Commandes COBOL

0 – Introduction : Le Cobol Immortel

1 – Structure d'un programme

2 – Syntaxe

3 – Déclaration de variable

4 – Utilisation de noms-conditions

5 – Affectation de variable

6 – Déclaration de tableau, affectation et recherche

7 – Condition (IF)

8 – Boucle (PERFORM VARYING)

9 – Branchements multiples simples (EVALUATE)

10 – Branchements multiples composés (EVALUATE)

11 – Opérations sur fichier

12 – Quelques instructions

13 – Débuggage

14 – Programme externe – le module

15 – Programme externe – la clause COPY

16 –Opérations arithmétiques

0. Introduction : Le Cobol Immortel :

Le **Cobol** ou **C**ommon **O**rganization **B**usiness **O**riented **L**anguage est par définition un langage orienté gestion. C'est l'un des tout premiers langages conçus dans l'histoire de l'informatique.

C'est en 1957, sur la demande du gouvernement des U.S.A, qu'une étude fut développée par un comité d'utilisateurs et de constructeurs d'ordinateurs : la **CODASYL** (**CO**nference on **D**Ata **SY**stems **L**anguages, en français « *Conférence sur les langages des systèmes de traitement de données »*).

Dans les projets de l'an 2000, plusieurs des programmes à convertir étaient rédigés en Cobol, du fait qu'il n'offre pas de structure particulière pour stocker une date, ou qu'aucune fonction n'a été prévue pour effectuer des calculs sur les dates, pour en formater l'affichage et pour vérifier qu'une année est bissextile.

Pourtant, à ce jour, malgré les critiques et les nombreux langages qui ont vu le jour depuis, le Cobol reste très employé :

- les **entreprises et les administrations** dans lesquelles sont encore implantés aujourd'hui les millions de programmes rédigés en Cobol durant ces quarante dernières années n'ont pas les moyens de réécrire leurs applications.

- le Cobol est un standard **international** largement **documenté**, réputé pour sa **stabilité** et sa **solidité**, dont les compilateurs sont universellement supportés, il est upwards compatible et il est disponible sur PC.

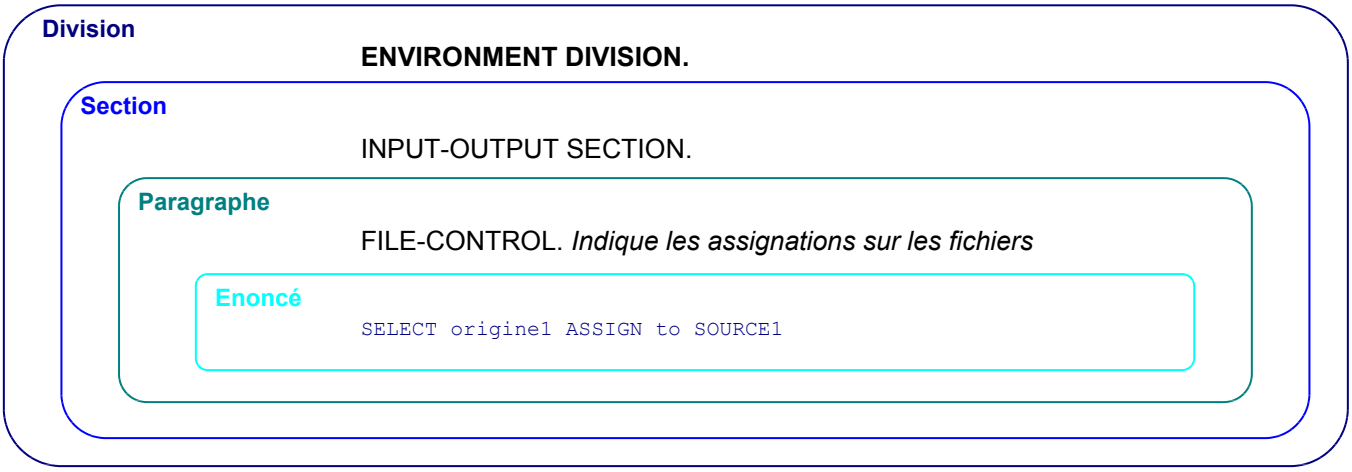
- "*N'oubliez pas que 85 % des applications cruciales pour les entreprises sont toujours basées sur Cobol*" rappelle Bart Abeel de Merant (éditeur anglo-américain spécialiste de Cobol, reconverti dans les plates-formes de commerce électronique). La proportion est sans doute encore plus impressionnante aux Etats-Unis.

Pour ce qui est de l'informatique moderne, il faut savoir que les utilisateurs et les programmeurs Cobol n'en sont pas exclus. Après 40 années d'existence, le Cobol n'est pas devenu old-fashioned : cet outil de développement, que le Y2K a remis en vedette, continue d'attirer les investissements et les réseaux PC, généralement connectés à un mainframe (gros système) lorsqu'il y a usage du Cobol, offrent aux utilisateurs la plupart des services de la toute récente informatique, du courrier intra-société aux connexions au web. Cette jouissance des produits les plus actuels empêche de voir poindre tout sentiment de 'démodé'.

Le nouveau Cobol, qui n'est pas encore tout à fait arrivé, s'ouvre à des techniques modernes comme la programmation orientée objet, aux GUI (graphical user interfaces), à Java ou à Internet et son HTML.

1. Structure d'un programme :

Un programme Cobol est structuré en 4 **divisions**, elles-mêmes composées en **sections** organisées en **paragraphes**. Et les paragraphes contiennent des **énoncés**.



IDENTIFICATION DIVISION. *Identifie le programme pour l'utilisateur*

PROGRAM-ID. Nom du programme
AUTHOR. Nom de l'auteur
DATE COMPILED. Date

ENVIRONMENT DIVISION. *Description sommaire de la configuration du calculateur utilisé, informations relatives à la gestion des entrées-sorties, aux conditions spéciales de traitement des fichiers*

CONFIGURATION SECTION.	*Renseignements sur l'ordinateur*
SPECIAL-NAMES.	
CURRENCY SIGN IS « F »	*Indique le symbole monétaire*
DECIMAL-POINT IS COMMA	*Indique le séparateur des décimales*
INPUT-OUTPUT SECTION.	
FILE-CONTROL.	*Indique les assignations sur les fichiers*

DATA DIVISION. *Définit toutes les données*

FILE SECTION.	*Description des fichiers*
WORKING STORAGE SECTION.	*Déclaration des variables internes*
LINKAGE SECTION.	*Indique les données d'un lien (module)*

PROCEDURE DIVISION. *Traitements*

2. Syntaxe :

Les noms de **divisions**, de **sections** et des **paragraphes** :

- commencent en colonne 8 (zone A)
- se composent de 1 a 30 caractères A … Z, 0 … 9 et - (trait d'union)
- se terminent par un point.

Les **instructions** :

- commencent en colonne 12 ou plus (zone B)
- peuvent s'étendre sur plusieurs lignes (sans couper de mot)
- chaque instruction peut prendre autant de ligne que nécessaire mais doit se terminer par un point (obligatoire dans les trois premières divisions, facultatif dans la quatrième – sauf si c'est la dernière instruction de la procédure).

Les **données** :

Chaque donnée porte un nom et un numéro (indicateur de niveau) qui indique le niveau d'une donnée dans une structure.

On distingue deux types de données :

- données de type "record" (ou enregistrement) qui peuvent se décomposer en sous-structure (indicateur de niveau 01)
- données de type "individuel" qui ne peuvent pas être décomposées (indicateur de niveau 77).
-

Les **commentaires** :

Commencent en colonne 7 par *.

3. Déclaration de variable :

Format : Indicateur de niveau nom de la variable [**PICTURE** type de la variable (taille)]
[**REDEFINES** nom variable 2]
[**VALUE**[**S**] {**SPACES**,valeur1 [valeur2…]]}

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING STORAGE SECTION.

PROCEDURE DIVISION.

a) Indicateur de niveau :

Cet indicateur peut-être :

01 : Enregistrement ou groupe de données
02 à 49 : sous-groupe de données
66 : Regroupement de zones
77 : Donnée élémentaire (c'est-à-dire non subdivisée)
88 : Nom-condition

b) Type de la variable (classe de données) :

A : Alphabétique (lettre ou espace)
X : Alphanumérique (tout caractère)
Z : Numérique (les zéros de tête sont effacés)
9 : Numérique (les zéros de tête sont conservés)
9 COMP ou **COMP-4**: Binaire
9 COMP-3 : Binaire packé

c) Formatage de la variable :

B : Blanc
P : Point décimal (présence de zéros virtuels)
V : Point décimal virtuel
S : Présence du signe (si absent, seulement le négatif sera affiché)
$: Position du signe monétaire
/ , . : éléments affichés tels quels
+ : Plus ou Moins
- : Espace ou Moins

d) Instruction REDEFINES :

Permet de préciser qu'un groupe de variables pointe sur l'adresse mémoire d'une autre variable.

```
01 ZoneTemp PIC X(30).

01 Type1 REDEFINES ZoneTemp.
    05 NomType1 PIC X(20).

01 Type2 REDEFINES ZoneTemp.
    05 NomType2 PIC X(30).
```

e) Exemples (ne pas oublier le POINT après une déclaration) :

```
77 chaine PICTURE X(20).
```
chaine est une chaîne de longueur 20 caractères.

```
77 nombre PICTURE 9(2).
```
ou
```
77 nombre PIC 99.
```
nombre est codé sur deux octets, PIC abréviation de PICTURE.

```
77 nombre PICTURE 9(2) VALUE 4.
```
la valeur 4 est affecté à la variable nombre lors de sa déclaration.

```
01 groupe.
    05 nom PIC X(20).
    05 prenom PIC X(20).
```
groupe est un groupe comprenant deux variables. Sa taille est le cumul de toutes les tailles des variables.

4. Utilisation de noms-conditions :

Au lieu de tester la valeur d'un élément d'un groupe, le Cobol nous permet de nommer cet élément et d'utiliser ce nom par la suite. Cela est possible via l'indicateur de niveau 88.

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING STORAGE SECTION.

PROCEDURE DIVISION.

a) Déclaration :

```
01 Sexe-personne PIC 9.
       88 Masculin VALUE 1.
       88 Feminin VALUE 2.
```

Dans cet exemple, les deux variables « Masculin » et « Féminin » sont créées et sont également accessibles depuis la zone groupe « Sexe-personne ». Celle-ci est retourne la valeur de la condition sélectionnée.

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING STORAGE SECTION.

PROCEDURE DIVISION.

b) Affectation :

```
SET Masculin to TRUE.
```

Ou

```
MOVE 1 TO Sexe-personne.
```

c) Utilisation :

```
If Sexe-personne = 1 THEN DISPLAY 'Homme'
```

Ou plus simplement :

```
If Masculin THEN
       DISPLAY 'Homme'
ELSE
       DISPLAY 'FEMME'
END-IF
```

Seulement une seule des variables peut être vraie. Cela fonctionne en fait comme un booléen avec d'autres langages.

5. Affectation de variable :

L'affectation d'une variable peut se faire soit dans la déclaration, avec l'instruction VALUE, soit par la commande MOVE.

Format : **MOVE** valeur ou variable **TO** nom de la variable réceptrice
[**OF** nom de la zone groupe]

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING STORAGE SECTION.

PROCEDURE DIVISION.

L'instruction OF permet de préciser la zone groupe si deux noms de variables sont identiques mais appartiennent à des zones groupes différentes.

Exemple :

DATA DIVISION.
WORKING STORAGE SECTION.

```
01 groupe1.
       05 nom PIC X(20).
       05 prenom PIC X(20).

01 groupe2.
       05 nom PIC X(20).
       05 prenom PIC X(20).
```

…..

PROCEDURE DIVISION.

```
MOVE « Robert » TO nom OF groupe1.
```

Il est également possible de faire des transfert entre les zones de mêmes noms :

```
MOVE CORRESPONDING groupe1 TO groupe2
```

Est équivalent à :

```
MOVE nom       OF groupe1 TO nom     OF groupe2
MOVE prenom    OF groupe1 To prenom  OF groupe2
```

6. Déclaration de tableau, affectation et recherche :

On peut utiliser des tableaux (simples) de une à sept dimensions.

Format : Indicateur de niveau nom du tableau **OCCURS** taille du tableau
[**INDEXED BY** nom de l'index]
[**ASCENDING KEY IS** nom de la clé de tri]

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING STORAGE SECTION.

PROCEDURE DIVISION.

a) Exemple :

```
01 Liste
       02 Etud OCCURS 65 INDEXED BY I ASCENDING KEY IS Nom
       03 Nom PIC X(12)
       03 Note PIC 99V99 OCCURS 1 TO 10
```

Cet exemple déclare un tableau Etud de 65 éléments, chacun contenant un nom et jusqu'à 10 notes.
« INDEXED BY I » indique le nom de l'index.
« ASCENDING KEY IS Nom » précise que le tri se fait par nom.

b) Affectation :

```
MOVE « ROBERT » TO Nom(1)
MOVE 12,5 TO Note(1 1)          1ère note de l'élève 1
MOVE 15 TO Note(1 2)            2ème note de l'élève 1
```

c) Recherche :

```
SEARCH ALL Etud
       AT END DISPLAY « Elément non trouvé »
       WHEN NOM-REC=NOM(I) DISPLAY « Elément trouvé »
```

Cet exemple recherche la variable NOM-REC dans tous le tableau. On indique NOM(I) car I est la variable d'index.

7. Condition :

Format : **IF** Condition **[THEN]**
Traitement
[**ELSE**
Traitement]
END-IF

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.

Expressions possibles :

IS [NOT] NUMERIC
IS [NOT] ALPHABETIC
IS [NOT] ALPHABETIC-LOWER *si texte en minuscule ou pas*
IS [NOT] ALPHABETIC-UPPER *si texte en majuscule ou pas*

IS [NOT] POSITIVE
IS [NOT] NEGATIVE
IS [NOT] ZERO

[NOT]> ou [NOT] GREATER
[NOT]< ou [NOT] LESS
[NOT]= ou [NOT] EQUAL
<=, >= AND, OR

8. Boucle :

Cette instruction permet d'effectuer une boucle. Un pas négatif est également autorisé.

Format : **PERFORM VARYING** variable **FROM** début BY pas UNTIL événement
traitement
END-PERFORM

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.

Exemple :

```
PERFORM VARYING i FROM 1 BY 1 UNTIL i = 5
      DISPLAY i
END-PERFORM
```

Affichage de 1, 2, 3 et 4 car il y a incrémentation du pas puis test de la valeur

Ainsi, pour lister un tableau de 10 éléments (boucle simple) :

```
PERFORM VARYING i FROM 1 BY 1 UNTIL i > 10
      DISPLAY liste(i)
END-PERFORM
```

9. Branchements multiples simples :

L'instruction EVALUATE permet d'évaluer séquentiellement plusieurs valeurs et d'exécuter le code correspondant. Une fois exécuté, un saut est effectué vers l'instruction de clôture (END-EVALUATE).

Format :

```
EVALUATE        {TRUE, FALSE, variable}
  WHEN    {TRUE, FALSE, variable, valeur}     instructions
  [WHEN   OTHER                               Instructions]
END-EVALUATE
```

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

PROCEDURE DIVISION.

a) Exemple avec variable dans clause EVALUATE :

```
EVALUATE Code-Transaction
  WHEN 1 MOVE 0.12 TO Taxes
  WHEN 2 MOVE 0.5 TO Taxes
  WHEN 5 THRU 9 MOVE 0.22 TO Taxes
  WHEN OTHER DISPLAY 'Erreur !'
END-EVALUATE.
```

Ou

```
EVALUATE Sexe-Personne
  WHEN TRUE MOVE 0.12 TO Taxes
  WHEN FALSE MOVE 0.5 TO Taxes
END-EVALUATE.
```

b) Exemple avec booléen (indicateur 88) dans clause EVALUATE :

```
EVALUATE TRUE
 WHEN Code  DISPLAY 'OK'
 WHEN OTHER DISPLAY 'Pas OK'
END-EVALUATE.
```

10. Branchements multiples composés :

L'instruction EVALUATE peut en plus être complétée avec le paramètre ALSO permettant d'inclure des sujets supplémentaires (jusqu'à 256). Ces sujets sont mises en parallèle avec les sujets du WHEN.

De plus, si plusieurs conditions doivent exécuter la même action, on écrit les WHEN successivement et l'action à exécuter à la suite du dernier WHEN.

Format :

```
EVALUATE      {TRUE, FALSE, variable}
              WHEN  {TRUE, FALSE, variable, valeur}      ALSO {TRUE, FALSE, variable}
              WHEN  {ANY, TRUE, FALSE, variable, valeur} ALSO {ANY, TRUE, FALSE, variable,valeur}
END-EVALUATE
```

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

a) Règles importantes à suivre :

- même nombre de sujets que d'objet dans chaque WHEN
- chaque sujet correspond à l'objet du même rang sujet et objet liés implicitement par un « = »

b) Exemple avec deux WHEN exécutant la même action :

```
EVALUATE Code

  WHEN 2
  WHEN 5  DISPLAY '2 ou 5'

  WHEN 6 THRU 9 DISPLAY 'entre 6 et 9'

END-EVALUATE
```

c) Exemple avec plusieurs sujets :

```
EVALUATE Code ALSO Nom

  WHEN 2 ALSO 'ROBERT' DISPLAY 'Ok.'

  WHEN 3 ALSO ANY DISPLAY 'Pas Ok.'

END-EVALUATE
```

11. Opérations sur fichier :

a) Assignation des fichiers logiques :

Format : **SELECT** fichier logique **TO** fichier physique
ORGANISATION SEQUENTIAL

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE CONTROL.
DATA DIVISION.
PROCEDURE DIVISION.

Exemple :

```
SELECT origine1 ASSIGN to SOURCE1
       ORGANISATION SEQUENTIAL
```

b) Format des enregistrements du fichier :

Format : **FD** fichier logique
LABEL RECORD STANDARD
RECORDING MODE F
BLOCK 0
DATA RECORD variable liée.

01 nom variable liée

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
FILE CONTROL.
DATA DIVISION.
FILE SECTION.
PROCEDURE DIVISION.

Précise le format des données qui seront lues ou écrites dans le fichier.

Exemple :

```
FD Nomfich     LABEL RECORD STANDARD
               RECORDING MODE F
               BLOCK 0
               DATA RECORD Enreg.

01 Enreg.
       05 Nom PIC X(20).
       05 Prenom PIC X(20).
```

Attention : un REDEFINES sur le niveau 01 n'est pas possible pour les fichiers.

c) Ouverture du fichier :

Format : **OPEN** {**INPUT**, **OUTPUT**, **I-O**, **EXTEND**} fichier logique

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
FILE CONTROL.
DATA DIVISION.
FILE SECTION.
PROCEDURE DIVISION.

Input : lecture
Output : Ecriture
I-O : Lecture-écriture
Extend : Ajout fin de fichier

Exemple :

```
OPEN INPUT Nomfich
```

d) Utilisation (lecture/écriture) et fermeture :

Format : **READ** fichier logique [**INTO** variable] [**AT END** instruction]

WRITE nom variable liée

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.
FILE CONTROL.

DATA DIVISION.

PROCEDURE DIVISION.

Lecture :

La lecture d'un fichier affecte par défaut la variable liée. L'instruction INTO permet de préciser la variable réceptrice des données.

L'instruction AT END permet de tester la fin de fichier. En affectant une variable « drapeau » après cette instruction, on peut savoir si le pointeur de fichier est arrivé à la fin.

Ecriture :

Permet d'écrire les données dans le fichier. Comme la variable liée est précisée dans la section DATA DIVISION, le système sait où écrire les données (pour le cas où plusieurs fichiers seraient ouverts).

Fermeture :

Format : **CLOSE** fichier logique

12. Quelques instructions :

ACCEPT

PROCEDURE DIVISION. Permet de saisir un texte ou un nombre au clavier et de le stocker dans une variable. Ex : `ACCEPT valeur.` (Microsoft). En Cobol, permet également de récupérer l'heure et la date système . Ex : `ACCEPT valeur FROM DATE.`

CONTINUE

PROCEDURE DIVISION. C'est une instruction sans fonction, qui permet de poursuivre le programme. Elle peut être utile lors de tests si l'on se trouve dans une condition qui doit comporter au moins une instruction.

DISPLAY

PROCEDURE DIVISION. Permet d'afficher un texte ou le contenu d'une variable. Après cette instruction, un retour chariot est automatiquement effectué. Pour afficher un texte, il faut le placer entre côtes. Le séparateur est l'espace. Ex : `DISPLAY 'Note : ' note '/20'`

INITIALIZE

PROCEDURE DIVISION. Met à zéro toute variable numérique indiquée ou affecte des espaces à une zone alphanumérique. Ex : `INITIALIZE note`

PERFORM

PROCEDURE DIVISION. Effectue un saut vers un paragraphe. Ex : `PERFORM SORTIE.` Ce saut peut-être répété jusqu'à ce qu'une condition soit remplie : Ex : `PERFORM SORTIE UNTIL FIN='VRAI'.`

SPACE/ZERO

DATA DIVISION/PROCEDURE DIVISION.
- Affecte une chaîne du nombre d'espace maximal qu'elle peut contenir. Ex : `MOVE SPACE TO Nom`
- Permet de questionner une variable. Ex : `IF Nom=SPACE THEN…`

STRING

PROCEDURE DIVISION.
Affecte la concaténation de une ou plusieurs chaînes dans une autre.

Format :	**STRING**	{Chaine, Variable}
		[{Chaine, Variable}]
	DELIMITED BY SIZE	
	INTO	Variable réceptrice

THRU

PROCEDURE DIVISION.
Cette instruction a un rôle de comparaison entre deux nombres. Elle est utilisée dans l'instruction EVALUATE, après un WHEN.
Ex : `EVALUATE Prix  WHEN 1 THRU 8 DISPLAY 'Trop Cher'.`
Ici, l'instruction WHEN comparera si « Prix » est compris entre 1 et 8.

13. Débuggage :

```
IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.

DATA DIVISION.

PROCEDURE DIVISION.
```

En ajoutant « WITH DEBUGGING MODE », on peut placer un « D » en colonne 7 devant les instructions. Celles-ci seront exécutées normalement après la compilation.

En rajoutant un astérisque devant « SOURCE-COMPUTER », toutes les instructions précédées du « D » seront ignorées.

14. Programme externe - le module :

Un module est un programme Cobol pouvant être appelé d'un autre programme et qui rend la main au programme principal à la fin de son exécution.

Il n'y a rien de spécifique à déclarer sinon des paramètres d'entrée et des paramètres de sortie communs aux deux sources.

Programme source :

```
IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

WORK-STORAGE SECTION.

77 Param1 PICTURE X(20).
77 Param2 PICTURE 99.

DATA DIVISION.

PROCEDURE DIVISION.

Call 'ProgMod' USING Param1, Param2

...

STOP RUN. ou GOBACK.
```

Module 'ProgMod' :

```
IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

LINKAGE SECTION.

77 Param1 PICTURE X(20).
77 Param2 PICTURE 99.

DATA DIVISION.

PROCEDURE DIVISION USING Param1, Param2.

...

GOBACK.
```

Les déclarations de la LINKAGE SECTION ne créent pas de nouvelles variables mais pointent sur les variables du programme principal.
Le nom des paramètres peut différer entre le programme et la section LINKAGE du module mais l'ordre des paramètres est primordial.

L'instruction GOBACK est nécessaire pour rendre la main au programme principal.

15. Programme externe - la clause COPY :

La clause COPY permet d'inclure, lors de la compilation, le contenu d'un autre source Cobol à l'emplacement de cette instruction et ce quelque soit le paragraphe ou le COPY se trouve.

Cette clause peut servir lorsque plusieurs programmes appellent un module et doivent lui passer des paramètres. Plutôt que de copier ces paramètres manuellement, la simple instruction « COPY nom de fichier » suffit.

Format : **COPY** fichier clause
[**REPLACING** ==(P)== **BY** == nom à rajouter==]

Exemple :

Fichier clause 'ProgClause' :

```
01 (P)Entree.
   05 Date-Entree(10).

01 (P)Sortie.
   05 Date-Sortie(10).
```

Fichier programme :

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORK-STORAGE SECTION.

```
COPY ProgClause
REPLACING ==(P)==
BY ==TEST-==
```

PROCEDURE DIVISION.

…

Fichier programme compilé:

IDENTIFICATION DIVISION.

ENVIRONMENT DIVISION.

DATA DIVISION.

WORK-STORAGE SECTION.

```
01 TEST-Entree.
   05 Date-Entree(10).

01 TEST-Sortie.
   05 Date-Sortie(10).
```

PROCEDURE DIVISION.

…

16. Opérations arithmétiques :

Elles s'effectuent au moyen des instructions ADD, SUBSTRACT, MULTIPLY et DIVIDE. Le verbe COMPUTE s'utilise surtout pour les calculs complexes qui nécessiteraient plusieurs ligne et même parfois des variables temporaires.

Type opération	Opération	Cobol
Addition	a = b + c	`ADD b,c GIVING a`
	b = b + c	`ADD c to B`
Soustraction	a = b – c	`SUBSTRACT c FROM b GIVING a`
	b = b – c	`SUBSTRACT c FROM b`
Multiplication	a = b * c	`MULTIPLY b BY c GIVING a`
	b = b * c	`MULTIPLY c BY b`
Division	a = b / c	`DIVIDE c INTO b GIVING a`
	b =b / c	`DIVIDE c INTO b`
	q = b DIV c	`DIVIDE b BY c GIVING q`
	r = b MOD c	`REMAINDER r`
Toutes	a = (a * b) + c	`COMPUTE a = ( a * b ) + c`

ON SIZE ERROR

La situation que l'on appelle SIZE ERROR est déclenchée lorsqu'un résultat déborde (partie entière) de la variable dans laquellle il doit aller.

Exemple :

```
77 Variable PIC 99 VALUE 90.
...
ADD  30  TO Variable
```

Normalement, « Variable » devrait devenir 120 mais il est de PICTURE 99. Si on ne spécifie rien, il fera l'addition et tronquera le chiffre à gauche. Cependant, on peut spécifier la clause ON SIZE ERROR de la facon suivante:

```
ADD 30 TO Variable
 ON SIZE ERROR DISPLAY 'Erreur, la taille de la variable est insuffisante'
```

La variable contient toujours 90. La clause « ON SIZE ERROR » pouvant contenir plusieurs énoncés, on doit terminer la fonction par END-fonction

Exemple :

```
ADD 30 TO Variable
 ON SIZE ERROR DISPLAY 'Erreur, la taille de la variable est insuffisante'
END-ADD.
```

© Eric Quillévéré 2001-02