LANGAGE COBOL - 1ère Partie

Avertissement

Fin des années 1990, pour diverses raisons, mais principalement en raison du passage à l'an 2000, la demande en programmeurs Cobol est redevenue non nulle. Si vous souhaitez avoir un aperçu de ce langage de gestion très présent jusque dans les années 80, je vous propose quelques éléments de syntaxe qui constituent une initiation au langage COBOL

J'ai volontairement omis certaines variantes dans le codage des instructions :

- par souci de clarté de l'exposé,
- pour éviter les instructions incitant à la rédaction de programmes mal structurés,
- parce qu'elles ne sont disponibles que sur un nombre restreint de compilateurs.

Langage de référence : COBOL Microsoft 4 / Micro Focus ®
Norme ANSI X3.23 - 85 (COBOL ANS 85)

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 (COnference on DAta SYstems Languages) entre 1958-60 à la demande du gouvernt américain.

Evolutions 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.
???? : on parle d'un (futur ?) Cobol Orienté Objets

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 ; une étude estimait qu'en 1995, 30% des lignes de code existantes étaient encore COBOL.
- COBOL est adapté au 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 PRELIMINAIRES

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.
  AUTHOR. Erny.
  DATE-WRITTEN. 5/8/92.

  ENVIRONMENT  DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. dpx2000.
  OBJECT-COMPUTER. dpx2000.

  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 exemple2.
```

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
    - un tiret => coupure d'un mot réservé (instruction, etc.), d'une variable, d'un littéral
    - un D => ligne de débogage

- seules les col. 8 à 72 sont utilisées pour la partie utile du programme.

- on distingue la marge A (col. 8) et la marge B (col. 12)
    Doivent débuter en col. 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 col. 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 nom du programmeur, date de création et de compilation.

- 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 :

1°) ***les mots réservés*** ou ***mots-clés*** : ont une signification précise pour le compilateur (ex. IDENTIFICATION, STOP, DISPLAY, ...)

2°) les ***mots-utilisateur*** : crées par l'utilisateur, servent à identifier des objets du programme ; on distingue :

- les ***noms de données*** : utilisés en PROCEDURE DIVISION et déclarés en WORKING-S S, 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.

3°) ***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 occurences 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 " "

4°) **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 mots obligatoires sont soulignés.
- 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.

- Une ***instruction*** est une combinaison syntaxiquement correcte de mots et de symboles ; elle figure dans la PROCEDURE DIVISION et spécifie une action particulière à acomplir.

- 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.

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 ou 01), 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] .

```
Ex.   77   prenom PIC XXXXXXXXXX.
      77   prenom PIC X(10).
      01   age          PIC 99 .
      77   titre        PIC A(20)     VALUE "Liste des employés".
      77   qte-en-stock PIC 9(5)      VALUE ZERO   USAGE BINARY.
      77   pi           PIC 9V9(4)    VALUE 3.1416  USAGE COMP.
      77   ligne-vide   PIC X(80)     VALUE SPACES.
      77   trait        PIC X(80)     VALUE ALL "-".
```

Les données alphabétiques (PIC A) et alphanumériques (PIC X) sont normalement alignées à gauche, alors que 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.

ex. 77 marque PIC X(7) VALUE "PEUGEOT".
=> marque est représentée (en hexa) par les 7 octets suivants :

50	45	55	47	45	4F	54
P	E	U	G	E	O	T

ex. 77 qte PIC 9(3) VALUE 152.
=> qte est représenté en hexa. par les 3 octets : 31 35 32

Il peut être utile dans certains cas (gain de place, optimisation des calculs, liaison avec d'autres langages) de demander un codage différent.

- USAGE COMP ou **COMPUTATIONAL** : format interne favorisant les calculs, dépend du compilateur ; généralement la même représentation que USAGE BINARY.

- USAGE BINARY : format binaire

ex. 77 qte PIC 9(3) VALUE 152.
=> en binaire = 1001 1000, soit 8 bits => 1 octet suffit au lieu de 3 !
=> en hexa. = 98

- 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.

ex. 77 qte PIC 9(3) VALUE 152.
=> en décimal condensé = 0001 0101 0010 => 1 octet et demi suffirait
=> en hexa. = 01 52

Contrairement à d'autres langages, un type de variable spécifique existe en COBOL pour optimiser la présentation des données. Ces ***PICTURE d'édition*** spécifiques seront abordés plus loin.

2. LES INSTRUCTIONS DE BASE

2.1. DISPLAY (*Afficher*)

Cette instruction permet l'affichage de littéraux et/ou du contenu de nom-données. 2 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.

FORMAT 1

DISPLAY donnée1 donnée2 ... [WITH NO ADVANCING]

Nota : - 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 ; ce paramètre n'est pas implémenté sur tous les compilateurs.

FORMAT 2 : non implémenté sur tous les compilateurs

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.
Nota : - 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.

FORMAT 1

ACCEPT nom-donnee.

FORMAT 2

Permet le transfert des date et heure-système dans une variable utilisateur :

ACCEPT nom-donnee FROM { DATE | DAY | DAY-OF-WEEK | TIME }

- DATE : var. d'image 9(6) : date système au format AAMMJJ
- DAY : var. d'image 9(5) : date système au format AAJJJ (JJJ = n° jour dans l'année)
- DAY-OF-WEEK : var. d'image 9 : n° du jour dans la semaine (1=lundi, ..., 7=dimanche)
- TIME : var. d'image 9(8) : HHMMSSCC (Heures, minutes, secondes, centièmes)

SIGNIFICATION DES PRINCIPAUX PARAMETRES

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"
SIZE n : n = taille de la zone affichée avant saisie
ZERO-FILL : remplace les espaces par des 0
BLANK SCREEN : efface l'écran avant affichage
BLANK LINE : efface la ligne avant affichage
REVERSE-VIDEO : affiche en inversion video

EFFACEMENT DE L'ECRAN

2 instructions disponibles :
- DISPLAY SPACES UPON CRT.

ou lors du premier affichage de données :
- DISPLAY donnée AT ... WITH BLANK SCREEN.

2.3. MOVE (*instruction d'affectation*)

La clause VALUE déjà vue permet d'affecter une valeur initiale (*initialisation*) 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 | littéral } TO nom-donnee2 [nom-donnee-3 ...]

Ex. 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)	12
9999	1234	X(3)	123
X(4)	1234	99.99	34.00
X(4)	1234	9(4)	1234

Représente un espace

2.4. INSTRUCTIONS DE CALCUL

ADDITION

ADD nom-donnee1 nom-donnee2 ... TO nom-donnee-n
litteral1 litteral2

ex. ADD 12 X Y 24.67 TO Z => Z = **Z** + 12 + X +Y + 24,67

ADD nom-donnee1 nom-donnee2 ... GIVING nom-donnee-n
litteral1 litteral2

ex. ADD 12 X Y GIVING Z => Z = 12 + X +Y

SOUSTRACTION

SUBTRACT nom-donnee1 nom-donnee2 ... FROM nom-donnee-n
litteral1 litteral2

ex. SUBTRACT X Y 23 FROM Z => Z = **Z** - (X +Y +23)

en ajoutant la clause GIVING nom-donnee-n on aura :
SUBTRACT X FROM Z GIVING Y => Y = Z - X

MULTIPLICATION

MULTIPLY nom-donnee1 BY nom-donnee2
litteral1

ex. MULTIPLY 0.186 BY montant => montant = montant * 0,186

en ajoutant la clause GIVING ...
MULTIPLY 0.186 BY montant GIVING tva => tva = 0,186 * montant

DIVISION

DIVIDE nom-donnee1 BY nom-donnee2 [GIVING nom-donnee-n]
litteral1 INTO

ex. DIVIDE a BY b => b = a / b
ex. DIVIDE a BY b GIVING c => c = a / b
ex. DIVIDE a INTO b => b = b / a

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

COMPUTE

COMPUTE nom-donnee [ROUNDED] = expression mahématique

ex. COMPUTE a = ((b ** c) + (21 * 18.6)) / 2

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

MODULO

L'instruction Modulo peut être obtenue directement en COBOL grâce à l'instruction DIVIDE suivi de la clause REMAINDER (reste).

ex. a 5 Mod 2
=> DIVIDE 5 BY 2 GIVING x REMAINDER a

Remarquez que dans ce cas la clause GIVING est nécessaire (x doit être déclaré, même s'il ne sert à rien dans le programme).

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 nom-programme

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)*.

Les clauses de fin d'instruction : END-instruction[]

La plupart des instructions du COBOL 85 sont munies d'une clause facultative de fin d'instruction.

Ex. **READ** fichier AT END MOVE 1 TO fin-fich
DISPLAY "Fichier vide"
END-READ

Dans les anciennes version COBOL, les clauses de fin d'instruction n'existaient pas ; il était de règle de terminer chaque instruction par un point. Avec la version 85, il est possible d'écrire toute PROCEDURE DIVISION sans utiliser le point (sauf cas obligatoires : STOP RUN., marques de §, ...). Nous utiliserons une syntaxe intermédiaire, commune, consistant à terminer les instructions par un point, sauf si elles constituent un bloc d'instructions incluses entre un nom-instruction et END-nom-instruction.

3. LA STRUCTURE ALTERNATIVE

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

FORMAT 1

IF condition THEN instruction-1 [instruction-2 ...] [ELSE instruction-3 [instruction-4 ...]]

Le point termine l'instruction IF ; 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 !

FORMAT 2 (COBOL 85)

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

La clause **END-IF** termine l'instruction IF. **Nous considèrerons cette clause comme obligatoire** car beaucoup plus sûre que le point, ; les instructions IF peuvent être imbriquées.

```
Ex.   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

FORMAT SIMPLIFIE :

```
                identificateur-1
EVALUATE        TRUE
                FALSE

                condition-1
WHEN                  identificateur-2                identificateur-3
                [NOT]littéral-1           THRU          littéral-2
                            expr.-arithmétique-1              expr.-arithmétique-2

      instruction(s)-1
```

...

[WHEN OTHER instruction(s)-2]
[END-EVALUATE]

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

Exemples d'EVALUATE :

```
...
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 *blocs* :

```
        ...                     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].

```
PROCEDURE DIVISION.
DEBUT.                                          marque de début facultative
        <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.
FIN.                                    marque de fin facultative
```

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

FORMAT 1

PERFORM paragraphe-j [THRU paragraphe-k] nom-donnée TIMES.
littéral

Exemple :

```
DEBUT.
        <instructions>
        PERFORM creation 12 TIMES.      exécute 12 fois les instr. 3 à 5
        STOP RUN.
```

FORMAT 2 (COBOL 85)

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

Exemple :

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

4.4. Instructions Tant Que ... FTQ et Répéter ... Jusqu'à ...

FORMAT 1

```
PERFORM paragraphe(s) [WITH TEST AFTER] UNTIL condition-de-sortie.
```

FORMAT 2 (COBOL 85)

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

Par défaut (TEST BEFORE), on a l'équivalent du TANT QUE, c. à d. que la condition est testée AVANT d'exécuter le paragraphe ; si on met la clause TEST AFTER, on a l'équivalent du REPETER JUSQU'A, c. à d. que la condition n'est testée qu'après avoir exécuté une 1ère fois le paragraphe.

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

Exemples :
(on suppose que le bloc **saisie** contient instruction_1 et instruction_2 -cf exemple *blocs page 16*-) :

Algo.	Cobol
Tant que N < 10 instruction_1 instruction_2 Fin TQ	PERFORM saisie UNTIL N >= 10. ou PERFORM UNTIL N >= 10 instruction_1 instruction_2 END-PERFORM.
Répéter instruction_1 instruction_2 Jusqu'à choix = "O"	PERFORM saisie TEST AFTER UNTIL choix = "O". ou PERFORM TEST AFTER UNTIL choix = "O" instruction_1 instruction_2 END-PERFORM.

4.5. Instruction Pour ... Fin Pour

FORMAT 1

PERFORM paragraphe(s) VARYING nom-donnee1 FROM nom-donnee2
littéral-1
BY nom-donnee3 UNTIL condition-de-sortie.
littéral-2

FORMAT 2 (COBOL 85)

PERFORM VARYING nom-donnee1 FROM nom-donnee2 BY nom-donnee3
littéral-1 littéral-2
UNTIL condition-de-sortie
<instructions>

END-PERFORM.

Exemples

Algo.	Cobol
Pour i de 1 à 50 instruction_1 instruction_2 FinPour	PERFORM saisie VARYING i FROM 1 BY 1 UNTIL i > 50. ou PERFORM VARYING i FROM 1 BY 1 UNTIL i > 50 instruction_1 instruction_2 END-PERFORM.
Pour i de 1 à 25 (pas 2) instruction_1 instruction_2 FinPour	PERFORM saisie VARYING i FROM 1 BY 2 UNTIL i > 25. ou 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]) : peu usité

- **PIC X**... variable de type alphanumérique (valeurs [tous caractères ASCII])

- **PIC** ...**9**... variable numérique :
  - **PIC S9**... variable numérique **signée** :ex. 77 V PIC S9(3). V [-999..+999]
  - **PIC 9..V9**.. variable **décimale** (réel) : ex. 77 V PIC 9(3)V99. V [0..999,99]
    ex. 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**.
par ex. si on a : 77 N PIC ZZ9. , l'instruction ADD 1 TO N est erronée !
par contre 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).
Ex. une variable de PICTURE Z(4) contenant la valeur 0015 donne 15 (=espace)
si la variable contient la valeur 0000 donne

Ex. une variable de PICTURE 9 contenant la valeur 00006 donne 6
si la variable contient la valeur 00000 donne 0

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

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

Ex. une variable PIC 9(5)V99 contenant 12345^{V}67 (^{V}=position virtuelle de la virgule)
que l'on "MOVE" dans une variable PIC 99B999V,99 donne 12345,67
(le V avant la virgule signifie que la virgule doit être insérée à la position de la vigule virtuelle)

la même variable contenant 00045^{V}67
que l'on "MOVE" dans une variable PIC ZZBZZ9^{V},99 donne 45,67

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'emplact de la virgule virtuelle.

Ex. une variable PIC 9(4)V99 contenant 1234^{V}56
que l'on "MOVE" dans une variable PIC 9(4).99 donne 1234.56

si on "MOVE" cette variable dans une var. 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.
SPECIAL-NAMES.
DECIMAL-POINT IS COMMA.
```

Si cette clause est présente, la variable précédente,
que l'on "MOVE" dans une variable PIC 9(4),99 donne 1234,56
si on "MOVE" cette variable dans une var. PIC 9.999,99 donne 1.234,56

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

Le caractère $, sauf s'il est précédé de + ou -, doit être placé à l'extrême gauche de la PICTURE ; il sera simplement inséré à la place qui lui est affectée dans la PICTURE. Si le compilateur est paramétré pour la France, on obtiendra un F symbole du franc lors de l'impression.

Ex. une variable PIC 9(5)V99 contenant 1234567
que l'on "MOVE" dans une var. PIC $ZZZB999V,99 donne $12345,67

Les caractères + ou - doivent être placés à l'extrême gauche de la PICTURE :
+ 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'extrème droite de la PICTURE :

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

Edition avec insertion flottante : $ + -

Ce format utilise les caractères $ ou + ou - sous forme d'une chaîne d'au moins 2 caractères situés à gauche d'une PICTURE. Il permet d'insérer un seul caractère +, - ou $ à gauche du premier chiffre non nul de la donnée.

Ex. une variable PIC 9(5) contenant 01418
que l'on "MOVE" dans une var. PIC $$$B$$9V,99 donne $1418,00

une variable PIC 9(4) contenant 0000
que l'on "MOVE" dans une var. PIC $B$$$ donne 5 espaces

par contre, une variable PIC 99V99 contenant 00V00
que l'on "MOVE" dans une var. PIC $$V,$$ donne ,
la virgule, caractère d'insertion étant toujours éditée.

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

Algo : ex. de déclaration d'une variable structurée :

```
Var
    VEHICULE :    struct
                  NUM_IMMAT :  struct
                                        NUM1 : entier
                                        NUM2 : chaîne(3)
                                        DEPT : entier
                                        fin-struct
                  MARQUE : chaine(30)
                  MODELE : chaine(30)
                  DATE_ACHAT :          struct
                                        JOUR : chaine(2)
                                        MOIS : chaine(2)
                                        AN : chaine(2)
                                        fin-struct
                  fin-struct
```

Traduction en COBOL :

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

Les nombres-niveaux doivent être compris entre 01 et 49 ; l'incrément d'un sous-niveau peut être supérieur à 1 : ex. 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 à laquelle on n'aura pas à faire référence dans le programme (intéressant pour l'édition).

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

Exemples

Déclaration en Algo (Variables)	**Déclaration en COBOL (Working-storage section)**
TPRIX : tab[1:12] de réel	01 T-PRIX. 02 PRIX OCCURS 12 PIC 9(8)V99.
TREPRES : tab[1:30] de struct NOM : chaine[20] SAL : réel fin-struct	01 T-REPRES. 02 NOM OCCURS 30 PIC X(20). 02 SAL OCCURS 30 PIC 9(6)V99.

	ou 01 T. 02 T-REP OCCURS 30. 03 NOM PIC X(20). 03 SAL PIC 9(6)V99.
TCA : tab[1:30] [1:12] de entier tableau à 2 dim.: CA mensuel par n° représentant	01 T-CA. 02 T-REPRES 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 (contrairement au langage C)

ACCES A UN ELEMENT D'UN TABLEAU

·**Tableau à une dimension :**

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

INITIALISATION D'UN TABLEAU

L'initialisation est possible de 3 manières différentes.

a) Initialisation statique à une valeur identiquepour toutes les cases

Exemple

```
01  T.
    02      T-REPRES OCCURS 30.
            03      NOM     PIC X(20)       VALUE SPACES.
            03      SAL     PIC 9(6)V99     VALUE0.
```

b) Initialisation statique avec des valeurs différentes : clause REDEFINES

Exemple

```
01  JOURS-MOIS VALUE "312831303130313130313031"
01  REDEFINESJOURS-MOIS.
            02      NB-JOURS        OCCURS12 PIC 99.
```

Résultat : NB-JOURS (I) vaudra 31, NB-JOURS (2) vaudra28, ... NB-JOURS (12) vaudra 31.

c) Initialisation dynamique

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

Exemple

```
PERFORM VARYING I FROM 1 BY 1 UNTIL I>30
            DISPLAY"Entrez le nom du représentantn°"I":"NO ADVANCING
```

```
            ACCEPT NOM (I)
            DISPLAY"Entrez son salaire de base :"NO ADVANCING
            ACCEPT SAL (I)
    END-PERFORM.
```

Exemple avectableau à 2 dimensions (Tableau des CA par représ. et par mois)

```
    PERFORM VARYING I FROM 1 BY 1 UNTIL I>30
            DISPLAY"Représentant n°"I
            PERFORMVARYING J FROM 1 BY 1 UNTIL J>12
                    DISPLAY"CA du mois n°"J":"NO ADVANCING
                    ACCEPTCA (I, J)
            END-PERFORM
    END-PERFORM.
```

6. LES SOUS-PROGRAMMES (Procédures sans paramètres)

6.1. Instructions

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

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

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

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

```
EXITPROGRAM.
```

Une entête facultativepermet de délimiter la fin d'un programme ou sous-programme :

```
END PROGRAM nom-prog.
```

6.2. Sous-programme interne/ sous-programme externe

Le source d'unsous-programme peut se trouver au choix :

- dans le mêmefichier que le programme appelant
- dans un autre fichier quisera compilé séparément. Dans ce cas on obtient un programme-objet pour le pgappelant et un programme-objet pour le sous-programme ; le sous-programme devraalors être lié au pg appelant lors de l'édition de liens (linkage).

Le choix de mettre le sous-pg dans le même fichierou dans un fichier séparé dépend de l'utilisation ultérieure du sous-programme.Si ce dernier doit pouvoir être utilisé par d'autres programmes, on choisira defaire un sous-pg externe, sinon on choisira de faire un sous-pg interne.

6.3. Exemples de sous-programme

·Le sous-programme TIRETS est**interne**:

Fichier PRINCIPAL.CBL

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PRINCIPAL.
 PROCEDURE DIVISION.
 DEBUT.
        DISPLAY"Je vais appeler le sous-pg".
        CALL"TIRETS".
        DISPLAY"Je suis revenudans le pg appelant".
        STOP RUN.
**************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. TIRETS.
 PROCEDURE DIVISION.
```

```
DEBUT.
        DISPLAY"----------".
        EXITPROGRAM.
ENDPROGRAMTIRETS.
ENDPROGRAMPRINCIPAL.
```

·Le sous-programme TIRETS est**externe**:

Fichier PRINCIPAL.CBL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRINCIPAL.
PROCEDURE DIVISION.
DEBUT.
        DISPLAY"Je vais appeler le sous-pg".
        CALL"TIRETS".
        DISPLAY"Je suis revenudans le pg appelant".
        STOP RUN.
```

Fichier TIRETS.CBL

```
IDENTIFICATION DIVISION.
PROGRAM-ID.TIRETS.
PROCEDUREDIVISION.
DEBUT.
        DISPLAY"----------".
        EXIT PROGRAM.
```

Marche à suivre :

-Compilation séparée avec COBOLPRINCIPAL.CBL àPRINCIPAL.OBJ
COBOL TIRETS.CBL àTIRETS.OBJ
- Edition de liens avec LINK PRINCIPAL.OBJ+TIRETS.OBJàPRINCIPAL.EXE

7. LES SOUS-PROGRAMMES AVEC PARAMETRES (Procédures)

7.1. Instructions

Instruction d'appel deprocédure :

CALL"sous-prog"USING [BYREFERENCE] p1 p2 ... ...
[BY CONTENT]

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

Les paramètres formels seront déclarés dans unesection spéciale de laDATA DIVISION: la **LINKAGESECTION**
Celle-ci devra suivre laWORKING-STORAGE SECTIONsi elle existe.

La liste des paramètres formels doit figurer dansl'en-tête de laPROCEDURE DIVISIONde la procédure:
PROCEDURE DIVISION**USING**pf1 pf2 ...

7.2. Procédures internes / externes

Là aussi il appartient au programmeur de choisirentre une procédure interne ou externe. Les règles de construction sont lesmêmes que pour les sous-programmes sans paramètres.

7.3. Exemple

Fichier PPAL.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"USINGBY CONTENT A B BY REFERENCE MAX.
        DISPLAY"Le maximum est"MAX.
        STOP RUN.
```

Fichier MAXIMUM.CBL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAXIMUM.
DATA DIVISION.
LINKAGESECTION.
77 NB1  PIC S99.
77 NB2  PIC S99.
77 NBMAX PIC S99.
PROCEDURE DIVISIONUSINGNB1 NB2 NBMAX.
DEBUT.
        IFNB1>NB2
                MOVE NB1 TO NBMAX
        ELSE
                MOVE NB2 TO NBMAX
        END-IF.
        EXITPROGRAM.
```

7.4. Les Fonctions en COBOL

Il n'y a pas en COBOL la possibilité pour leprogrammeur de créer des**fonctions**.Néanmoins, le COBOL 85 inclut 42 fonctions dites"intrinsèques"pouvant être utilisées dans un programme.

ex. UPPER-CASE (CH) renvoie la chaine CHconvertie en majuscules,
INTEGER-OF-DAY(date) renvoie le n° absolu du jour correspondant à date,
SQRT(nombre) renvoie la racine carrée de nombre.

Une fonction intrinsèque pourra être appelée dans unprogramme avec la clause FUNCTION, par ex.

```
        DISPLAY"Entrez la date du jour (auformat AAAAMMJJ) :".
        ACCEPT DATE-JOUR.
        DISPLAY"Entrez votre date denaissance (AAAAMMJJ) :".
        ACCEPT DATE-NAIS.
        COMPUTE NB-JOURS = FUNCTIONINTEGER-OF-DAY (DATE-JOUR) -
                FUNCTIONINTEGER-OF-DAY (DATE-NAIS).
        DISPLAY"Vous avez vécu"NB-JOURS.
```

8. LES INSTRUCTIONS DE MANIPULATION DE CHAINES

La puissance de définition des données en Cobol(tailles prédéfinies des PIC combinées aux données structurées) fait que nousaurons rarement à utiliser ces instructions. Parmi les plus utiles :

INSPECTch**TALLYING**nb**FORALL**car.

Ex. : si ch contient"ABRACADABRA"et carcontient"A", alors nb recevra 5

INSPECTch**REPLACINGALL**car1**BY**car2.

Ex. : si ch contient"ELECTRICITE", car1contient"E"et car2 contient"I"
alorsvar sera remplacé par "ILICTRICITI".

STRINGch1 ch2 ...**INTO**ch3. concaténationde chaînes

Ex. :STRING ch1"-"ch2 INTO ch3.
si ch1 contient"Azerty"et ch2 contient"Uiop", alors ch3 recevra"Azerty-Uiop".

UNSTRINGch1**DELIMITEDBY**[**ALL]**ch2**INTO**ch3ch4 ...

Ex. :UNSTRING"Ces 3 mots"DELIMITED BY ALL SPACE INTO ch1 ch2 ch3.
ch1 contiendra"Ces", ch2 contiendra"3", ch3 contiendra"mots".

Table des matières