COBOL

partie 1

May 31, 2015

Alternatives - suite

Modification de référence

Référence une zone ou une partie de zone qui doit être déclarée en **USAGE DISPLAY**

Syntaxe :

```
ID1  [{ IN/OF } ID2 ]  (expA1 : [expA2])
```

- expA1 = Position du premier caractère référencé.
- expA2 = Longueur de la partie référencée (jusqu'à la fin si on ne précise pas) Les : sont obligatoires

Exemple :

```
77 dateJour PIC 9(8).
77 mois PIC 99.

ACCEPT dateJour FROM date YYYYMMDD
MOVE dateJour(5 : 2) to mois
DISPLAY mois
```

Informations supplémentaires

- Il ne faut pas abuser de cette méthode, les structures sont préférables.
- Si la taille du 2ème paramètre dépasse la taille de la variable Code 8, sauf si cette taille est une autre variable.

Evaluate

Correspond au *selon que* du cours de logique

Format général :

```
EVALUATE sujet
    WHEN selection1 proc1
    [WHEN...]
    [WHEN OTHER proc999]
END-EVALUATE
```

Evaluate élément

Version la plus simple et la plus proche du `selon que` de logique.

- Sujet : Une constante / variable / Expression
- Selection1 : `NOT {id1/lit1/expA1} [THRU {id2/lit2/expA2}]`
  - Doit être de type compatible pour être comparable à " sujet "
  - Le THRU signifie une intervalle de valeur, par exemple si la sélection vaut : vaut `1 THRU 4`, alors l'EVALUATE sera vrai quand le sujet vaut de 1 à 4.

Exemple

```
77 MOTORISATION PIC 9.
    88 ESSENCE ESSENCE VALUE 1.
    88 DIESEL    DIESEL VALUE 2.
    88 LPG LPG   VALUE 3.
    88 ELECTRIQUE  VALUE 4.
    88 HYBRIDE VALUE 5.

EVALUATE MOTORISATION
    WHEN 1 DISPLAY ESSENCE
    WHEN 2 DISPLAY DIESEL
    WHEN 3 DISPLAY LPG
    WHEN 4 DISPLAY ELECTRIQUE
    WHEN 5 DISPLAY HYBRIDE
    WHEN OTHER DISPLAY AUTRE
END-EVALUATE
```

Evaluate booleen

Pratique pour pouvoir utiliser des expressions logiques ou les noms-conditions.

- Sujet : TRUE ou FALSE ou EXPRESSION LOGIQUE
  - Selection1 : Expression logique / niveau 88 / TRUE / FALSE
  - Selection1 doit être de type compatible pour être comparable à " sujet "

Exemple Déclaration : voir exemple du `Evaluate élément`

```
        EVALUATE TRUE
            WHEN MOTORISATION = 1 DISPLAY ESSENCE
            WHEN MOTORISATION = 2 DISPLAY DIESEL
            WHEN MOTORISATION = 3 DISPLAY LPG
            WHEN MOTORISATION = 4 DISPLAY ELECTRIQUE
            WHEN MOTORISATION = 5 DISPLAY HYBRIDE
            WHEN OTHER DISPLAY AUTRE
        END-EVALUATE

        EVALUATE TRUE
            WHEN ESSENCE DISPLAY ESSENCE
            WHEN DIESEL DISPLAY DIESEL
            WHEN LPG DISPLAY LPG
            WHEN ELECTRIQUE DISPLAY ELECTRIQUE
            WHEN HYBRIDE DISPLAY HYBRIDE
            WHEN OTHER DISPLAY AUTRE
        END-EVALUATE

01 lesVehicules.
        03 MOTORISATION OCCURS 50 PIC 9.
            88 ESSENCE VALUE 1.
            88 DIESEL VALUE 2.
            88 LPG VALUE 3.
            88 ELECTRIQUE VALUE 4.
            88 HYBRIDE VALUE 5.
77 i PIC 99.

**** Dans une boucle :
    EVALUATE TRUE
        WHEN ESSENCE(i) DISPLAY ESSENCE
        WHEN DIESEL(i) DISPLAY DIESEL
        WHEN LPG OF MOTORISATION(i) DISPLAY LPG
```

```
        WHEN ELECTRIQUE OF MOTORISATION(i) DISPLAY ELECTRIQUE
        WHEN HYBRIDE(i) DISPLAY HYBRIDE
        WHEN OTHER DISPLAY AUTRE
    END-EVALUATE
```

Evaluate multiple

Forme plus complexe consistant à faire plusieurs évaluations en même temps.

- Sujet : `sujet1 ALSO sujet2 [ALSO ... ]`
- Selection : `selection1 ALSO selection2 [ ALSO ... ]`
  - Le mot ANY signifie que toute valeur est bonne pour le sujet correspondant.

Exemple

```
EVALUATE N1 ALSO N2
    WHEN 0 ALSO 0   // Quand N1 vaut 0 et N2 vaut 0
    WHEN 0 ALSO 1 // Quand N1 vaut 0 et N2 vaut 1
    WHEN 1 ALSO ANY // Quand N1 vaut 1 et N2 vaut n'importe quelle valeur.
    WHEN OTHER ...
END-EVALUATE
```

Les traitements de chaînes

En cobol, on peut concaténer, éclater et analyser des chaînes grâce aux verbes STRING, UNSTRING et INSPECT.

String

Permet de concaténer des zones alphanumériques dans une variable alphanumérique.

- On peut indiquer des délimiteurs de zones.
- On peut éventuellement indiquer une position de réception dans la variable réceptrice.
- On peut intercepter un problème lié à la variable réceptrice.

Format général abrégé :

```
STRING  ceQuiEstAConcaténer
    INTO reception
    [Gestion d'overflow/erreur]
END-STRING
```

Format détaillé :

```
STRING
    {ID1 / LIT1}    [{ID2 / LIT2} ...]
    DELIMITED BY {ID3 / LIT3 / SIZE}
    [{ID4/LIT4} ... DELIMITED BY {ID3 / LIT3 / SIZE} ...]

    INTO ID5  [WITH POINTER ID6]
    [ON OVERFLOW PROC1]
    [NOT ON OVERFLOW PROC2]
[END-STIRNG]
```

- Les zones émettrices sont celles qui sont entre STRING et INTO
- Les littéraux doivent être alphanumériques (entre double quotes)
- Si le délimiteur n'est pas présent, toute la zone émettrice est transférée.
- Le délimiteur ne fait pas partie de la concaténation.

Zone émettrice :

- **DELIMITED BY** : Un seul délimiteur possible par zone ou groupe de zones à concaténer.
  - **BY SIZE** : On prend la zone émettrice dans son entièreté.
  - **Littéral ou variable** : On prend la zone émettrice limitée à la rencontre du littéral ou de la valeur de la variable. Ce n'est pas limité à un seul caractère par délimiteur

Zone de réception :

- **ID5** : La variable de réception de la chaîne obtenue par concaténation des zones émettrices. ATTENTION ! Sa taille doit être suffisante pour contenir les chaînes concaténées.
  - La concaténation se fait caractère par caractère dans la variable ID5. Les positions non concernées par la concaténation restent donc inchangées !

- **ID6** : La variable numérique qui indique à quelle position d'ID5 va être placé le premier caractère de la concaténation, les positions de 1 à (ID5-1) restent donc inchangées.
  - ID6 doit pouvoir contenir (taille de la zone réceptrice +1) afin de pouvoir gérer un overflow si la zone de réception est trop petite.
- Sans pointeur, la concaténation démarre à la position 1.

Gestion d'overflow :

- **ON OVERFLOW** : Permet de gérer un problème de pointeur qu'il soit explicite dans l'instruction ou pas.
  - La valeur du pointeur < 1
  - La valeur du pointeur $>$ (longueur zone réceptrice)

ATTENTION : Avec ou sans gestion d'overflow, le résultat est toujours le même, mais il est possible de savoir si le problème a eu lieu !

Exemple :

- Exemple 1 :

```
   77 CH1 PIC X(5) VALUE ''COBOL''.
   77 CH2 PIC X(5) VALUE ''C++''.
   77 CH3 PIC X(15) VALUE ALL ''C''.
77 CH4 PIC X(6) VALUE ''Labo C''.

    STRING CH1 CH2 DELIMITED BY SPACE
        INTO CH3
```

Résultat : `CH3 = ''COBOLC++CCCCCCC''`

Dans cet exemple, DELIMITED BY SPACE ou DELIMITED BY SIZE n'aurait absolument rien changé.

- Exemple 2 :

```
    STRING
        CH4 DELIMITED BY SPACE
        CH2 DELIMITED BY SIZE
        INTO CH3 WITH POINTER 4
```

Résultat : `CH3 = ''COBLaboC++CCCCC''`

- Exemple 3 :

```
STRING CH1 CH2 DELIMITED BY SIZE
    INTO CH4
    ON OVERFLOW DISPLAY ''Dépassement''
END-STRING
```

Résultat : Dépassement `CH4 = ''COBOLC''`

Unstring

Permet d'éclater une chaîne dans plusieurs chaînes avec possibilité :

- D'indiquer des délimiteurs d'éclatement. (DELIMITED BY)
- De connaître le délimiteur rencontré. (DELIMITER IN)
- De connaître le nombre de caractères transférés dans chaque zone réceptrice. (COUNT IN)
- De connaître le nombre de chaînes transférées. (TALLYING IN)
- De démarrer l'éclatement à une certaine position. (WITH POINTER)
- D'intercepter une position invalide pour la chaîne à éclater ou un manque de zone réceptrice. (ON OVERFLOW / NOT ON OVERFLOW)

Format détaillé

```
UNSTRING ID1
    [DELIMITED BY [ALL] {ID2/LIT2}  [OR ...]]
    INTO ID3 [DELIMITER IN ID4] [COUNT IN ID5]
              [ID6 ...]
    [WITH POINTER ID7]
    [TALLYING IN ID8]
    [ON OVERFLOW PROC1]
    [NOT ON OVERFLOW PROC2]
[END-UNSTRING]
```

La zone à éclater

- **ID1** : La chaîne à éclater.
- **DELIMITED BY** : délimite l'éclatement de la variable à éclater.

  - Sans DELIMITED BY, la zone est éclatée suivant les longueurs des zones réceptrices.

- **ALL** : Quand ALL est présent devant le délimiteur, cela veut dire que le délimiteur équivaut à toutes les occurrences de ce délimiteur. Ex : DELIMITED BY ALL * Le délimiteur peut être * ou ** ou ******.
- **OR** : Permet d'inscrire plusieurs délimiteurs possibles.
- **ID2 / LIT2** : le(s) délimiteur(s).
- **WITH POINTER** : Bien que se trouvant après la zone de réception dans l'instruction, le pointeur agit sur la zone à éclater.
- **ID7** : Variable numérique du pointeur indiquant la position de début d'éclatement dans la zone à éclater.

La zone de réception

- **ID3 / ID6** : Des variables de réceptions.
  - Doivent être déclarées en USAGE DISPLAY.
- **DELIMITER IN** : Stocke le délimiteur rencontré dans une variable.
- **ID4** : La variable qui stocke le délimiteur rencontré. Il peut y avoir plusieurs DELIMITER IN et variables stockant le délimiteur (une par zone réceptrice).
- **COUNT IN** : Compte le nombre de caractères transférés dans la zone réceptrice associée sans tenir compte d'une éventuelle troncature.
  - Si la variable de réception est trop petite pour la zone à éclater, le COUNT IN compte tout de même toute la taille de la zone à éclater.
- **ID5** : Variable numérique stockant la valeur comptée par le COUNT IN.
- **TALLYING IN** : Compte le nombre de variable réceptrice.
- **ID8** : Variable numérique stockant la valeur comptée par TALLYING IN.
  - ID8 est incrémentée de 1 à chaque nouvelle variable de réception détectée.
  - ATTENTION : Cette variable n'est pas remise à 0 automatiquement à chaque nouvel UNSTRING.

Gestion d'overflow

- Permet de gérer un problème de pointeur :
  - La valeur du pointeur < 1.
  - La valeur du pointeur $>$ (longueur zone à éclater)
- Permet de savoir si le nombre de zones réceptrices est insuffisant.
- Tout comme le STRING, avec ou sans gestion d'overflow, UNSTRING donne le même résultat.

Informations supplémentaires

- Les zones réceptrices non affectées sont inchangées.
- Si le nombre de caractère transférés est plus petit que la longueur de la zone réceptrice, cette dernière est allongée de blancs si celle-ci est une chaîne ou de 0 si celle-ci est une variable numérique.
- Si on envoie une variable trop longue dans une nouvelle variable, on perd la fin, mais le COUNT IN compte toute la taille de la zone de départ.

Exemple

```
77 CH1 PIC X(14) VALUE ''ABCdAa*bc**ABC''.
    77 CH2 PIC X (5).
    77 CH3 PIC X(5).
    77 CH4 PIC X(5).

    77 CPT PIC 9.
    77 CPT1 PIC 9.
    77 CPT2 PIC 9.
    77 CPT3 PIC 9.
    77 D1 PIC X.
    77 D2 PIC X.
    77 D3 PIC X.

    UNSTRING CH1 DELIMITED BY ''*'' OR ''B''
        INTO
CH2 DELIMITER IN D1 COUNT IN CPT1
CH3 DELIMITER IN D2 COUNT IN CPT2
CH4 DELIMITER IN D3 COUNT IN CPT3
        WITH POINTER 3
        TALLYING IN CPT
        ON OVERFLOW DISPLAY ''Pas assez de zones réceptrices''
    END-UNSTRING
```

- Pas assez de zones réceptrices :
  - `CH2 = ''CdAa '' / D1 = ''*'' / CPT1 = 4`
  - `CH3 = ''bc'' / D2 = ''*'' / CPT2 = 2`
  - `CH4 = '' '' / D3 = ''*'' / CPT3 = 0`
  - CPT : 3 (A vérifier tout de même)

Inspect

- Permet de compter le nombre d'occurrences d'un caractère ou d'une suite de caractère. (TALLYING)
- Permet de remplacer certaines occurrences par d'autres de même taille (IMPORTANT). (REPLACING)
- Permet de convertir des caractères en d'autres caractères. (CONVERTING)
- TALLYING peut être couplé avec REPLACING, dans ce cas, les comptages sont faits avant les remplacements.

Deux formats possibles

Pour ce format, une des deux options est présente.

```
INSPECT ID1
    [TALLYING...]
    [REPLACING...]

INSPECT ID1
    CONVERTING ...
```

Option TALLYING

Format :

```
TALLYING
    ID2 FOR
    Argument de recherche [limitation]
    [Argument de recherche [limitation]
                ...]

    [ID5 FOR
```

```
Argument de recherche [limitation]
[Argument de recherche [limitation]
            ...]
            ...]
```

- **ID2 et ID5** sont des variables numériques (compteurs).
- Argument de recherche :
  - {CHARACTERS/ {ALL/LEADING} {ID3 / LIT3}}
    - **ID3** : Variable en USAGE DISPLAY
    - **LIT3** : Constante non numérique, ne compte que pour un caractère.
    - **CHARACTERS** : Tous les caractères d'ID1 sont comptés, pas très intéressant seul mais pratique pour connaître la taille d'une chaîne.
    - **ALL** : Toutes les occurrences d'ID3 ou LIT3 sont comptés.
    - **LEADING** : Seul les occurrences en tête de la zone de recherche sont comptées.
- Limitation :
  - `[{BEFORE / AFTER] INITIAL {ID4 / LIT4}}`
    - **ID4** : Variable en USAGE DISPLAY
    - **LIT4** : Constante non numérique.
    - **BEFORE** : On compte toutes les occurrences d'ID3 ou LIT3 jusqu'à la rencontre du délimiteur ID4 ou LIT4. Si ID4 ou LIT4, n'est pas présent, c'est comme si on recherchait sur toute la zone.
    - **AFTER** : On compte toutes les occurrences d'ID3 ou LIT3 après la première rencontre du délimiteur ID4 ou LIT4. Si ID4 ou LIT4 n'est pas présent, le résultat donnera 0.

Exemple

- INSPECT CH1 TALLYING CPT FOR ALL ''*"
  - Compte toutes les occurrences de *
    - Ex : ''*ZL*BA**" donnera 4.
- INSPECT CH1 TALLYING CPT FOR LEADING ''*" –
  - Compte les occurrences de * en tête de zone :
    - Ex : ''**45ZZ**" donnera 2.
    - Ex : ''*458 **" donnera 0.
    - Ex : ''*1*****" donnera 1.

- INSPECT CH1 TALLYING CPT FOR ALL ‘’XYZ”
  - Compte toute les occurrences de XYZ.
    - Ex : ‘’ABCXYZXYO” donnera 1.
- INSPECT CH1 TALLYING CPT FOR CHARACTERS BEFORE INITIAL SPACE
  - Compte tous les caractères avant le premier espace.
    - Ex : ‘’ABCD ABC” donnera 4.
- INSPECT CH1 TALLYING CPT FOR ALL ‘’Z” AFTER INITIAL ‘’X”
  - Compte tous les Z après le premier X rencontré.
    - Ex : ‘’AZZXPZLZ” donnera 2.

Option REPLACING

Format général :

```
REPLACING oldString BY newString
```

2 Formats possibles :

```
REPLACING CHARACTERS BY {ID2 / LIT2}
        [{BEFORE/AFTER} INITIAL {ID3/LIT3}]

REPLACING {ALL/LEADING/FIRST} {ID4/LIT4} BY {ID5/LIT5}
        [{BEFORE/AFTER}  INITIAL {ID6/LIT6}]
              [...]
```

- ID2 / LIT2 : Un seul caractère qui va remplacer tous les caractères de la zone visée.
  - Ex : INSPECT CH1 REPLACING CHARACTERS BY SPACE AFTER INITIAL ‘’.”
    - Remplace tous les caractères par des espaces après le premier point.
- ID4 et ID5 doivent être de même longueurs.
- **FIRST** : Remplace seulement la première occurrence.
- Pour les limitations autres que le FIRST : voir option TALLYING.

Informations supplémentaires

- Si plusieurs remplacements sont demandés, la recherche de ces remplacements se fait dans l'ordre de spécification. Une position de la zone de recherche ne peut faire l'objet que d'un remplacement.
  - Ex :

```
INSPECT CH1 REPLACING ALL ‘’x’’ by ‘’-‘’
   REPLACING ALL ‘’-‘’ by ‘’y’’

Les ‘’-‘’ qui ont remplacés les ‘’x’’ ne seront pas remplacés par des ‘’y’’
```

- Le comportement est le même qu'avec l'option TALLYING, voir exemple TALLYING.

Option CONVERTING

Format :

```
INSPECT ID1
    CONVERTING {ID2/LIT2} TO {ID3/LIT3}
    [{BEFORE/AFTER} INITIAL {ID4/LIT4} [...]]
```

- Dans ID1 : On remplace chaque ième caractère d'ID2/LIT2 par le ième caractère d'ID3/LIT3.
- ATTENTION : ID2/LIT2 et ID3/LIT3 doivent être de même longueur !

Exemple

```
INSPECT CH1 CONVERTING ‘’XYZ’’ TO ‘’ABC’’
```

Si CH1 vaut `‘’XYOXPZABC’’`, on obtiendra `‘’ABOAPCABC’’` (A vérifier tout de même)

Les fonctions intrinsèques

Permettent de réaliser des traitements sans devoir écrire les programmes correspondants.

Invoquer une fonction intrinsèque

Format :

```
FUNCTION nomFonction [(liste arguments)]
```

- La liste des arguments peut être fixe, variable, ou vide.
- Les arguments peuvent être des variables, des constantes, ou des expressions voir même d'autres fonctions intrinsèques.

3 types de fonctions intrinsèques

- Les fonctions numériques
  - le résultat est un réel.
- Les fonctions entières
  - le résultat est un entier.
- Les fonctions alphanumériques
  - le résultat est une chaîne.

L'affectation du résultat d'une fonction intrinsèque se fait différemment selon le type :

- Type numérique et entier : COMPUTE
  - Ex : `COMPUTE racine = FUNCTION SQRT(n)`
- Type alphanumérique : MOVE
  - Ex : `MOVE FUNCTION REVERSE(CH1) TO CH2`

Quelques fonctions utiles

Toutes les fonctions ne sont pas présentes, seulement les plus utiles.

Catégorie mathématique

- **FUNCTION SQRT(arg)** : fonction numérique calculant la racine carrée d'un nombre supérieur ou égal à 0.
- **FUNCTION COS(arg) / SIN(arg) / TAN(arg)** : fonctions numériques calculant respectivement le cosinus, le sinus et la tangente d'un angle donné en radians.
- **FUNCTION ACOS(arg) / ASIN(arg) / ATAN(arg)** : fonctions numériques donnant en radians l'arc dont le cosinus, sinus, tangente vaut arg.
  - **ACOS** et **ASIN** : arg est compris entre -1 et 1.
  - **ACOS** : valeur de retour comprise entre 0 et PI.
  - **ASIN** et **ATAN** : valeur de retour comprises entre –PI/2 et PI/2
- **FUNCTION FACTORIAL(arg)** : fonction entière donnant la factorielle d'un nombre.
  - arg doit être compris entre 0 et 28 (compris) ou 29 selon le compilateur.
- **FUNCTION LOG(arg)** : fonction numérique donnant le logarithme népérien (en base e) d'un nombre.
  - arg > 0
- **FUNCTION LOG10(arg)** : fonction numérique donnant le logarithme en base 10 d'un nombre.
  - arg > 0
- **FUNCTION INTEGER-PART(arg)** : fonction entière donnant la partie entière d'un nombre réel.
  - Ex : -26.4 donnera -26
  - Ex : 26.4 donnera 26
- **FUNCTION INTEGER(arg)** : fonction numérique donnant le plus grand entier <= arg
  - Ex : -26.4 donnera -27
  - Ex : 26.4 donnera 26
- **FUNCTION MOD(entier1 entier2)** : fonction entière donnant le reste de la division entière de deux entiers.
  - entier2 doit être différent de 0.
  - Calculé par : entier1 – (entier2 * FUNCTION INTEGER(entier1/entier2))

- **FUNCTION REM(réel1 réel2)** : fonction numérique donnant le reste de la division entière de deux réels.
  - réel2 doit être différent de 0
  - Calculé par : réel1 – (réel2 * FUNCTION INTEGER-PART(réel1/réel2))
- **FUNCTION SUM(liste arguments)** : fonction entière ou numérique donnant la somme des arguments.
  - Pour un tableau :
    * `COMPUTE somme = FUNCTION SUM(tab1(all))`
    * `COMPUTE somme = FUNCTION SUM(tab2(all all))`

Catégorie statistique

- **FUNCTION RANGE(liste d'arguments)** : calcule l'étendue des arguments.
- **FUNCTION MEAN(liste d'arguments)** : calcule la moyenne arithmétique des arguments.
- **FUNCTION MEDIAN(liste d'arguments)** : calcule la médiane des arguments.
- **FUNCTION VARIANCE(liste d'arguments)** : calcule la variance des arguments.
- **FUNCTION STANDARD-DEVIATION(liste d'arguments)** : calcule l'écart type des arguments.
- **FUNCTION MIDRANGE(liste d'arguments)** : calcule la moyenne de la plus grande valeur avec la plus petite valeur.

Catégorie caractère

- **FUNCTION UPPER-CASE(arg)** : fonction alphanumérique retournant une chaîne de même longueur que arg en replaçant les minuscules par les majuscules.
- **FUNCTION LOWER-CASE(arg)** : Pareil mais majuscules vers minuscules.
- **FUNCTION REVERSE(arg)** : Retourne une chaîne de même longueur inversé.
- **FUNCTION CHAR(arg)** : retourne un caractère correspondant à arg qui est une valeur entre 1 et 256 d'une position dans la table EBCDIC.

- **FUNCTION ORD(arg)** : fonction numérique qui retourne la position de arg dans la table EBCDIC.
- **FUNCTION NUMVAL(arg)** : fonction numérique donnant un nombre présent dans la chaîne arg.

Catégorie date et heure

- **FUNCTION CURRENT-DATE** : retourne une chaîne de 21 caractères.
  - 8 caractères `AAAAMMJJ`
  - 8 caractères `HHMMSSCC`
  - 5 caractères représentant le retard par rapport au GMT : `SHHMM` Où S est le signe (+ ou -)
- **FUNCTION WHEN-COMPILED** : retourne une chaîne de 21 caractères identiques à celle de CURRENT-DATE donnant la date de compilation.
- **FUNCTION DATE-OF-INTEGER (arg)** : fonction numérique retournant le nombre de 8 chiffres : AAAAMMJJ. arg est un entier positif donnant le nombre de jours écoulés depuis le 31/12/1600
- **FUNCTION DAY-OF-INTEGER (arg)** : fonction numérique donnant un nombre de 7 chiffres AAAAJJJ. arg est un entier positif donnant le nombre de jours écoulés depuis le 31/12/1600
- **FUNCTION INTEGER-OF-DATE (arg)** : fonction numérique retournant un nombre de 7 chiffres représentant le nombre de jours écoulés depuis le 31/12/1600 jusqu'à la date arg.
  - arg est un nombre de 8 chiffres AAAAMMJJ
- **FUNCTION INTEGER-OF-DAY(arg)** : fonction numérique retournant un nombre de 7 chiffres représentant le nombre de jours écoulés depuis le 31/12/1600 jusqu'à la date arg.
  - arg est un nombre de 7 chiffre AAAAJJJ

Catégorie générale

- **FUNCTION MAX(liste d'arguments)** : fonction (type des arguments) donnant le maximum des arguments.
- **FUNCTION MIN(liste d'arguments)** : fonction (type des arguments) donnant le minimum des arguments.

- **FUNCTION ORD-MAX(liste d'arguments)** : fonction entière donnant la position du maximum.
- **FUNCTION ORD-MIN(liste d'arguments)** : fonction entière donnant la position du minimum.

Catégorie financière

- **FUNCTION ANNUITY(arg1 arg2)** : fonction numérique où arg1 est numérique plus grand ou égal à 0 et arg2 est entier positif.
  - Arg1 est le taux d'intérêt.
  - Arg2 est le nombre de période.
  - Si arg = 0, calculé par 1 / arg2
  - Calculé par : arg1 / (- (1 + arg1)**(-arg2))
- **FUNCTION PRESENT-VALUE(arg1, arg2, ...)** : fonction numérique dont tous les arguments sont numériques.
  - Donne la valeur actuelle au taux arg1 d'une suite de versement ar2 de fin de période.
  - Arg1 doit être plus grand que -1

Les sous-programmes

Un sous-programme est un module pouvant être appelé par un autre programme ou sous-programme.

- Des paramètres peuvent être transmis
- Plusieurs possibilités d'écrire des sous-programmes :
- Un seul membre avec des programmes imbriqués.
- Un seul membre avec des programmes consécutifs.
- Plusieurs membres, des sources séparées.
- Un mélange de possibilités précédentes.

Format d'un programme COBOL :

```
IDENTIFICATION DIVISION.
...
[ENVIRONMENT DIVISION.
...]
[DATA DIVISION.
...]
[PROCEDURE DIVISION [...].
...]
END PROGRAM programID.
```

Appel d'un sous-programme

L'instruction CALL sert à appeler les sous-programmes, il en existe plusieurs formats :

```
    CALL {ID1 / LIT1}
        [USING {[BY REFERENCE] ID2 [...]
                        [BY CONTENT {ID2/LIT2} [...]]} ]
        [ON OVERFLOW proc1]
    [END-CALL]

CALL {ID1 / LIT1}
        [USING {[BY REFERENCE] ID2 [...]
                        [BY CONTENT {ID2/LIT2} [...]]} ]
        [ON EXCEPTION proc1]
        [NOT ON EXCEPTION proc2]
    [END-CALL]
```

- **ID1/LIT1** désigne le nom du programme appelé qui est une chaîne de caractère (longueur max 8 sur mainframe et 10 sur AS400, en majuscule sur AS400 !).
  - Le nom du programme est celui donné dans le PROGRAM-ID.
- Attention : les variables locales d'un sous programmes gardent leurs valeurs d'un appel à l'autre.
  - Comment changer cela ?
    - Le programme appelé est **INITIAL** : `PROGRAM-ID. nomPGM [IS] INITIAL [PROGRAM]`
  - L'instruction CANCEL permet de réinitialiser les variables du sous-programme depuis le programme appelant.
    - `CANCEL {ID1 / LIT1}`
- Transmission de paramètre(s) via le USING dans le CALL :

  - Le passage peut se faire :
    * **BY REFERENCE** : le programme appelé reçoit les adresses des paramètres.
    * **BY CONTENT** : le programme appelé reçoit les adresses de copies des paramètres.

- Récupération des paramètres dans le programme appelé :
  - Via le USING à la suite de PROCEDURE DIVISION : `PROCEDURE DIVISION USING listeVariables`
    * Les variables ont une correspondance par positions des paramètres passés
    * Le nombre de paramètres passés et de paramètres reçus doit être le même.
    * Les variables doivent être déclarées en LINKAGE SECTION peu importe l'ordre et le nom.
    * Les déclarations s'y font en 01 ou 77 sans REDEFINES.
    * Le VALUE y est interdit sauf pour un niveau 88.

- Sortie d'un sous-programme :
  - **STOP RUN** : clôture tous les programmes : rend la main au système qui libère toutes les ressources. Cette instruction ne doit pas être dans un sous-programme.
  - **GOBACK** : provoque un retour dans le programme appelant à l'instruction qui suit l'appel. Même valable pour le programme principal.
  - **EXIT PROGRAM** : Sans effet pour le programme principal. Concevable pour un sous-programme.

Gestion d'erreur

ON OVERFLOW et **ON EXCEPTION** ont la même fonction, la procédure qui suit sera exécutée si le sous-programme n'a pas pu être appelé.

Exemples des différents types de sous-programmes

Pour plus d'exemples : voir les codes du cours de COBOL2G et les labos.

Sous-programme imbriqué

```
IDENTIFICATION DIVISION.
PROGRAM-ID. pgm1.
PROCEDURE DIVISION.
0-EXEMPLE1.
DISPLAY "Dans pgm1"
CALL "pgm2"
DISPLAY "Dans pgm1"
CALL "pgm2"
DISPLAY "Dans pgm1"
STOP RUN
.
IDENTIFICATION DIVISION.
PROGRAM-ID. pgm2.
PROCEDURE DIVISION.
0-EXEMPLE2.
DISPLAY "Dans pgm2"
GOBACK
.
END PROGRAM pgm1.
END PROGRAM pgm2.
```

Sous-programme consécutif :

```
IDENTIFICATION DIVISION.
PROGRAM-ID. pgm1.
PROCEDURE DIVISION.
0-EXEMPLE1.
DISPLAY "Dans pgm1"
CALL "pgm2"
DISPLAY "Dans pgm1"
CALL "pgm2"
DISPLAY "Dans pgm1"
STOP RUN
.
END PROGRAM pgm1.
IDENTIFICATION DIVISION.
PROGRAM-ID. pgm2.
PROCEDURE DIVISION.
0-EXEMPLE2.
DISPLAY "Dans pgm2"
GOBACK
.
END PROGRAM pgm2
```