COBOL Bonnes pratiques

2. UTILISATION DU DEBUGGING MODE

2.1 Mode « debug »

Une ligne de source COBOL est dite en mode « debug » lorsqu'elle a un D en colonne 7.
Attention, pour qu'une instruction soit en mode « debug », si cette instruction est codée sur plusieurs lignes, il faut que toutes les lignes aient un D en colonne 7.
Exemple :

```
----+----1--…
      D    DISPLAY 'Maaaxxx, fonction ' W-FONCTION
      D            ', paramètre1=' W-PARM1
      D            ', paramètre2=' W-PARM2
```

Les instructions avec un D en colonne 7 ne sont exécutées que si le programme est compilé avec l'option « WITH DEBUGGING MODE » dans la CONFIGURATION SECTION :

```
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-370 WITH DEBUGGING MODE.
```

☞	Avant la livraison d'un composant, il faut enlever l'option « with debugging mode», puis recompiler le composant : il ne doit pas y avoir d'erreur de compilation.

☺	Evitez de mettre des points « period » à la fin des instructions en mode Debug, cela évitera d'avoir des erreurs de compilation au moment où on enlève l'option debugging mode.

On peut mettre en mode Debug (avec un D en colonne 7) :
- des déclarations de variables
- des instructions de mise en forme des variables ou du texte à afficher (par exemple move d'un code retour déclaré en binaire signé, vers un picture d'édition, String, …)
- des instructions conditionnelles utiles pour mieux gérer la mise en page du texte édité en sysout
- des instructions comme ACCEPT .. . FROM DATE ou ACCEPT … FROM TIME destinées à récupérer date et heure système afin de les afficher dans le texte des messages édités en SYSOUT

5.4.1 Booléens (noms de conditions)
Les noms de condition sont définis par le niveau 88, avec une clause Value.
Un niveau 88 est toujours associé à une donnée élémentaire (définie juste avant le – ou les – niveau(x) 88).
Exemple :

```
77  W-INDIC-FIN PIC X.

        88  PAS-FIN-FICHIER      VALUE '0'.

        88  FIN-FICHIER          VALUE '1'.
```

Lorsqu'une variable est associée à un ou plusieurs noms de condition comportant exclusivement des value numériques comme 0 et 1, il est préférable du point de vue des performances de définir la variable en PIC X.

5.4.2 Indices

pour gérer les postes d'un tableau : **binaire**
=> PIC S9(4) BINARY ou PIC S9(9) BINARY

Il faut éviter d'utiliser un type chaîne de caractères **PIC 9(n)** comme indice de tableau.

Extrait de "IBM Enterprise COBOL for z/OS Programming Guide" :
To produce the most efficient code for a BINARY data item, ensure that it has:
- A sign (an S in its PICTURE clause)
- Eight or fewer digits

5.4.3 Compteurs

binaire => PIC S9(4) BINARY ou PIC S9(18) BINARY suivant la valeur maximum attendue dans le compteur

Extrait de "IBM Enterprise COBOL for z/OS Programming Guide" :
To produce the most efficient code for a BINARY data item, ensure that it has:
- A sign (an S in its PICTURE clause)
- Eight or fewer digits

Il faut éviter d'utiliser un type chaîne de caractères **PIC 9(n)** comme compteur, surtout pour les compteurs définis uniquement en working.

Si besoin est, on fera une conversion de type pour une belle mise en forme du compteur (via une chaîne de caractères numériques éditée) seulement une seule fois à la fin du programme au moment du DISPLAY des compteurs.

7.3 MOVE

Un MOVE avec comme zone réceptrice un nom de groupe permet d'initialiser toutes les données du groupe, y compris les données de nom FILLER, mais sans tenir compte du type des données élémentaires.

MOVE effectue du padding pour remplir toute la zone réceptrice si la zone émettrice est plus courte que le zone réceptrice, et de la troncature si la zone émettrice est plus longue que le zone réceptrice

Utiliser les mots réservés COBOL : SPACE ou SPACES, ZERO ou ZEROES, LOW-VALUE ou LOW-VALUES, HIGH-VALUE ou HIGH-VALUES.

7.4 Initialisation de la zone réceptrice d'une instruction STRING

L'instruction STRING n'effectue pas d'initialisation de la zone réceptrice (zone désignée par la clause INTO).

Si le total des longueurs des zones et littéraux à concaténer est inférieur à la longueur de la zone réceptrice, la fin de celle-ci restera en l'état (pas de padding) et pourra donc comporter des caractères résiduels parasites.

2 méthodes classiques :

- faire systématiquement un MOVE SPACES TO zone-réceptrice avant chaque STRING (efficace mais peu performant si le string est effectué un grand nombre de fois et si la zone réceptrice est longue)
- ajouter à la fin de la liste des variables et littéraux à concaténer un littéral ou une constante ayant même longueur que la zone réceptrice et initialisée à SPACES,
  exemple :
  STRING 'blabla' A ' blabla ' B ' ' DELIMITED BY SIZE INTO zone-réceptrice.

7.5 Initialisation des zones de lecture d'un fichier

Un READ INTO équivaut à READ suivi de MOVE : donc il ne sert à rien d'initialiser la zone de travail avant la lecture.

10. GESTION DES TABLEAUX

Un indice est une constante ou une variable de type nombre entier qui contient le numéro du poste du tableau (en commençant à 1 pour le 1er poste).

Un index est un variable de type spécifique (usage is INDEX) qui contient le déplacement relatif d'un poste du tableau par rapport à l'adresse de début du tableau.

Un index est défini par la clause INDEXED BY directement dans la déclaration du tableau

Dans tous les cas, il faut **privilégier l'utilisation des INDEX** plutôt que celle des indices.

Si on a impérativement besoin d'indices, les déclarer de préférence en binaire : PIC S9(4) BINARY.

On peut :

convertir une valeur d'index en valeur d'indice et réciproquement par l'instruction SET : SET I TO IX,

incrémenter un INDEX pour pointer sur le poste suivant : SET IX UP BY 1,

décrémenter un index pour pointer sur le poste précédent : SET IX DOWN BY 1,

etc..

Exemple de programmation avec indice .

```
01 I PIC S9(4) BINARY.
01 I-MAX PIC S9(4) BINARY VALUE +50.
01 TABLEAU.
        02  ELEM PIC X(5) OCCURS 50.

        PERFORM VARYING I FROM 1 BY 1
           UNTIL TROUVE OR  I > I-MAX
                IF ELEM(I) = 'AAAAA'
                        SET TROUVE TO TRUE
                END-IF
        END-PERFORM.
        IF TROUVE
                MOVE I TO  …
```

Le même traitement avec un index :

```
01 I PIC S9(4) BINARY.
01 TABLEAU.
        02  ELEM PIC X(5) OCCURS 50
                        INDEXED BY IX.

        PERFORM VARYING IX FROM 1 BY 1
           UNTIL TROUVE OR  IX > 50
                IF ELEM(IX) = 'AAAAA'
                        SET TROUVE TO TRUE
                END-IF
        END-PERFORM.


        IF TROUVE
                SET I TO IX
                MOVE I TO  …
```

pour un cas testé : 2 fois moins de consommation CPU qu'un perform avec indices.

On peut utiliser SEARCH pour une recherche séquentielle « serial search » dans un tableau

L'exemple précédent devient :

```
01 I PIC S9(4) BINARY.

01 TABLEAU.

        02  ELEM PIC X(5) OCCURS 50

                        INDEXED BY IX.


        SET IX TO 1.

        SEARCH ELEM VARYING IX

           WHEN ELEM(IX) = 'AAAAA'

                SET TROUVE TO TRUE
        END-SEARCH.
        IF TROUVE
                SET I TO IX
        MOVE I TO ...
```