIC F(5)9.99`

7.2. Inversion des rôles de la virgule et du point

`DECIMAL-POINT IS COMMA`

Dans les formats édités et les littéraux numériques, le point devient un symbole d'insertion simple et c'est la virgule qui assure l'alignement sur la marque décimale.

Exemple : `E18  PIC  Z.ZZZ.ZZ9,99`

`MOVE 1500000 TO E15` donne 1.500.000,00

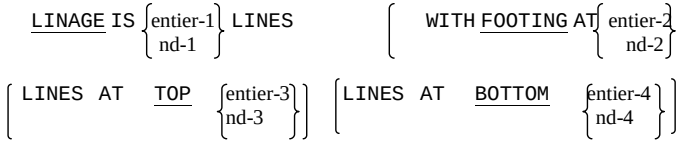
8. Gestion de page

8.1. Définition de la page logique

Il est possible de définir une PAGE LOGIQUE en termes de nombres de lignes. Une page logique comprend 3 zones :

- une marge supérieure de page (`TOP`)
- un corps de page (`LINAGE`)
- une marge inférieur de page (`BOTTOM`)

♦ On appelle pied de page (`FOOTING`) un certain nombre de lignes défini en fin de corps. La définition de page se fera dans la phrase `FD` associée au fichier d'édition par les clauses:



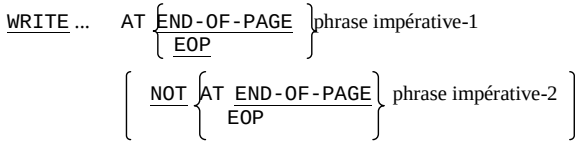
♦ Les entiers (ou les rubriques entières non signées) indiquent le nombre de lignes composant la zone concernée ; la valeur par défaut de TOP et BOTTOM est zéro, celle de FOOTING est égale au corps de page.

♦ La page logique est un concept indépendant de la page physique ; le fichier d'édition est formé d'une suite de pages logiques CONTIGUES.

♦ La clause LINAGE entraîne la génération automatique d'un compteur de lignes, LINAGE-COUNTER, contenant à tout moment le n° de ligne auquel l'imprimante est positionné à l'intérieur du corps de page courant. Il doit être qualifié par le nom du fichier s'il y a plusieurs fichiers d'édition il est réinitialisé à 1 à chaque changement de page logique.

8.2. Utilisation

Les locutions TOP et BOTTOM définissent les espacements (éventuellement variables) entre 2 pages consécutives.Les valeurs associées à LINAGE et FOOTING permettent de détecter respectivement le débordement de page et la fin de page et sont utilisées en relation avec la locution END-OF-PAGE d'une instruction WRITE :



La phrase impérative est exécutée, après que soit effectuée l'écriture :

♦ dès que l'on atteint la zone `FOOTING` (i.e. dès que la valeur de `LINAGE-COUNTER` est supérieure ou égale à entier-2)

♦ ou si l'instruction fait déborder de la page (i.e. dès que `LINAGE-COUNTER` est supérieur à entier-1).

Exemple :

```
FD IMPRIM   LABEL RECORD STANDARD
        LINAGE 50 TOP 6  BOTTOM 10.
 01 LIGNE.
       -
       -
       -
       -
            WRITE LIGNE AT EOP PERFORM FIN-PAGE.
       -
       -
FIN-PAGE. PERFORM SOUS-TOTAL ...
               WRITE LIGNE FROM EN-TETE AFTER PAGE.
```

N-B : 1) On pourrait aussi tester après l'écriture d'une ligne "normale" :

```
WRITE LIGNE
      IF LINAGE-COUNTER > 44  PERFORM FIN-PAGE.
```

2) COBOL prévoir un module d'édition (`REPORT-WRITER`) complet permettant la description de l'aspect physique et la production d'états structurés.

Exercices

	EMETTEUR		RECEPTEUR	
	CONTENU	**IMAGE**	**IMAGE**	**CONTENU**
A	12345	S9(5)	-ZZBZZ9V,99	
B	00345	S9(5)	$ZZBZZ9V,99	
C	00000	S9(5)	-ZZBZZ9V,99	
D	00000	S9(5)	$ZZBZZ9V,99	
E	00000	S9(5)	ZZBZZZ	
F	12345	9(3)V99	$ZZBZZ9V,99	
G	12345	9(5)	$****9V,99	
H	00234	9(5)	$**B**9V,99	
I	00000	9(5)	$*****V,99	
J	00000	9(5)	$****9V,99	
K	12345	99V9(3)	$**B**9V,99	
L	12345	9(5)	$$$B$$9V,99	
M	00345	9(5)	$$$,$$9.99	
N	00000	9(5)	$$$B$$9V,99	
O	12345	9(4)V99	$$$B$$9V,99	
P	12345	V9(5)	$$$B$$9V,99	
Q	-12345	S9(5)	-ZZBZZ9V,99	
R	-00345	S9(5)	-ZZBZZ9V,99	
S	00123	S9(5)	---B---V,99	
T	-00005	S9(5)	---B---V,99	
U	12345	S9(5)	+ZZZZZV,99	
V	-12345	S9(5)	+ZZBZZZV,99	
W	00345	S9(5)	++++++V,99	
X	12345	9(5)	BB999V,99	
Y	12345	S9(5)	ZZZBZZZV,99CR	
Z	00234	S9(5)	ZZZBZZZV,99CR	

COMMUNICATION INTER-PROGRAMME

1. Principes

- Un **programme principal** (ou programme **appelant**) écrit en COBOL peut appeler un (ou plusieurs) **sous-programme(s)** (ou programme(s) **appelé(s)** rédigés en COBOL ou dans un autre langage, et compilés séparément. Après édition de liens, l'ensemble constituera **une seule unité d'exécution**.

- Le contrôle est passé au sous-programme et, après son exécution, le traitement du programme principal reprend à l'instruction qui suit l'appel du sous-programme.

- Dans la norme **ANS74**, Les données traitées ou paramètres sont communes aux deux programmes; programme principal et sous-programme se transmettent, suivant un protocole fixé, les **adresses** des paramètres (passage par adresses ou références) ; il n'y a donc **PAS** de duplication des valeurs.

  La norme **ANS85**, permet au contraire le choix, lors de l'appel du sous-programme, entre le passage par par valeurs (`CONTENT`) et par adresses (`REFERENCE`).

- Dans le sous-programme, les paramètres doivent être décrits en `LINKAGE SECTION` ; dans le programme principal, les trois sections de la `DATA DIVISION` conviennent. Toutefois, en ANS74, que ce soit dans le programme principal ou dans le sous-programme, les descriptions des paramètres doivent être obligatoirement de niveau `01` ou `77`. ANS85 lève cette restriction en autorisant en paramètres des données élémentaires ou de groupe, de niveau quelconque.

2. Sous-programme COBOL

Un sous-programme est un programme (COBOL) classique dont il diffère éventuellement sur trois points :

- Il comprend une `LINKAGE SECTION` pour la description des paramètres dits **formels**.

- La déclaration de la division des traitements est complétée d'une clause `USING` fixant l'ordre des paramètres.

```
PROCEDURE DIVISION   [USING nd-1 [,nd-2]...].
```

Remarque : S'il n'y a aucun paramètre, `LINKAGE SECTION` et clause `USING` sont omises.

- L'instruction `EXIT PROGRAM` (ou `GOBACK`) doit être utilisée pour redonner le contrôle au programme principal.

Remarque : L'exécution d'une instruction `STOP RUN` à l'intérieur d'un sous- programme provoque l'arrêt immédiat et définitif du travail en cours (y compris le programme principal).

3. Appel d'un sous-programme

```
CALL { nd-1   } USING { [BY REFERENCE] {nd-2} ... } ...
     { lit-1  }       { [BY CONTENT]   {nd-2} ... }

{ [ON OVERFLOW ph-impér1]                          }
{                                                  }
{ [ON EXCEPTION ph-impér-1]                        }
{ [NOT ON EXCEPTION ph-impér-1]                    }

[END-CALL]
```

- lit-1 est le nom de programme appelé

- nd-1 est une donnée alphanumérique susceptible de contenir le nom du programme appelé.

- nd-2 ... sont les paramètres dits d'appels, décrits dans la `DATA DIVISION` du programme principal et de niveau 01 ou 77, (ou autre). Les index et les structures de fichier (FD) ne doivent pas être utilisés en paramètres.

- Option par défaut de passage des paramètres : `BY REFERENCE`

- Un mode de passage (`CONTENT` ou `REFERENCE`) est transitif sur tous les paramètres le suivant.

- Les listes de paramètres du programme principal et du sous-programme doivent être en concordance parfaite du point de vue ordre, type, représentation interne et n° de niveau des paramètres. Il n'y a aucune corrélation entre les noms de données du sous-programme et ceux du programme principal.

- Le sous-programme est initialisé lors du premier appel et conserve son état lors des appels ultérieurs, sauf s'il est doté de l'attribut `INITIAL`, ou si `CANCEL` a été exécuté dans le programme principal.

- Les clauses `OVERFLOW` et `EXCEPTION` sont validées si le programme appelé ne peut être chargé (ou ne réside pas) en mémoire, ou si la place disponible pour l'y charger est insuffisante.

4. Instructions spécifiques des sous-programmes

4.1. CANCEL

```
CANCEL { nd-1  } { nd-2  } ...
       { lit-1 } { lit-2 }
```

nd-1, nd-2, lit-1, lit-2,.... désignent des noms de sous-programmes. Cette instruction permet de libérer la zone mémoire occupée par le(s) sous-programme(s) cité(s) .

Pratiquement, elle permet de garantir que, lors du prochain appel, le sous-programme sera dans son **état initial**.

4.2 EXIT PROGRAM

```
EXIT PROGRAM.
```

- Marque la fin logique du sous-programme
- Cette instruction doit être la seule du paragraphe qui la contient
- Si le sous-programme possède l'attribut `INITIAL`, l'instruction est semblable à `CANCEL`, mais sans libération de ressources mémoire.
- Si le programme n'est pas un programme appelé, l'instruction est sans effet.
- Dans les dialectes GOBACK a un fonctionnement similaire.

5. Objets externes partagés

- Tout objet défini dans un programme COBOL lui est **interne**, et les ressources qu'il nécessite sont associées au seul programme.
- Un objet sera dit **externe** si ses ressources sont attachées non à un objet (programme) particulier, mais à l'unité d'exécution résultante. Pour cela, il faut le doter de l'attribut EXTERNE.
- Une clause `EXTERNAL` peut être associé à une spécification de fichier dans un programme, ou à un groupe de niveau `01` en `WORKING-STORAGE SECTION`. Tous les **éléments subordonnés** (zones de données) sont **automatiquement externalisés**.

  ```
  FD nom-fich [IS EXTERNAL]
  .....
  ```

  Dans ce cas, si un programme appelé contient des structures de mêmes caractéristiques (noms, formats), l'article chargé dans le programme principal lui sera accessible.

- La clause VALUE est interdite pour des objets externalisés (sauf nom-condition).

6. Exemple d'utilisation

MOY est un sous-programme permettant de calculer la moyenne de deux nombres.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MOY.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 SOMME PIC 9(5) COMP.

LINKAGE SECTION.
77 MOYENNE PIC 9(4)V99 COMP.
01 NOMBRES.
     02 N1 COMP-1.
     02 N2 PIC 9(4).


PROCEDURE DIVISION USING NOMBRES MOYENNE.
PO.  ADD N1 N2 GIVING SOMME
     DIVIDE 2 INTO SOMME GIVING MOYENNE.
P1.  EXIT PROGRAM.
```

Soit un programme principal comportant en DATA DIVISION les descriptions suivantes :

```
77 N1 PIC 9(4)V99 COMP.
01 COUPLE1.
     04 NB1 COMP-1.
     04 NB2 PIC 9(4).
01 COUPLE2.
     02 A1 COMP-1.
     02 A2 PIC 9(4).
```

On peut écrire en division des traitements :

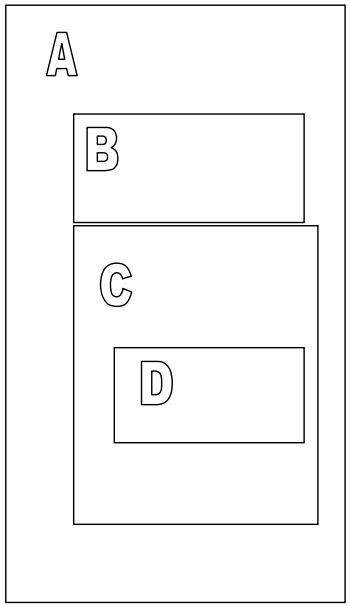
```
    '
    '
    CALL "MOY" USING COUPLE2 N1
    '
    '
    CALL "MOY" USING COUPLE1 N1
```

PROGRAMMES CONTENUS

En ANS85, un programme-source peut en **contenir** entièrement un (ou plusieurs) autres .
Cette disposition permet aux programmes de partager certaines ressources.

L'ensemble de ces programmes est compilé en même temps et ne donne création qu'à un seul programme-objet.
A ne pas confondre avec la notion classique de sous-programmes (programmes appelant et appelé étant des unités de compilations différentes, liées en une seule unité d'exécution).

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
.............

IDENTIFICATION DIVISION.
PROGRAM-ID. B.
END-PROGRAM  B.

IDENTIFICATION DIVISION.
PROGRAM-ID. C.
.............

IDENTIFICATION DIVISION.
PROGRAM-ID. D.
.............

END-PROGRAM  D.

END-PROGRAM  C.

END-PROGRAM  A.
```



- A contient **directement** B et C et **indirectement** D

C contient directement D.

- Dans cette structure l'en-tête `END-PROGRAM` nom est obligatoire.

- Les programmes contenus peuvent faire référence à des ressources des programmes contenants.

Pour cela, il faut déclarer, **dans le programme contenant**, la clause `GLOBAL` pour des groupes de niveau `01` en `FILE SECTION` ou `WORKING-STORAGE SECTION`, ou des phrases `FD`.
Tout programme **contenu**, directement ou indirectement dans un tel programme-contenant pourra référencer la donnée sans la décrire à nouveau.

- Tous les éléments subordonnés à une structure globale sont globalisés.

- Normalement un programme **contenu** ne peut être appelé que par celui le contenant directement.

La clause `COMMON`, dans son `PROGRAM-ID` permettra qu'il le soit également par des programmes de la même unité, autres que celui le contenant.

BIBLIOTHEQUE

Ce module offre la possibilité, pendant la phase de compilation, d'insérer dans un programme COBOL n'importe quel texte source enregistré dans une bibliothèque du système.

```
COPY nom-texte [ {OF|IN} bibliothèque ]
   [ { REPLACING { = = pseudo-text-1= = | nd-1 | lit-1 | mot-1 } BY { = = pseudo-text-2= = | nd-2 | lit-2 | mot-2 } }.... ]
```

- l'instruction COPY doit être suivie d'un point
- mot désigne un mot COBOL quelconque
- pseudo-text-2 peut être vide, mais non pseudo-text-1
- le texte à copier ne peut contenir de directive COPY
- les remplacements portent sur **toutes** les occurrences des éléments cités

En ANS85, l'instruction REPLACE permet des remplacements partiels

Format 1 `REPLACE {= = pseudo-text-1= = BY = = pseudo-text-2= =} ...`

Format 2 `REPLACE OFF`

- les règles sont les mêmes que pour COPY
- COPY a pécédence sur REPLACE
- le texte est remplacé jusqu'à rencontre de la prochaine instruction REPLACE (avec une nouvelle spécification ou OFF) ou, par défaut, de la fin du programme.

Fonctions Intrinsèques

Depuis 1985, COBOL comporte un module de fonctions intrinsèques (intégrées), permettant différentes opérations.

N.B.: L'utilisation de certaines fonctions n'est possible que si le module **virgule flottante** est installé.

1 Généralités

- Une fonction est un élément temporaire dont la valeur est déterminée au moment de son exécution. Elle est spécifiée par un nom réservé.

- La valeur retournée par une fonction détermine le type de celle-ci: alphanumérique, numérique, ou entière.

- Les fonctions relatives aux dates reposent sur le calendrier grégorien. Par convention, l'origine des temps est fixée au Lundi, 1er janvier 1601, correspondant à l'entier 1.

- Une fonction peut nécessiter zéro, un ou plusieurs arguments positionnels, dont les types peuvent être:
  1. Numérique: toute expression arithmétique
  2. Alphabétique: rubrique élémentaire ou littéral alphabétique
  3. Alphanumérique: toute rubrique ou tout littéral non-numérique
  4. Entier: une expression arithmétique dont l'évaluation aboutit à un entier (signé ou pas).

- Si une fonction accepte qu'un argument soit répété un nombre variable de fois, on peut spécifier une table avec ALL en indice; dans ce cas, les entrées de la table correspondant à l'échelle des valeurs de l'indice correspondant seront sélectionnées.

```
01 EXEMPLE.
     05 TOTAL                  PIC 999.
     05 VECTEUR.
          10 IND  OCCURS 5     PIC 99.


COMPUTE TOTAL = FUNCTION SUM (IND(1), IND(2), IND(3), IND(4), IND(5))
```
Est équivalent à :
```
COMPUTE TOTAL = FUNCTION SUM (IND(ALL))
```


2 Fonctions

- La colonne Arguments indique le type des arguments

A = alphabétique, I = entier, N = numérique, X = alphanumérique

- La colonne type indique le type de la fonction.

I = entier, N = numérique, X = alphanumérique

2.1 Fonctions mathématiques

Fonction	Arguments	Type	Valeurs
ACOS	N1	N	Arcosinus de N1
ASIN	N1	N	Arcsinus de N1
ATAN	N1	N	Arctangente de N1
COS	N1	N	Cosinus de N1
SIN	N1	N	Sinus de N1
TAN	N1	N	Tangente de N1
FACTORIAL	I1	I	Factorielle de I1
LOG	N1	N	Logarithme népérien de N1
LOG10	N1	N	Logarithme décimal de N1
MAX	N1 …	N	Valeur maximale des arguments
MIN	N1 …	N	Valeur minimale des arguments
MEAN	N1 …	N	Moyenne arithmétique des arguments
MEDIAN	N1 …	N	Valeur médiane des arguments (après) tri
MIDRANGE	N1 …	N	Moyenne des arguments minimal et maximal
INTEGER	N1	N	Plus grand entier inférieur ou égal à N1
INTEGER-PART	N1	N	Partie entière de N1
MOD	I1, I2	I	I1 modulo I2
RANDOM	I1	N	Nombre pseudo-aléatoire (entre 0 et 1)
RANGE	N1 …	N	Différence entre les arguments maximal et minimal
REM	N1,N2	N	Reste de N1/N2 (quotient arrondi à l'entier)
SQRT	N1	N	Racine carrée de N1
STANDARD-DEVIATION	N1	N	Ecart-type des arguments
SUM	N1 …	N	Somme des arguments
VARIANCE	N1	N	Variance des arguments

2.2 Fonctions financières

Fonction	Arguments	Type	Valeurs
ANNUITY	N1,I2	N	Taux de l'annuité pour I2 périodes à intérêt N1, pour un investissement de 1 F
PRESENT-VALUE	N1, N2 …	N	Valeur actuelle d'une série de versements périodiques N2 au taux N1

2.3 Fonctions sur les caractères

Fonction	Arguments	Type	Valeurs
CHAR	I1	X	Caractère de rang I1 dans l'alphabet-machine
LENGTH	A1 ou X1	N	Longueur de l'argument
LOWER-CASE	A1 ou X1	X	Lettres mises en minuscules
UPPER-CASE	A1 ou X1	X	Lettres mises en majuscules
MAX	X1 …	X	Valeur maximale de la liste
MIN	X1 …	X	Valeur minimale de la liste
NUMVAL	X1	N	Valeur numérique d'une chaîne de chiffres
NUMVAL-C	X1	N	Valeur numérique d'une chaîne de chiffres avec séparateurs de milliers et symbole monétaire
ORD	X1	I	Rang de l'argument dans l'alphabet-machine
ORD-MAX	X1	I	Rang de l'argument maximum dans l'alphabet-machine
ORD-MIN	X1	I	Rang de l'argument minimum dans l'alphabet-machine
REVERSE	X1	X	Ordre inverse des caractères

2.4 Fonctions sur les dates

Fonction	Arguments	Type	Valeurs
CURRENT-DATE	Aucun	X	Date, heure et fuseau (1)
DATE-OF-INTEGER	I1	I	Date (YYYYMMDD) équivalente à I1
DAY-OF-INTEGER	I1	I	Date julienne (YYYYDDD) équivalente à I1
INTEGER-OF-DATE	I1	I	Entier équivalent à la date I1 (YYYYMMDD)
INTEGER-OF-DAY	I1	I	Entier équivalent à la date julienne I1 (YYYYDDD)
WHEN-COMPILED	Aucun	X	Date et heure de compilation

(1) La fonction `CURRENT-DATE` renvoie une valeur de 21 caractères:

année	9999
mois	99
jour	99
heure	99
minute	99
seconde	99
cent.sec.	99
fuseau X	(+ ou - par rapport à Greenwich, 0 si le système ne le gère pas)
heure fuseau	99
minutes fuseau	99

Exemple

```
COMPUTE JOUR = FUNCTION REM (FUNCTION INTEGER-OF-DATE (date), 7)
```

renvoie le jour de la semaine correspondant à une *date* donnée (0 = dimanche

TRI – FUSION INTEGRES

1. Généralités

Le module de tri-fusion permet de trier ou de combiner un ou plusieurs fichiers à l'intérieur d'un programme COBOL. De plus, il est possible d'appliquer des traitements particuliers aux enregistrements concernés juste avant ou après la phase de tri (ou fusion) par des procédures spécifiques. Les principes généraux du tri-fusion intégré sont similaires à ceux des utilitaires respectifs.

2. Environment Division

Les fichiers d'entrée (locution `USING` de l'instruction) et de sortie (locution `GIVING`) sont définis de façon classique dans le programme ; les fichiers de travail du tri seront définis par une phase `SELECT` limitée à la clause `ASSIGN`

```
SELECT    fichier -tri ASSIGN TO   fspec
```

3. Data Division

Les fichiers d'entrée et de sortie seront décrits classiquement par FD.
Le fichier de travail sera décrit de manière analogue, mais sa description :

- commence par `SD` (Sort Description) au lieu de `FD`,
- ne peut comporter que les clauses `RECORD CONTAINS` et `DATA RECORD`

L'article du fichier de tri sera décrit classiquement ; il contiendra en particulier tous les critères utilisés dans l'instruction de tri. Il peut y avoir plusieurs descriptions d'articles pour le fichier de tri, mais les critères ne seront décrits que dans une seule (la première). Une rubrique servant de critère ne peut comporter de clause `OCCURS`, ni être subordonnée à une rubrique contenant OCCURS.

Exemple

```
 SD FITRI.
 01 ENRTRI.
        05 CRIT4     PIC 9(5).
        05 FILLER    PIC X(52).
        05 CRIT2     PIC XXX.
        05 RUB-A     PIC X(25).
        05 CRITERE   PIC X(10).
        05 RESTE     PIC X(68).
```


4. Procédure division

4.1. Instruction SORT

Elle permet d'ordonner, grâce au fichier de tri, les articles provenant d'un ou plusieurs fichiers d'entrée puis de les placer dans un fichier de sortie. Une PROCEDURE D'ENTREE permet éventuellement de traiter les articles AVANT de les délivrer au tri (p. ex. les modifier, en rajouter ou supprimer) ; une PROCEDURE DE SORTIE permet éventuellement de traiter les articles APRES la phase de tri, avant de les écrire dans le fichier de sortie.

N-B : On ne peut faire aucune opération sur le fichier de tri les seules instructions autorisées pour traiter ce fichier sont RELEASE et RETURN.

```
SORT fichier-tri

  ON { ASCENDING  } KEY critère-1  [critère-2]  ...
     { DESCENDING }

  ( ON { ASCENDING  } KEY critère-3  [critère-4]  ...  ) ...
       { DESCENDING }

  [ WITH DUPLICATES IN ORDER ]

        { INPUT PROCEDURE  IS section-1  [THRU section-2] }
        { USING fichier-entrée-1 [fichier-entrée-2] ...   }

        { OUTPUT PROCEDURE IS section-3 [THRU section-4]  }
        { GIVING fichier-sortie                           }
```

- `THRU` peut s'écrire `THROUGH`

- `USING` (resp. `GIVING`) et procédure d'entrée (resp. sortie) sont mutuellement exclusifs.

- Le fichier-tri doit être défini en `SD`.

- Les critères doivent être définis dans l'article de fichier-tri ; leur ordre reflète leur hiérarchie (premier = majeur).

- Avec `WITH DUPLICATES IN ORDER` les doublons seront traités dans l'ordre où ils figurent dans le(s) fichier(s) d'entrée ou sont délivrés par la procédure d'entrée.

- Une procédure d'entrée ou de sortie ne peut contenir d'instruction `SORT` ou `MERGE` ; elle ne peut non plus inclure d'instruction `ALTER`, `GO`, `PERFORM` référençant des procédures extérieures.

- L'instruction `RELEASE` sert à transférer un article dans le fichier de travail, depuis la procédure d'entrée ; l'instruction `RETURN` rend disponible à la procédure de sortie un article du fichier de travail.

- Si la locution USING est utilisée, les fichiers d'entrée ne doivent PAS être ouverts lorsque l'instruction SORT est exécutée ; idem pour GIVING et le fichier de sortie.

4.2. Instructions RELEASE et RETURN

```
RELEASE  enreg [FROM zone]
```

L'instruction RELEASE transfère un article à la phase initiale du tri. enreg est le nom d'article figurant dans la description SD ; zone désigne une rubrique à partir de laquelle sera renseigné l'article (cf WRITE FROM).
Après exécution d'une instruction RELEASE, l'article n'est plus disponible dans le buffer du fichier-tri.

```
RETURN fichier-tri RECORD [INTO zone] AT END phrase impérative
  [NOT  AT END phrase impérative ]
[ END-RETURN ]
```

L'instruction RETURN permet d'obtenir un article issu de la phase finale du tri. La condition AT END est validée lorsque le dernier article du fichier de travail a été délivré à la procédure de sortie.

4.3. Instruction MERGE

Elle permet de combiner deux ou plusieurs fichiers ordonnés sur des critères identiques puis rend les articles disponibles pour une procédure de sortie ou un fichier de sortie.

```
 MERGE  fichier-fusion

ON { ASCENDING  } KEY critère-1 [critère-2]...
   { DESCENDING }

[ ON   ASCENDING    KEY critère-3  [critère-4]  ...  ]...
       DESCENDING

USING  fichier-entrée-1 fichier-entrée-2 [fichier-entrée-3]…

{ OUTPUT PROCEDURE IS section-1 [THRU section-2] }
{                                                }
{ GIVING fichier-sortie                          }
```

- RETURN sera utilisée en cas de procédure de sortie.

5. Exemples

5.1. Tri sans procédure

```
PROCEDURE DIVISION.
    DEBUT. OPEN INPUT (ou OUTPUT) FIENT.
    TRAIT.   traitement du fichier FIENT...
             CLOSE FIENT.

    TRI.     SORT FITRI ASCENDING CRIT2 DESCENDING CRIT4
             USING  FIENT GIVING FISOR.
    SUITE.   OPEN INPUT FISOR
             traitement  du fichier FISOR  ...
             CLOSE FISOR.
...
```

5.2. Tri avec procédure d'entrée

```
PROCEDURE DIVISION.
-
-
TRI. SORT FITRI ON ASCENDING CRIT3
       INPUT PROCEDURE ENTRI GIVING  FISOR.
-
-
ENTRI SECTION.
E1.    OPEN INPUT FIENT.
E2.    READ FIENT AT END GO E9.
-
-      traitement
-
E5.    RELEASE ENTRI FROM ARTICLE. GO E2.
E9.    CLOSE FIENT.
```

5.2. Fusion avec procédure de sortie

```
PROCEDURE DIVISION.
-
-
-
FUSION. MERGE FITRI DESCENDING RUB-A ASCENDING CRIT2
          USING FIENT1 FIENT2 FIENT3
          OUTPUT PROCEDURE FUSOR.
-
-
-
FUSOR SECTION.
F1.     OPEN OUTPUT FISOR.
F2.     RETURN FITRI AT END GO F9.
-
```

```
-     traitement
-
F6.     WRITE ARTICLE FROM ENTRI.
       GO F2.
F9.     CLOSE FISOR.
```

LISTE DES MOTS RESERVES

ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALL
ALPHABET
ALPHABETIC
ALPHANUMERIC
ALPHANUMERIC-EDITED
ALPHANUMERIC-LOWER
ALPHANUMERIC-UPPER
ALSO
ALTER
ALTERNATE
AND
ANY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEEP
BEFORE
BINARY
BLANK
BLINK
BLOCK
BOTTOM
BY
CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMON
COMMUNICATION
COMP
COMP-3
COMP-6
COMPUTATIONAL
COMPUTATIONAL-3
COMPUTATIONAL-6
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTENT
CONTINUE
CONTROL
CONTROLS
CONVERTING
COPY
CORR
CORRESPONDING
COUNT
CRT
CRT-UNDER
CURRENCY
CURSOR
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK
DE
DEBUG-CONTENTS
DEBUGGING
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC
ECHO
EGI
ELSE
EMI
ENABLE
END
END-ALL
END-CALL
END-COMPUTE
END-DELETE
END-DIVIDE
END-EVALUATE
END-IF
END-MULTIPLY
END-OF-PAGE
END-PERFORM
END-READ
END-RECEIVE
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END-SUBTRACT
END-UNSTRING
END-WRITE
END-OF-PAGE
ENTER
ENVIRONMENT
EOL
EOP
EOS
EQUAL
ERASE
ERROR
ESI
EVERY
EXCEPTION
EXIT
EXTEND
EXTERNAL
FALSE
FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FOOTING
FOR
FROM
GENERATE
GIVING
GLOBAL
GO
GREATER
GROUP
HEADING
HIGH
HIGH-VALUE
HIGH-VALUES
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIALIZE
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
I-O
I-O-CONTROL
IS
JUST
JUSTIFIED
KEY
LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW
LOW-VALUE
LOW-VALUES
MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
NATIVE
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC
NUMERIC-EDITED
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORDER
ORGANIZATION
OTHER
OUTPUT
OVERFLOW
PACKED-DECIMAL
PADDING
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINT
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROMPT
PURGE
QUEUE
QUOTE
QUOTES
RANDOM
RD
READ
RECEIVE
RECORD
RECORDS
REDEFINES
REEL
REFERENCE
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACE
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN
SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2

SUB-QUEUE-3	SUBTRACT	SUM
SUPRESS	SW0	SW1
SW8	SWITCH0	SWITCH-0
SWITCH-1	SWITCH-2	SWITCH-3
SWITCH-4	SWITCH-5	SWITCH-6
SWITCH-7	SWITCH-8	SWITCH-9
SYMBOLIC	SYNC	SYNCHRONIZED
SYSIN	SYSOUT	TAB
TABLE	TALLY	TALLYING
TAPE	TERMINAL	TERMINATE
TEXT	THAN	THEN
THROUGH	THRU	TIME
TIMES	TO	TOP
TRAILING	TRUE	TYPE
UNDERLINE	UNIT	UNLOCK
UNSTRING	UNTIL	UP
UPDATE	UPON	USAGE
USE	USING	VALUE
VALUES	VARYING	WHEN
WITH	WORDS	WORKING-STORAGE
WRITE	ZERO	ZEROES
ZEROS		

MISE AU POINT

1. GENERALITES

Le module de mise au point (**Debugging**) de COBOL offre au programmeur des moyens de contrôler l'exécution de son algorithme. Ils consistent en :

- des lignes de mise au point (**D** en colonne 7)
- des sections déclaratives (USE FOR DEBUGGING)
- un registre spécial (DEBUG-ITEM) géré par le système et accessible au pg

Ces outils seront mis en (ou hors) fonction grâce à des inverseurs (commutateurs, "switchs") logiques, soit lors de la compilation, soit lors de l'exécution.

Ce module est classé obsolète en ANS85.

2. INVERSEURS

L'**inverseur de compilation** est constitué par la clause WITH DEBUGGING MODE du paragraphe SOURCE-COMPUTER. Sa présence dans le pg entraîne la compilation de toutes les instructions de mise au point (D en col. 7 et sections de mise au point) y figurant. Son absence implique que les instructions précédentes seront ignorées du compilateur (i.e. considérées comme des commentaires).

L'**inverseur d'exécution** est un switch logique mis en ou hors fonction lors de l'appel de l'exécutable, grâce à un ordre ou un paramètre particulier.
S'il est ON, les phrases USE FOR DEBUGGING sont validées; sinon (OFF), elles sont inhibées.

3. SECTIONS DECLARATIVES DE MISE AU POINT

Ce sont des sections spécifiques placées dans la partie déclaratives, au début de celle-ci (donc **avant** toute éventuelle section de traitement des erreurs d'entrée-sortie), identifiées par une phrase USE FOR DEBUGGING ... et constituant des **blocs autonomes** (pas de référence d'une procédure déclarative à une procédure non-déclarative, dans un sens ou dans l'autre). Cf chap.5 "Gestion des Fichiers".

```
USE FOR DEBUGGING ON  { {[ALL REFERENCES OF] nd } ...
                      { {nom-fichier} ...
                      { {nom-procédure} ...
                      { ALL PROCEDURES                  } ...
```

- Toute variable apparaissant après ON peut
  - n'apparaître que **dans une seule phrase** USE
  - n'y figurer qu'**une seule fois**.

- Si ALL PROCEDURES est cité dans une phrase USE, aucun nom de procédure ne peut figurer dans aucune phrase USE.

- Si nom-donnée contient une clause OCCURS ou est subordonné à une rubrique la contenant, on ne spécifie pas l'indice ou l'index normalement nécessaire.

Règles d'application

- Lorsqu'un nom-fichier est spécifié, la section déclarative est exécutée **après** exécution valide d'une instruction OPEN, CLOSE, READ, START ou DELETE.

- Lorsqu'un nom-procédure est spécifié, la section déclarative est exécutée immédiatement **avant** chaque exécution de la procédure citée; s'il s'agit de ALL PROCEDURES, cette règle s'applique pour chaque procédure du pg.

- Lorsqu'un nom-donnée est spécifié **seul**, la section déclarative est exécutée:
  a) **avant** toute instruction WRITE ou REWRITE qui cite explicitement le nom-donnée (cas de l'article), ou après la locution FROM;
  b) **après** chaque initialisation, modification, évaluation de nom-donnée quand cette rubrique apparaît après VARYING, AFTER, UNTIL dans une instruction PERFORM;
  c) **après** toute autre instruction qui cite explicitement nom-donnée ET entraîne la **modification** de son contenu.

- Lorsqu'un nom-donnée est spécifié précédé de la locution ALL REFERENCES OF, la section déclarative est exécutée :
  a) idem a) ci-dessus
  b) idem b) ci-dessus
  c) **avant** transfert du contrôle quand nom-donnée est cité dans la locution DEPENDING ON d'une instruction GO TO;
  d) **apres** execution de toute autre instruction qui cite explicitement nom-donnée (même si elle ne modifie pas son contenu).

Remarques :

- Si nom-donnée est spécifié dans une phrase qui n'est ni exécutée ni évaluée, la section déclarative n'est PAS exécutée.
- Pour une instruction donnée, la déclarative n'est exécutée qu'une seule fois, quel que soit le nombre de fois où nom-donnée est cité dans l'instruction; pour une itérative (PERFORM), elle est exécutée à chacue iteration.

4. DEBUG-ITEM

Ce registre spécial, généré automatiquement, est associé à chaque exécution d'une section déclarative de mise au point; il fournit des informations sur les conditions gui ont provooué l'exécution de cette dernière; sa description **implicite** est la suivante :

```
01  DEBUG-ITEM.
  02  DEBUG-LINE       PIC X(6).
  02  FILLER           PIC X    VALUE SPACE.
  02  DEBUG-NAME       PIC   X(30).
  02  FILLER           PIC X    VALUE SPACE.
  02  DEBUG-SUB-1      PIC S9999 SIGN LEADING SEPARATE.
  02  FILLER           PIC X    VALUE SPACE.
  02  DEBUG-SUB-2      PIC S9999 SIGN LEADING SEPARATE..
  02  FILLER           PIC X    VALUE SPACE.
  02  DEBUG-SUB-3      PIC S9999 SIGN LEADING SEPARATE.
  02  FILLER           PIC X    VALUE SPACE.
  02  DEBUG-CONTENTS   PIC X(n).
```

Tous les noms cités sont des **mots réservés**.

- DEBUG-CONTENTS a une taille suffisante pour contenir la donnée definie par les règles ci-après.
- Le contenu de DEBUG-ITEM est **mis à blanc avant** chaque exécution d'une section déclarative; sa mise à jour s'exécute suivant les règles de MOVE (sans aucune conversion pour DEBUG-CONTENTS) .
- DEBUG-LINE contient un numéro de ligne-source (listing).
- DEBUG-NAME contient les 30 premiers caractères du nom qui a provoqué l'exécution de la section déclarative.
- Si la donnée qui provoque l'exécution de la déclarative est indicée ou indexée , le rang de chaque niveau est placé respectivement dans DEBUG-SUB-1,DEBUG-SUB-2,DEBUG-SUB-3.

Règles d'application

A - Exécution résultant d'un nom-donnée
DEBUG-LINE contient le n° de ligne de l'instruction origine,
DEBUG-NAME contient le nom-donnée,
DEBUG-CONTENTS contient le contenu de cette rubrique.

B - Exécution résultant d'un nom-fichier
DEBUG-LINE contient le n° de ligne de l'instruction origine,
DEBUG-NAME contient le nom-fichier,
DEBUG-CONTENTS contient l'article lu en entier.

C - Exécution résultant d'une procédure citée dans PERFORM
DEBUG-LINE contient le n° de ligne de l'instruct'on PERFORM,
DEBUG-NAME contient le nom-procédure appelé,
DEBUG-CONTENTS contient "PERFORM LOOP".

D - Exécution résultant d'un nom-procédare rencontré séquentiellement
DEBUG-LINE contient le n° de ligne de l'instruction précédente,
DEBUG-NAME contient nom-procédure ,
DEBUG-CONTENTS contient "FALL THROUGH".

E - Exécution résultant d'un nom-procédure cité dans une locution INPUT ou OUTPUT d'une instruction SORT ou MERGE
DEBUG-LINE contient le no de ligne de l'instruction SORT ou MERGE,
DEBUG-NAME contient le nom-procédure d'entrée (ou de sortie),
DEBUG-CONTENTS contient suivant le cas "SORT INPUT", "SORT OUTPUT"
ou "MERGE OUTPUT".

F - Exécution résultant d'un nom-procédure cité dans GO TO
DEBUG-LINE contient le n° de ligne de l'instructionGO TO,
DEBUG-NAME contient le nom-procédure,
DEBUG-CONTENTS est à blanc.

G - Exécution résultant du premier nom-procédure cité dans le pg
DEBUG-LINE contient le n° de la première instruction;
DEBUG-NAME contient nom-procédure,
DEBUG-CONTENTS contient "START PROGRAM".

H - Exécution résultant d'un nom-procédure cité dans ALTER
DEBUG-LINE contient le n° de ligne de l'instruction ALTER,
DEBUG-NAME contient nom-procédure,
DEBUG-CONTENTS contient le nom de la procédure à exécuter (associée à TO).

PRINCIPALES DIFFERENCES

ENTRE ANS74 ET ANS85

Programmes contenus

Données GLOBALES et EXTERNES

Portée des instructions (END-...)

Lettres minuscules

Modificateurs de référence

Fonctions intrinsèques (1989)

EVALUATE

Option CONVERTING dans INSPECT

INITIALIZE

Opération nulle CONTINUE

Passage de paramètres par valeur

Dimensions de tableaux à 7

En-têtes de division facultatifs

Clause VALUE autorisée avec OCCURS

Transfert numérique édité vers numérique

Clause WITH NO ADVANCING dans DISPLAY

CODAGE

Valeurs hexadécimales: h1 = demi-octet gauche (forts poids)
h2 = demi-octet droit (faibles poids)

Code ASCII

	h1							
h2	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Perforation Hors-texte

	1	2	3	4	5	6	7	8	9	0
+	A	B	C	D	E	F	G	H	I	{
-	J	K	L	M	N	O	P	Q	R	}

Code EBCDIC

	h1											
	4	5	6	7	8	9	A	B	C	D	E	F
0	SP	&	-						{	}	\	0
1			/		a	j	~		A	J		1
2					b	k	s		B	K	S	2
3					c	l	t		C	L	T	3
4					d	m	u		D	M	U	4
5					e	n	v		E	N	V	5
6					f	o	w		F	O	W	6
7					g	p	x		G	P	X	7
8					h	q	y		H	Q	Y	8
9				`	i	r	z		I	R	Z	9
A		!	|	:							
B	.	$	,	#								
C	<	*	%	@								
D	(	)	_	'								
E	+	;	>	=								
F		^	?	"				ç				