

LANGAGE COBOL

Avertissement

Pour diverses raisons (maintenance de programmes, ajouts de sous-programmes, manque de temps pour une migration Cobol vers autre langage), la demande en programmeurs Cobol existe encore. dans le milieu bancaire, notamment au Luxembourg, un développeur sachant programmer ou ayant des bases en Cobol est apprécié et surtout recherché.

Dans ce cours certaines variantes dans le codage des instructions ont été volontairement omises par souci de clarté et pour éviter les instructions incitant à la rédaction de programmes mal structurés.

Le langage de référence

ACUCobol norme Cobol ANS 85

Editeur à utiliser : e-Cobol

Fichiers :

- **code source** : fichier de 8 caractères maxi + extension **.CBL** à ajouter lors de la sauvegarde du source
exemple : essai.CBL
- **compilation** : touches CTRL-F9 à partir de l'éditeur e-Cobol
Deux fichiers sont générés :
 - le fichier avec l'extension **.LST** contenant la liste du programme et ses erreurs éventuelles
 - le fichier avec l'extension **.OUT** contenant le programme compilé en pseudo-langage machine non exploitable sans le programme permettant son exécution
- **édition de liens** : touches MAJ-F9 à partir de l'éditeur e-Cobol
- **exécution** : touche F9 à partir de l'éditeur e-Cobol

CHAPITRE 1 : PRESENTATION GENERALE DU LANGAGE

1. HISTORIQUE

Le langage COBOL, acronyme de COmmon Business Oriented Language, c. à d. *langage commun orienté gestion* a été mis au point par un comité de la CODASYL (COntference on DAta SYstems Languages) entre 1958-60 à la demande du gouvernement américain.

Evolution du langage :

- 1960 : création du COBOL-60
- 1968 : standardisation par l'ANSI (American National Standards Institute) : COBOL ANS 68
- 1974 : nouvelle norme COBOL ANS 74
- 1985 : nouvelle norme COBOL ANS 85 : prise en compte de l'évolution de la programmation vers une programmation structurée et modulaire (limitation de l'instruction GO TO, clauses de fin d'instruction : END-...)
- 1989 : incorporation au COBOL 85 de fonctions intrinsèques (qui n'appartiennent qu'à Cobol).
- ???? : futur Cobol Orienté Objets existant à **normaliser**

2. POINTS FORTS DU LANGAGE

- Assez bonne indépendance de la machine utilisée (portabilité), à condition de s'abstenir d'utiliser des fonctionnalités spécifiques du compilateur, différentes de la norme ANSI.
- Diffusion importante du langage COBOL dans les entreprises, principalement les entreprises de grande taille ; on estime que 20 à 30% des lignes de code écrites le sont encore en COBOL.
 - COBOL est adapté aux problèmes de gestion (puissantes instructions d'E/S)
 - Ecriture proche du langage naturel (anglais) – on lui reproche d'ailleurs souvent son manque de concision.

3. Exemples préliminaires

Exemple 1 : programme minimal

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. exemple1.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
77 prenom PIC X(30).  
PROCEDURE DIVISION.  
debut-prog.  
    DISPLAY "Entrez votre prénom".  
    ACCEPT prenom.  
    DISPLAY "Bonne année, " prenom.  
    STOP RUN.
```

Exemple 2 :

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. exemple2.  
DATE-WRITTEN. 5/8/92.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
77 prix PIC 9(5).  
77 qte PIC 99.  
77 total PIC 9(7).  
PROCEDURE DIVISION.  
debut-prog.  
    DISPLAY "Prix : " NO ADVANCING.  
    ACCEPT prix.  
    DISPLAY "Quantité : " NO ADVANCING.  
    ACCEPT qte.  
*    Calcul du total  
    MULTIPLY prix BY qte GIVING total.  
    DISPLAY "Total = " total.  
fin-prog.  
    STOP RUN.  
END PROGRAM.
```

4. MISE EN PAGE

Contrairement à la plupart des autres langages, COBOL nécessite **une mise en page très rigoureuse**, due aux contraintes imposées autrefois par les cartes perforées.

- - la **colonne 7** peut contenir :
 - une ***** => ligne de commentaires
- seules les **colonnes 8 à 72** sont utilisées pour la partie utile du programme.
- on distingue la **marge A (colonne 8)** et la **marge B (colonne 12)**
 - Doivent débiter en **colonne A** les entêtes de **division**, de **section**, de **fin de programme**, les **noms de paragraphes**, les **indicateurs de niveau** tels **FD ...**, les nombres de niveau **77** et **01**.
 - **Tout le reste** sera écrit à partir de la **colonne B**.

5. STRUCTURE DU PROGRAMME

Tout programme COBOL est formé de **4 divisions maximum**, devant se **suivre obligatoirement** dans cet ordre :

- **IDENTIFICATION DIVISION.** : nom du programme (obligatoire), et éventuellement date de création.
- **ENVIRONMENT DIVISION.** : décrit le matériel utilisé et définit les liens entre les données et leur support physique. Cette division peut être vide.
- **DATA DIVISION.** : contient la description des données qui sont traitées par le programme.
- **PROCEDURE DIVISION.** : suite des instructions formant le corps du programme ; celui-ci peut être composé de un ou plusieurs paragraphes (un paragraphe est constitué d'un nom de paragraphe suivi d'un point et d'un espace au moins, et de zéro, une ou plusieurs phrases).

Une phrase se compose de une ou plusieurs **instructions COBOL** et **se termine par un point**.

Une **instruction commence** obligatoirement par un **mot réservé COBOL** et peut être constituée de plusieurs mots et de séparateurs formant une commande COBOL syntaxiquement correcte.

Plusieurs paragraphes peuvent eux-mêmes être **regroupés** en **sections**.

6. LES ELEMENTS DU LANGAGE

Les mots sont formés à partir des caractères alphanumériques ; un mot ne peut pas dépasser 30 caractères.

Il y a 4 catégories de mots :

- **les mots réservés** ou **mots-clés** : ont une signification précise pour le compilateur (ex. IDENTIFICATION, STOP, DISPLAY, ...)
- **les mots-utilisateur** : créés par l'utilisateur, servent à identifier des objets du programme ; on distingue :
 - les **noms de données** : utilisés en PROCEDURE DIVISION et déclarés en WORKING-STORAGE SECTION, ce sont les variables utilisées dans le programme.
 - les **noms-condition** : précisent les valeurs que peuvent prendre une donnée.
 - les **noms de paragraphe** et **noms de section** en PROCEDURE DIVISION : servent à identifier une séquence d'instructions.
- **les constantes** :
 - les **constantes figuratives** :
COBOL a attribué un nom à certaines valeurs spécifiques pouvant être prises par une variable ; ces mots peuvent être utilisés au singulier ou au pluriel :
 - ZERO, ZEROS, ZEROES = 0
 - SPACE(S) = un ou plusieurs espaces
 - HIGH-VALUE(S) = plus grande valeur possible dans le jeu de caract. utilisé
 - LOW-VALUE(S) = plus petite valeur possible dans le jeu de caract. utilisé
 - QUOTE(S) = un ou plusieurs caractères {"}
 - ALL = en combinaison avec un littéral, remplace toutes les occurrences d'une variable par ce littéral
 - les **constantes littérales** :
 - Numériques : chiffres de 0 à 9, signe + ou -, point décimal
 - Non-numériques : tous les caractères écrits entre " "
- **les opérateurs** :
 - **arithmétiques** : + - * / **
 - **relationnels** :
 - > ou GREATER THAN
 - < ou LESS THAN
 - >= ou NOT < ou NOT LESS THAN
 - <= ou NOT > ou NOT GREATER THAN
 - = ou EQUAL TO
 - NOT =
 - **logiques** : AND, OR, NOT

7. CONVENTIONS SYNTAXIQUES

Pour la présentation de la syntaxe du langage, nous adopterons les conventions suivantes :

- Les **mots réservés** sont écrits en **majuscules**.
- Les **informations** fournies par le **programmeur** sont écrites en **minuscules**.
- Les **accolades** indiquent un choix possible entre **plusieurs options**.
- Les **crochets** indiquent un **élément facultatif**.
- Les **points de suspension** signifient qu'il est possible de **répéter la spécification** les précédant.

CHAPITRE 2 : LE LANGAGE

1. DECLARATION DE VARIABLE

Définition de la variable en DATA DIVISION, WORKING-STORAGE SECTION ; 3 possibilités selon qu'on a affaire à une **donnée élémentaire** (niveau 77), une **donnée structurée** (niveau 01), un **nom-condition** (niveau 88). Ces 2 dernières seront examinées dans le détail ultérieurement).

77 **nom-variable** **PIC type** [**VALUE valeur**] [**USAGE codage-memoire**] .

Exemples

```
77 prenom      PIC XXXXXXXXXXXX•
77 prenom      PIC X(10) •
77 titre       PIC A(20)  VALUE "Liste des employés"•
77 qte-en-stock PIC 9(5)  VALUE ZERO  USAGE BINARY•
77 piPIC       9V9(4)    VALUE 3.1416  USAGE COMP•
77 ligne-vide  PIC X(80)  VALUE SPACES•
77 traitPIC    X(80)     VALUE ALL "-"•
```

Les données **alphabétiques** et l'**espace** (PIC A) et alphanumériques (PIC X) sont **alignées à gauche**.
Les données **numériques** (PIC 9) sont **alignées à droite**.

La clause **USAGE** permet au programmeur de spécifier la façon de représenter la valeur d'une **variable numérique** dans la mémoire (format interne des données) :

- Pour le **type X et A**, l'**USAGE** est toujours **DISPLAY**;
- Pour les **variables numériques** (9), l'**USAGE** est **DISPLAY** par défaut.

Les différentes possibilités sont :

- **USAGE DISPLAY** : 1 code ASCII par chiffre.
- **USAGE COMP** ou **BINARY** : format binaire favorisant les calculs, dépend du compilateur ;

Exemple

```
77 qte      PIC 9(3)  VALUE 152•
```

→ en binaire = 1001 1000, soit 8 bits => 1 octet suffit au lieu de 3 en usage Display !

- **USAGE PACKED-DECIMAL** : décimal condensé, c. à d. que chaque chiffre décimal est représenté par un demi-octet : le 0 décimal correspond à 0000 en binaire, le 9 décimal à 1001.

Exemple

```
77 qte      PIC 9(3)  VALUE 152•
```

→ en décimal condensé = 0001 0101 0010 => 1 octet et demi suffirait

2. LES INSTRUCTIONS DE BASE

2.1. DISPLAY (Afficher)

Cette instruction permet l'affichage de littéraux et/ou du contenu de données. Deux formats sont disponibles : le 2ème format permet le positionnement à l'écran mais il ne permet l'affichage que d'une donnée à la fois.

```
DISPLAY donnée1 donnée2 [NO ADVANCING]
DISPLAY donnée [AT ...] [WITH ...]
```

- les éléments à afficher doivent être séparés par un espace.
- si le paramètre NO ADVANCING n'est pas présent, le curseur se positionne au début de la ligne suivante après l'affichage.

Exemples

```
DISPLAY "Bonjour" prenom
```

```
DISPLAY "Entrez votre nom : " NO ADVANCING
```

```
DISPLAY "Azerty" AT 0514 WITH HIGHLIGHT
```

```
DISPLAY "Menu du jour" AT LINE ligne COL colonne WITH BLANK SCREEN
```

2.2. ACCEPT (Saisir)

Permet d'introduire des données provenant généralement du clavier dans une variable utilisateur.

- pour les données numériques, les caractères saisis sont cadrés à gauche à l'écran.
- le compilateur n'affiche pas automatiquement le caractère décimal (.) lors de la saisie.
- le compilateur ne complète pas avec des espaces ou des zéros quand tous les caractères ne sont pas saisis.

Nous verrons **ultérieurement** une autre méthode (**SCREEN SECTION**) plus performante.

```
ACCEPT nom-donnee
ACCEPT nom-donnee FROM DATE           date système au format AAMMJJ
ACCEPT nom-donnee FROM DAY            date système au format AAJJJ (JJJ = n° jour dans l'année)
ACCEPT nom-donnee FROM DAY-OF-WEEK   fournit le numéro du jour dans la semaine (lundi =1, ...)
ACCEPT nom-donnee FROM TIME          heure système au format HHMMSSCC
```

SIGNIFICATION DES PRINCIPAUX PARAMETRES (qui suivent le mot réservé WITH)

BEEP	émet un signal sonore lors de la saisie/affichage
BLINK	la donnée clignote
FULL	oblige à remplir la zone de saisie
PROMPT	affichage d'un contenu de zone avant saisie ; ex. PROMPT "?" remplit la zone avec des ???
REQUIRED	oblige à saisir au moins un caractère dans la zone
NO-ECHO	saisie sans affichage, "en aveugle"
BLANK SCREEN	efface l'écran avant affichage
BLANK LINE	efface la ligne avant affichage
REVERSE-VIDEO	affiche en inversion vidéo
DISPLAY donnée AT ... WITH BLANK SCREEN	effacement de l'écran lors du premier affichage

2.3. MOVE (instruction d'affectation ←)

La clause VALUE déjà vue permet d'affecter une valeur initiale à une variable lors de la déclaration en WORKING-STORAGE SECTION ; mais pour affecter une valeur à une donnée en cours de programme il faut utiliser l'instruction MOVE.

MOVE nom-donnee1 **TO** nom-donnee2 [nom-donnee-3 ...]

Exemples

```
MOVE 0 TO qte-cdee qte-en-stock qte-livree
MOVE "azerty" TO nom
MOVE qte-cdee TO qte
```

Pour que l'instruction MOVE fonctionne correctement, il faut que **l'élément émetteur et l'élément récepteur** soient de **types compatibles** ; attention aux troncatures quand les images respectives ne sont pas de même longueur ; voici quelques exemples d'affectation :

ELEMENT EMETTEUR		ELEMENT RECEPTEUR	
Picture	Valeur	Picture	Valeur
99V99	12.34	999V99	012.34
99V99	12.34	99V9	12.3
9(4)	1234	9(3)	234
9V99	1.2	99V999	01.200
9V99	1.23	99.99	01.23
9	1	V99	00
9999	12	X(3)	12Δ
9999	1234	X(3)	123
X(4)	1234	99.99	34.00
X(4)	1234	9(4)	1234

Δ représente un espace

2.4. INSTRUCTIONS DE CALCUL

ADDITION

ADD nom-donnee1 nom-donnee2 ... **GIVING** nom-donnee-n

Exemple : **ADD** 12 X Y **GIVING** Z $\rightarrow Z \leftarrow 12 + X + Y$

ADDITION "avec cumul"

ADD nom-donnee1 nom-donnee2 ... **TO** nom-donnee-n

Exemple : **ADD** 12 X Y 24.67 **TO** Z $\rightarrow Z \leftarrow Z + 12 + X + Y + 24,67$

SOUSTRACTION

SUBTRACT nom-donnee1 nom-donnee2 ... **FROM** nom-donnee-n [**GIVING** nom_donnee-q]

exemples : **SUBTRACT** X Y 23 **FROM** Z $\rightarrow Z \leftarrow Z - (X + Y + 23)$
SUBTRACT X **FROM** Z **GIVING** Y $\rightarrow Y \leftarrow Z - X$

MULTIPLICATION

MULTIPLY nom-donnee1 **BY** nom-donnee2 [**GIVING** nom_donnee-3]

exemples : **MULTIPLY** 0.196 **BY** montant \rightarrow montant \leftarrow montant \times 0,196
MULTIPLY 0.196 **BY** montant **GIVING** tva \rightarrow tva \leftarrow 0,196 \times montant

DIVISION

DIVIDE nom-donnee1 **BY** nom-donnee2 [**GIVING** nom-donnee-n]

DIVIDE nom-donnee1 **INTO** nom-donnee2

exemples : **DIVIDE** a **BY** b $\rightarrow b \leftarrow a / b$
DIVIDE a **BY** b **GIVING** c $\rightarrow c \leftarrow a / b$
DIVIDE a **INTO** b $\rightarrow b \leftarrow b / a$

Nota : Les 4 instructions de calcul peuvent être terminées par la clause **ROUNDED** (arrondi).

COMPUTE

COMPUTE nom-donnee [**ROUNDED**] = expression mathématique

exemple : **COMPUTE** a = ((b ** c) + (21 * 18.6)) / 2

● **Remarque importante :** Dans ces instructions de calcul, les données doivent toujours être des données numériques (format PIC 9...), et non **pas des données d'édition** sauf après **GIVING**.

MODULO

Instruction **DIVIDE** avec obligatoirement la clause **GIVING** suivie de la clause **REMAINDER**.

exemple : **DIVIDE** 11 **BY** 3 **GIVING** quotient **REMAINDER** reste \rightarrow reste \leftarrow 11 MOD 3

2.5. LES INSTRUCTIONS DE FIN

L'instruction de fin de programme : **STOP RUN•**

Cette instruction obligatoire provoque l'arrêt de l'exécution du programme et rend le contrôle au système d'exploitation ; il n'est pas interdit d'utiliser plusieurs instructions STOP RUN dans un même programme, mais cette pratique est **fortement déconseillée**.

L'en-tête de fin de programme : **END PROGRAM•**

Cette en-tête (ce n'est pas une instruction) facultative sert à délimiter la fin du programme identifié dans le paragraphe PROGRAM-ID. nom-prog. Cet en-tête sert principalement à délimiter des sous-programmes imbriqués (cf § **Sous-programmes**).

L'instruction de sortie de sous-programme : **EXIT PROGRAM•**

Cette instruction indique la fin logique d'un sous-programme ou procédure appelée ; elle rend le contrôle au programme appelant. (cf § **Sous-programmes**).

3. LA STRUCTURE ALTERNATIVE

3.1. SI <condition> ALORS <instructions> SINON <autres-instructions> FINSI

IF condition THEN instr-1 [instr-2 ...] [ELSE instr-3 [instr-4 ...]] END-IF•

La clause **END-IF** termine l'instruction **IF**. Les instructions **IF** peuvent être imbriquées.

Attention aux erreurs : si vous mettez un point entre instruction-3 et instruction-4 par ex., l'instruction-4 sera exécutée quelle que soit la condition !

Exemple

```
IF candidat = "O" OR "o"
  DISPLAY "Age du candidat : "
  ACCEPT age
  DISPLAY "Casier vierge ? (O/N) : "
  ACCEPT casier-vierge
  IF age >= 21 AND (casier-vierge = "O" OR "o")
    DISPLAY "Candidature valide"
  ELSE
    DISPLAY "Candidature non valide"
  END-IF
END-IF•
```

3.2. SELON CAS FAIRE FinCAS

Principe : les **WHEN** sont examinés séquentiellement ; dès que l'un d'eux est vérifié, l'instruction associée est exécutée et le programme se branche à l'instruction qui suit **END-EVALUATE**.

Exemples :

DISPLAY "Entrez votre choix (1 à 8) : " **NO ADVANCING**

ACCEPT choix

EVALUATE choix

WHEN 1

PERFORM affichage

WHEN 2

PERFORM calculs

WHEN 3 THRU 8

DISPLAY "Modules non encore disponibles"

WHEN OTHER

DISPLAY "Choix non valide"

END-EVALUATE•

DISPLAY "Entrez votre Chiffre d'affaires : " **NO ADVANCING**

ACCEPT ca

EVALUATE TRUE

WHEN (ca > 0) **AND** (ca < 150000)

DISPLAY "Résultats médiocres"

WHEN (ca >= 150000) **AND** (ca < 300000)

DISPLAY "Résultats corrects"

WHEN ca >= 300000

DISPLAY "Excellents résultats"

WHEN OTHER

DISPLAY "Valeur incorrecte"

END-EVALUATE•

4. LA STRUCTURE REPETITIVE

4.1. NOTION DE PARAGRAPHE (BLOC D'INSTRUCTIONS)

Un paragraphe (ou bloc) est un ensemble d'instructions délimitées par un nom de paragraphe obligatoirement en marge A. Le paragraphe s'arrête où commence un nouveau nom de paragraphe.

Exemple :

```
... instructions principales
STOP RUN●
BLOC1●      début du bloc ou paragraphe BLOC1
<instructions>●
BLOC2●      début du bloc ou paragraphe BLOC2
<instructions>●
```

4.2. EXECUTION SANS REPETITIVE D'UN OU PLUSIEURS PARAGRAPHES

```
PERFORM paragraphe-j [ THRU paragraphe-k ]
```

Exemple :

```
PROCEDURE DIVISION.
DEBUT●      marque de début
<instructions>
PERFORM saisie●      exécution des instructions 1 et 2
<instructions>
PERFORM saisie THRU affichage●      exécution des instructions 1 à 6
<instructions>
STOP RUN●
SAISIE●     début du bloc ou paragraphe SAISIE
instruction_1.
instruction_2.
CREATION●   début du bloc ou paragraphe CREATION
instruction_3.
instruction_4.
instruction_5.
AFFICHAGE●  début du bloc ou paragraphe AFFICHAGE
instruction_6.
```

- Les **blocs appelés** doivent toujours **se trouver après le STOP RUN** car une fois le bloc exécuté, le programme se rebranche à l'instruction qui suit l'appel (seul le STOP RUN met fin au pg).

4.3. REPETER N FOIS

```
PERFORM n TIMES  
  
  <instructions>  
  
END-PERFORM
```

Exemple :

```
DEBUT●  
  PERFORM 12 TIMES  
    instruction_3  
    instruction_4  
    instruction_5  
  END-PERFORM●  
STOP RUN●
```

4.4. Instruction TANT QUE ... FAIRE ... FTQ

```
PERFORM UNTIL condition-de-sortie  
  
  <instructions>  
  
END-PERFORM
```

☛ Pour traduire la *condition* du TANT QUE d'un algorithme, en raison de la syntaxe propre au COBOL, il faut **inverser la condition** (traduire la condition de *continuation* par une condition *de sortie*).

Algorithme	Cobol
Tant que N < 10 faire	PERFORM UNTIL N >= 10
instruction_1	instruction_1
instruction_2	instruction_2
Fin TQ	END-PERFORM●

4.5. Instructions REPETER ... JUSQU'A...

```
PERFORM WITH TEST AFTER UNTIL condition-de-sortie  
  
  <instructions>  
  
END-PERFORM
```

La clause **TEST AFTER** permet de tester la condition qu'après avoir exécuté une 1ère fois le paragraphe.

Algorithme	Cobol
Répéter	PERFORM WITH TEST AFTER UNTIL choix = "O"
instruction_1	instruction_1
instruction_2	instruction_2
Jusqu'à choix = 'O'	END-PERFORM●

4.6. Instruction POUR ... FAIRE ... FIN POUR

```
PERFORM VARYING nom-donnee1 FROM nom-donnee2 BY nom-donnee3 UNTIL condition-de-sortie
<instructions>
END-PERFORM
```

Exemples :

Algo.	Cobol
<u>Pour</u> i de 1 à 50 faire instruction_1 instruction_2 <u>FinPour</u>	PERFORM VARYING i FROM 1 BY 1 UNTIL i > 50 instruction_1 instruction_2 END-PERFORM•
<u>Pour</u> i de 1 à 25 (pas 2) instruction_1 instruction_2 <u>FinPour</u>	PERFORM VARYING i FROM 1 BY 2 UNTIL i > 25 instruction_1 instruction_2 END-PERFORM•

5. COMPLEMENTS AUX VARIABLES COBOL

5.1. Les picture standards : A X 9 S V

- PIC A... → variable de type alphabétique (valeurs ∈ [A...Z, a...z, espace])
- PIC X... → variable de type alphanumérique (valeurs ∈ [tous caractères ASCII])
- PIC ...9... → variable numérique
- PIC S9... → variable numérique **signée** (exemple 77 V PIC S9(3) → V ∈ [-999..+999])
- PIC 9..V9.. → variable **décimale** (réel) (exemple 77 V PIC S9V9(4) → V ∈ [-9,9999..+9,9999])

5.2. Les PICTURE d'édition : Z * B 0 , . + - CR DB

Ces PICTURES d'édition peuvent s'ajouter aux PICTURES standards afin de faciliter les éditions (affichages ou impressions) ; ils sont principalement utilisés pour les variables numériques et permettent de répondre à des besoins particuliers de présentation des données.

☞ Une variable est dite **variable d'édition** si elle contient au moins un PICTURE d'édition.

☞ Une variable d'édition exige l'USAGE DISPLAY.

☛ Une variable d'édition **ne peut pas être utilisée DANS un calcul**.

Si on a : 77 N PIC ZZ9., l'instruction **ADD 1 TO N** est erronée !

Si on a : 77 M PIC ZZ9., l'instruction **ADD 5 TO 7 GIVING M** est juste.

Principe d'utilisation : on effectue les calculs avec des variables standards, puis on "MOVE" ces variables dans des variables d'édition adéquates avant de les éditer.

Edition avec remplacement des zéros par des espaces (Z) ou par des astérisques (*)

Cette méthode permet de **ne pas éditer les zéros à gauche et de les remplacer, soit par des espaces** (suppression des zéros non significatifs), **soit par des *** (protection des sommes sur les chèques et mandats).

Exemples :

une variable **PIC Z(4)** contient :

- la valeur 0015, cela donne $\Delta\Delta15$ (Δ =espace)
- la valeur 0000, cela donne $\Delta\Delta\Delta\Delta$

une variable **PIC ****9** contient :

- la valeur 00006, cela donne ******6**
- la valeur 00000, cela donne ******0**

Edition avec insertion simple : , (virgule) B (espace) 0 (zéro)

Chaque caractère inséré compte pour une position sur la ligne d'édition.

Exemples :

une variable **PIC 9(5)V99** contient :

- 12345**V**67 (**V**=position virtuelle de la virgule) que l'on "MOVE" dans une variable **PIC 99B999,99** donne $12\Delta345,67$
- 00045**V**67 que l'on "MOVE" dans une variable **PIC ZZBZZ9,99** donne $\Delta\Delta\Delta\Delta45,67$

Edition avec insertion spéciale : séparateur décimal et séparateur des milliers

Cette insertion utilise le point décimal utilisé aux USA pour représenter notre virgule. Ce point ne peut être inséré qu'à l'emplacement de la virgule virtuelle.

Exemples :

une variable **PIC 9(4)V99** contenant 1234**V**56

- que l'on "MOVE" dans une variable **PIC 9(4).99** donne 1234.56
- que l'on "MOVE" dans une variable **PIC 9,999.99** donne 1,234.56

Cette forme n'est pas intéressante en France. Mais il existe une clause DECIMAL-POINT IS COMMA acceptée par la plupart des compilateurs, qui permet d'inverser le rôle du point décimal et de la virgule séparateur des milliers ; cette clause doit figurer comme-suit :

ENVIRONMENT DIVISION•

CONFIGURATION SECTION•

SPECIAL-NAMES•

DECIMAL-POINT IS COMMA•

Exemples :

Si cette clause est présente, la variable **PIC 9(4)V99** contenant 1234**V**56

- que l'on "MOVE" dans une variable **PIC 9(4),99** donne 1234,56
- que l'on "MOVE" dans une variable **PIC 9.999,99** donne 1.234,56

Avec cette clause, la virgule représente le caractère décimal même dans les opérations arithmétiques.

Edition avec insertion fixe : + (plus) - (moins) CR (crédit) DB (débit)

Les caractères + ou - doivent être placés à l'extrême gauche du PIC :

+ donne :

- + si la donnée éditée est positive ou nulle
- si la donnée est négative

-donne :

- espace si la donnée est positive ou nulle
- si la donnée est négative

Les caractères CR ou DB doivent être placés à l'extrême droite de la PIC :

CR donne

- 2 espaces si la donnée est positive ou nulle
- CR si la donnée est négative

DB donne

- 2 espaces si la donnée est positive ou nulle
- DB si la donnée est négative

La clause **BLANK WHEN ZERO**, ajoutée après la PICTURE, permet d'éliminer tous les caractères d'insertion parasites lorsque la donnée d'édition est nulle.

5.3. LES VARIABLES STRUCTUREES

Déclaration en Algo.

```
VEHICULE (  
  NUM_IMMAT (  
    NUM1 : entier  
    NUM2 : chaîne(3)  
    DEPT : chaîne(2)  
  )  
  MARQUE : chaîne(30)  
  MODELE : chaîne(30)  
  DATE_ACHAT : date  
)
```

Déclaration en Cobol

```
01 VEHICULE•  
  02 NUM-IMMAT•  
    03 NUM1 PIC 9(4)•  
    03 NUM2 PIC A(3)•  
    03 DEPT PIC A(2)•  
  02 MARQUE PIC X(30)•  
  02 MODELE PIC X(30)•  
  02 DATE-ACHAT•  
    03 JOUR PIC 9(2)•  
    03 FILLER PIC X VALUE "/"•  
    03 MOIS PIC 9(2)•  
    03 FILLER PIC X VALUE "/"•  
    03 AN PIC 9(2)•
```

☞ Les **nombre-niveaux** sont compris entre 01 et 49 ; l'incrément d'un sous-niveau peut être supérieur à 1.

Exemple :

```
01 TRUC•  
  05 MACHIN•  
    10 BIDULE• etc.
```

☞ Le nom de donnée **FILLER** permet de définir, dans une donnée structurée, une donnée élémentaire à laquelle il est prévu de **ne pas avoir accès** ; évite d'avoir à imaginer un nom de variable.

☛ La **structure principale** est toujours **considérée** comme étant de **type alphanumérique**, même si les variables élémentaires qui la constituent sont de type numérique !

5.4. LES TABLEAUX

Déclaration en Algo.

T_PRIX : tableau [1..12] de réel

T_REPRES : tableau [1..30]

de (

NOM : chaîne(20)

SAL : réel

)

T_CA : tableau [1..30, 1..12] de entier

Déclaration en Cobol (en Working-Storage Section)

01 T-PRIX●

02 PRIX OCCURS 12 PIC 9(8)V99●

01 T●

02 T-REPRES OCCURS 30●

03 NOM PIC X(20)●

03 SAL PIC 9(6)V99●

01 T-CA●

02 T-REP OCCURS 30●

03 CA OCCURS 12 PIC 9(6)●

☛ La clause **OCCURS** ne peut pas figurer au niveau 01

☛ Le 1er élément a toujours l'indice 1

ACCES A UN ELEMENT D'UN TABLEAU

Tableau à une dimension

élément (indice) ou élément OF tableau (indice)

Exemple : PRIX (3) NOM (14) ou NOM OF T-REP (14) SAL (8)

Tableau à plusieurs dimensions :

élément (ind1, ind2, ...)

Exemple : CA (28, 1) contient le C.A. du 28ème représentant en janvier

INITIALISATION D'UN TABLEAU

- Initialisation statique à une valeur identique pour toutes les cases

Exemple :

01 T●

02 T-REPRES OCCURS 30●

03 NOM PIC X(20) VALUE SPACES●

03 SAL PIC 9(6)V99 VALUE 0●

- Initialisation statique avec des valeurs différentes : clause **REDEFINES**

Exemple :

01 JOURS-MOIS VALUE "312831303130313130313031"●

01 REDEFINES JOURS-MOIS●

02 NB-JOURS OCCURS 12 PIC 99●

Résultat : NB-JOURS (1) vaudra 31, NB-JOURS (2) vaudra 28, ... NB-JOURS (12) vaudra 31.

- Initialisation dynamique

En utilisant des instructions de saisie ou d'affectation dans la procédure division.

6. LES SOUS-PROGRAMMES (Procédures sans paramètres)

6.1. Instructions

Le programme appelant appelle le sous-programme (programme appelé) avec l'instruction :

```
CALL "sous-prog".
```

sous-prog désigne le nom du programme appelé. Ce nom doit apparaître dans le paragraphe PROGRAM-ID du sous-programme.

L'exécution d'un sous-programme doit se terminer par l'instruction :

```
EXIT PROGRAM.
```

6.2. Sous-programme

Un sous-programme se trouve dans un autre fichier qui sera compilé séparément. Dans ce cas on obtient un programme-objet pour le programme appelant et un programme-objet pour le sous-programme ; le sous-programme devra alors être lié au programme appelant lors de l'édition de liens (**linkage**).

6.3. Exemples de sous-programme

Fichier PRINCIPA.CBL

```
IDENTIFICATION DIVISION●  
PROGRAM-ID● PRINCIPA●  
PROCEDURE DIVISION●  
DEBUT●  
    DISPLAY "Je vais appeler le sous-programme"●  
    CALL "TIRETS"●  
    DISPLAY "Je suis revenu dans le programme appelant"●  
STOP RUN●
```

Fichier TIRETS.CBL

```
IDENTIFICATION DIVISION●  
PROGRAM-ID● TIRETS●  
PROCEDURE DIVISION●  
DEBUT●  
    DISPLAY "-----"●  
EXIT PROGRAM●
```

6.4. LES SOUS-PROGRAMMES AVEC PARAMETRES (Procédures)

```
CALL "sous-prog" USING [ BY { REFERENCE, CONTENT } ] p1 p2 ... ..
```

BY REFERENCE = passage par adresse (c'est le mode par défaut en COBOL).

BY CONTENT = passage par valeur.

Les **paramètres formels** seront **déclarés** dans une **section spéciale de la DATA DIVISION** :
la **LINKAGE-SECTION**

Celle-ci devra **suivre la WORKING-STORAGE SECTION** si elle existe.

La liste des **paramètres formels** doit **figurer dans l'en-tête** de la PROCEDURE DIVISION de la procédure:

PROCEDURE DIVISION USING pf1 pf2 ...

Exemple :

Fichier PRINCIPA.CBL

```
IDENTIFICATION DIVISION●
PROGRAM-ID. PRINCIPAL●
DATA DIVISION●
77  A      PIC S99●
77  B      PIC S99●
77  MAX    PIC S99●
PROCEDURE DIVISION●
DEBUT●
    DISPLAY "ENTREZ 2 ENTIERS"●
    ACCEPT A●
    ACCEPT B●
    CALL "MAXIMUM" USING BY CONTENT A B BY REFERENCE MAX●
    DISPLAY "LE MAXIMUM EST " MAX●
    STOP RUN●
```

Fichier MAXIMUM.CBL

```
IDENTIFICATION DIVISION●
PROGRAM-ID. MAXIMUM●
DATA DIVISION●
LINKAGE SECTION●
77  NB1    PIC S99●
77  NB2    PIC S99●
77  NBMAX  PIC S99●
PROCEDURE DIVISION USING NB1 NB2 NBMAX●
DEBUT●
    IF NB1 > NB2
        MOVE NB1 TO NBMAX
    ELSE
        MOVE NB2 TO NBMAX
    END-IF●
    EXIT PROGRAM●
```

7. LES INSTRUCTIONS DE MANIPULATION DE CHAINES

La puissance de définition des données en Cobol (tailles prédéfinies des PIC combinées aux données structurées) fait que nous aurons rarement à utiliser ces instructions.

Parmi les plus utiles :

```
INSPECT ch TALLYING nb FOR ALL car.
```

Exemple :

si **ch** contient "ABRACADABRA" et **car** contient "A", alors **nb** recevra **5**

```
INSPECT ch REPLACING ALL car1 BY car2.
```

Exemple :

si **ch** contient "ELECTRICITE", **car1** contient "E" et **car2** contient "I"
alors **ch** contiendra "ILICTRICITI".

```
STRING ch1 ch2 ... INTO ch3.      concaténation de chaînes
```

Exemple :

```
STRING ch1 "-" ch2 INTO ch3
```

si **ch1** contient "Azerty" et **ch2** contient "Uiop", alors **ch3** recevra "Azerty-Uiop" .

```
UNSTRING ch1 DELIMITED BY [ALL] ch2 INTO ch3 ch4 ...
```

Exemple :

```
UNSTRING "Ces 3 mots" DELIMITED BY ALL SPACE INTO ch1 ch2 ch3
```

ch1 contiendra "Ces", ch2 contiendra "3", ch3 contiendra "mots".

```
ACCEPT OMITTED
```

Attend l'appui sur la touche Entrée.

8. LES FICHIERS

8.1. Introduction

Un fichier COBOL se décompose en enregistrements. Un enregistrement se subdivise en rubriques.

Un sous-enregistrement est une collection d'informations homogènes, subdivisions d'un enregistrement, et subdivisées elles-mêmes en rubriques.

Chaque rubrique doit être associée à son type , sa longueur, sa signification.

Un fichier est en général consulté ou mis à jour

Les types de fichiers

Fichier **permanent** : fichier conservé au delà de la durée d'exécution du programme qui l'a créé

Fichier **paramètre** : fichier qui sert à stocker des paramètres (exemple : fichier des taux de TVA)

Fichier **d'archive** : fichier qui sert de copie de sauvegarde à un fichier permanent

Fichier **mouvement** : fichier qui provoque la mise à jour de fichiers permanents (exemple : saisie des commandes de la journée dans un fichier mouvement, mise à jour du fichier permanent en fin de journée)

Fichier **manoeuvre** : fichier intermédiaire qui permet le passage d'informations entre les des traitements successifs.

Organisation

Séquentielle : on accède aux enregistrements les uns après les autres.

Relative : on peut accéder directement à un enregistrement en utilisant son indicatif.

Séquentielle indexée : les fichiers sont couplés à une hiérarchie de tables contenues dans un fichier index.

8.2. Les fichiers séquentiels

Un fichier d'organisation séquentielle ne peut être ouvert qu'en création (output), en ajout (extend) ou en lecture (input).

On ne peut ni supprimer, ni modifier un enregistrement.

Déclaration de fichier

```
ENVIRONMENT DIVISION•  
INPUT-OUTPUT SECTION•  
FILE-CONTROL•  
    SELECT FICHER-CLIENTS ASSIGN TO FILE-NAME  
    ORGANIZATION SEQUENTIAL  
    ACCESS SEQUENTIAL  
    FILE STATUS IS FILE-STATUS•
```

Déclaration du nom de fichier (en Working-Storage Section)

```
77 FILE-NAME    PIC X(...)•
```

Déclaration d'enregistrement

```
DATA DIVISION•  
FILE SECTION•  
FD FICHER-CLIENT•  
01 F1-ENR•  
    02 F1-NOM          PIC X(30)•  
    02 F1-ADRESSE     PIC X(40)•  
    02 F1-TELEPHONE  PIC 9(10)•
```

Ouverture de fichier

```
OPEN INPUT FICHER-CLIENT•  
OPEN OUTPUT FICHER-CLIENT•  
OPEN EXTEND FICHER-CLIENT•
```

Lecture, écriture

```
READ FICHER-CLIENT  
    AT END MOVE VRAI TO FIN-FICHER-CLIENT  
    NOT AT END PERFORM TRAITER-ENREGISTREMENT-CLIENT  
END-READ•
```

```
WRITE F1-ENR.
```

Fermeture de fichier

```
CLOSE FICHER-CLIENT.
```

Gestion des erreurs d'accès à un fichier (valable pour toutes les organisations)

Cette gestion peut se configurer pour toutes les organisations de fichiers seulement si la clause FILE STATUS est présente dans la déclaration du fichier à gérer.

WORKING-STORAGE SECTION•

01 FILE-STATUS PIC X(2)•

88 FILE-NOT-FOUND VALUE "35"•

01 REDEFINES FILE-STATUS PIC X•

88 FILE-OKAY VALUE "0"•

...

PROCEDURE DIVISION•

DECLARATIVES•

FILE-ERROR SECTION•

USE AFTER STANDARD ERROR PROCEDURE ON FICHIER-CLIENT•

END DECLARATIVES•

8.3. Fichiers indexés

Dans un fichier d'organisation *indexée*, les enregistrements sont identifiés par la valeur de leur *clé*. La clé est un champ de l'enregistrement.

Trois *modes d'accès* sont possibles avec les fichiers indexés

- **accès aléatoire (random)** : l'accès (read, write, rewrite, delete) se fait en précisant la clé de l'enregistrement.
- **accès séquentiel (séquentiel)** l'accès se fait dans l'ordre des enregistrements. L'instruction start permet de se repositionner sur un enregistrement dont on connaît la clé.
- **accès dynamique (dynamic)** combinaison des deux modes ci-dessus.

Déclaration d'un fichier

ENVIRONMENT DIVISION●

INPUT-OUTPUT-SECTION●

FILE-CONTROL●

SELECT F-VOITURES

ASSIGN TO FILE-NAME

ORGANIZATION INDEXED

ACCESS RANDOM

RECORD KEY PLAQUE●

...

DATA DIVISION●

FILE SECTION●

FD F-VOITURES●

01 ENR-VOITURE●

02 PLAQUE PIC X(10)●

02 MARQUE PIC X(10)●

02 MODELE PIC X(10)●

02 COULEUR PIC X(10)●

8.3.1. Accès aléatoire (random)

OPEN

Un fichier indexé peut être ouvert dans un des modes suivants : **INPUT**, **OUTPUT**, ou **I-O**.

Exemple : **OPEN I-O F-VOITURES**•

READ

```
MOVE "123 ABC 45" TO PLAQUE
READ F-VOITURES
INVALID KEY
    DISPLAY "PAS DE VOITURE DANS LE FICHIER"
NOT INVALID KEY
    DISPLAY "MODELE = " MODELE
END-READ
```

WRITE

```
MOVE "123 ABC 45" TO PLAQUE
MOVE "SIMCA" TO MARQUE
MOVE "1000" TO MODELE
...
WRITE ENR-VOITURE
INVALID KEY
    DISPLAY "VOITURE DEJA ENREGISTREE."
NOT INVALID KEY
    DISPLAY "OK."
END-WRITE
```

REWRITE

```
MOVE "123 ABC 45" TO PLAQUE.
READ F-VOITURES
INVALID KEY
    DISPLAY "VOITURE ABSENTE"
NOT INVALID KEY
    MOVE "BLEU" TO COULEUR
    REWRITE ENR-VOITURE
    INVALID KEY
        DISPLAY "**** ERREUR INTERNE (REWRITE)"
    NOT INVALID-KEY
        DISPLAY "OK."
    END-REWRITE
END-READ
```

DELETE

```
MOVE "123 ABC 45" TO PLAQUE.
DELETE F-VOITURES
INVALID KEY
    DISPLAY "VOITURE ABSENTE"
NOT INVALID KEY
    DISPLAY "OK"
END-DELETE
```

CLOSE

```
CLOSE F-VOITURES
```

8.3.2. Accès séquentiel (séquentiel)

En accès séquentiel, l'ouverture peut se faire dans l'un des 4 modes : **INPUT, OUTPUT, I-O, EXTEND**.

START permet de se positionner sur un enregistrement. L'instruction **DELETE** n'est **pas autorisée**.

READ

```
READ F-VOITURES NEXT
AT END
....
NOT AT END
....
END-READ
```

Les enregistrements sont lus dans l'ordre dans lequel ils sont stockés dans le fichier. Cet ordre dépend de l'implémentation, ce n'est pas forcément l'ordre des clés.

WRITE, REWRITE

```
WRITE ENR-VOITURE
```

Le rewrite ne peut se faire que si on a lu préalablement un enregistrement.

START

```
MOVE "123 XY 89" TO PLAQUE.
START F-VOITURES, KEY = PLAQUE
INVALID KEY
    MOVE FAUX TO TROUVE
NOT INVALID KEY
    READ F-VOITURES NEXT
    AT END
        MOVE FAUX TO TROUVE
    NOT AT END
        MOVE VRAI TO TROUVE
    END-READ
END-START
```

Le critère de comparaison peut être =, > ou >=.

8.3.3. Accès dynamique (dynamic)

Un fichier **DYNAMIC** ne peut pas être ouvert en mode **EXTEND**.

Instructions disponibles

- **OPEN** fichier mode
- **START** fichier **KEY** **INVALID KEY** ...
- **READ** fichier **NEXT AT END** ...
- **READ** fichier **INVALID KEY** ...
- **WRITE** enregistrement **INVALID KEY** ...
- **REWRITE** enregistrement **INVALID KEY** ...
- **DELETE** fichier **INVALID KEY** ...
- **CLOSE** fichier

8.3.4. Clés multiples

Exemple:

```
ENVIRONMENT DIVISION•  
INPUT-OUTPUT-SECTION•  
FILE-CONTROL•  
    SELECT F-VOITURES  
    ASSIGN TO FILE-NAME  
    ORGANIZATION INDEXED  
    ACCESS RANDOM  
    RECORD KEY PLAQUE  
    ALTERNATE RECORD KEY MARQUE WITH DUPLICATES•  
...  
PROCEDURE DIVISION•  
...  
    MOVE "PEUGEOT" TO MARQUE•  
    START F-VOITURES KEY = MARQUE  
    INVALID KEY  
        DISPLAY "PAS DE MARQUE TOTO"  
    NOT INVALID KEY  
        MOVE FAUX TO FIN-FICHER  
        PERFORM UNTIL FIN-FICHER = VRAI  
            READ F-VOITURES NEXT  
            AT END  
                MOVE VRAI TO FIN-FICHER  
            NOT AT END  
                IF MODELE = "Z53"  
                    DISPLAY PLAQUE MODELE  
                END-IF  
            END-READ  
        END-PERFORM  
    END-START•
```

4. Fichiers relatifs

Un fichier d'organisation **relative** est une sorte de tableau dont les cases, indicées de 1 à n , contiennent un enregistrement ou rien.

Les fichiers relatifs ont beaucoup de points communs avec les fichiers indexés : modes d'accès (aléatoire, séquentiel, dynamique), modes d'ouverture (lecture, écriture, lecture-écriture, extension).

Les différences principales sont que :

- la clé est nécessairement numérique (c'est le numéro de l'enregistrement),
- la clé n'est pas un champ de l'enregistrement,
- il n'y a pas de clé secondaire.

Déclaration d'un fichier

```
ENVIRONMENT DIVISION•
INPUT-OUTPUT-SECTION•
FILE-CONTROL•
    SELECT F-ARTICLES
    ASSIGN TO FILE-NAME
    ORGANIZATION RELATIVE
    ACCESS RANDOM
    RELATIVE KEY NUMERO-ARTICLE•
...
DATA DIVISION•
FILE SECTION•
FD F-ARTICLES•
01 ENR-ARTICLE•
    02 DESIGNATION    PIC X(10)•
    02 PRIX-UNITAIRE  PIC 9(6)V99•
...
WORKING-STORAGE SECTION.
77 NUMERO-ARTICLE    PIC 9(3)•
```

4.2. Accès aléatoire (random)

Se fait en renseignant la clé, puis en lançant l'opération (READ, WRITE, REWRITE, DELETE) avec (obligatoirement) la clause INVALID KEY.

4.3. Accès séquentiel (sequential)

Les opérations autorisées sont

- **READ** fichier **NEXT, AT END ...**
- **WRITE** enregistrement
- **REWRITE** enregistrement
- **START KEY** condition clé, **INVALID KEY ...**

4.4. Accès dynamique (dynamic)

Combinaison des modes séquentiel et aléatoire:

- **READ** fichier, **INVALID KEY ...**
- **READ** fichier **NEXT, AT END ...**
- **WRITE** enregistrement **INVALID KEY ...**
- **REWRITE** enregistrement **INVALID KEY**
- **START KEY** condition clé, **INVALID KEY ...**

9. Compléments

8.1. Screen Section

Exemple:

WORKING-STORAGE SECTION●

01 LIGNE-TIRETS●

02 FILLER PIC X(80) VALUE ALL "-"●

01 LIGNE-TITRE●

02 FILLER PIC X VALUE "|"●

02 FILLER PIC X(22) VALUE ALL SPACES●

02 FILLER PIC X(34) VALUE "IUT Dept Info.- Gestion de fichiers"●

02 FILLER PIC X(22) VALUE ALL SPACES●

02 FILLER PIC X VALUE "|"●

77 WCHOIX PIC X●

...

SCREEN SECTION

01 ECRAN-MENU●

02 LINE 1 COL 1 PIC X(80) FROM LIGNE-TIRETS●

02 LINE 2 COL 1 PIC X(80) FROM LIGNE-TITRE●

02 LINE 3 COL 1 PIC X(80) FROM LIGNE-TIRETS●

02 LINE 5 COL 39 VALUE "MENU"●

02 LINE 7 COL 22 VALUE "Saisie d'etudiant(s) 1"●

02 LINE 9 COL 22 VALUE "Liste des etudiants (un par un) 2"●

02 LINE 11 COL 22 VALUE "Liste des étudiants (par paquets) ... 3"●

02 LINE 13 COL 22 VALUE "Recherche par nom 4"●

02 LINE 15 COL 22 VALUE "Recherche par numéro 5"●

02 LINE 17 COL 22 VALUE "Quitter Q"●

02 LINE 20 COL 22 VALUE "==> Votre choix : "●

02 HIGHLIGHT UPPER PIC X USING WCHOIX●

...

PROCEDURE DIVISION●

* *affichage du menu*

DISPLAY ECRAN-MENU●

* *saisie du choix*

ACCEPT ECRAN-MENU●