Programmation COBOL

Sébastien Jeudy

(Février 2015)

Plan du support

1. Généralités

Introduction

- COBOL est un **langage de programmation créé en 1959** (officiellement le 18 Septembre 1959)

- Son nom est l'acronyme de **COmmon Business Oriented Language** qui révèle sa vocation originelle :

  être un **langage commun pour la programmation d'applications de gestion**

- C'est un langage de haut niveau qui doit être **converti en code machine exécutable (0 et 1) à l'aide d'un compilateur**

Historique et versions (1/2)

- **1959 :** développement initié par la CODASYL (Conference on Data Systems Language, USA)

- **COBOL 60 (1960) puis COBOL-61 (1961) :** premières versions majeures

- **COBOL-68 (1968) :** standard ANSI avec gestion des tables, accès séquentiel, accès aléatoire, random processing, tri, report writer, segmentation et bibliothèque

Historique et versions (2/2)

- **COBOL-74 (1974) puis COBOL-85 (1985) :** adoption généralisée de la programmation structurée → *versions les plus répandues*
- **1989 :** ajout de fonctions intrinsèques (opérations mathématiques plus complexes, logiques ou sur des chaînes de caractères)

- **COBOL 2002 (ISO/IEC 1989-2002) :** introduction de la programmation orientée objet et du support de l'Unicode, du XML,...
- **COBOL 2014 (ISO/IEC 1989-2014) :** version en cours de développement

Caractéristiques

- **Orienté gestion :** conçu pour des applications orientées gestion (banque, assurance, finance, industrie, défense,...) traitant d'énormes volumes de données
- **Standard :** peut être compilé et exécuté sur différentes plateformes (Gros et Mini Systèmes IBM, PC, Windows, Unix/Linux, Mac OS X,...), tout en étant compatible avec ses versions antérieures

- **Structuré :** dispose de structures de contrôle ainsi que différentes divisions qui le rendent plus facile à lire et à maintenir
- **Robuste :** intègre de nombreux outils de débogage et de tests

Installation / Compilation / Exécution (1/2)

- **Avec OpenCOBOL :** www.opencobol.org → *plusieurs plateformes*
  (aujourd'hui GnuCOBOL : sourceforge.net/projects/open-cobol/)

- **Exemple d'installation (Linux Ubuntu) :**
  sudo apt-get install libc6-dev
  sudo apt-get install open-cobol

- **Aide :** man cobc
- **Edition (exemple) :** gedit helloworld.cob

Installation / Compilation / Exécution (2/2)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLOWORLD.
PROCEDURE DIVISION.
DISPLAY 'Hello World'.
STOP RUN.
```

- **Sauvegarder / Quitter :** CTRL+S / CTRL+Q
- **Compilation :** cobc -free -x helloworld.cob
- **Exécution :** ./helloworld

Structure d'un programme (1/8)

- Un programme COBOL peut comporter jusqu'à **4 divisions** contenant des **sections** formées de **paragraphes**
- Chaque paragraphe commence par une étiquette (label) et contient des **phrases (sentences) se terminant par un point**
- Chaque phrase est **composée d'instruction(s) ou de déclaration(s) (statements)** commençant par un verbe et comportant éventuellement des clauses

- Les compilateurs modernes permettent le **format libre qui n'impose plus le colonnage** et sont **insensibles à la casse** (sauf pour les chaînes de caractères)

Structure d'un programme (2/8)

- **Schématiquement :**

PROGRAM

DIVISIONS

SECTIONS

PARAGRAPHS

SENTENCES

STATEMENTS

Structure d'un programme (3/8)

1) IDENTIFICATION DIVISION :

- **Première et seule division obligatoire** d'un programme COBOL
- Utilisée par le programmeur et le compilateur **pour identifier le programme**
- Dans cette division, **PROGRAM-ID est le seul paragraphe obligatoire :** précise le **nom du programme** sur 1 à 30 caractères (on peut aussi préciser DATE-WRITTEN et AUTHOR)

- **Voir exemple « helloworld.cob »**

Structure d'un programme (4/8)

2) ENVIRONMENT DIVISION : informations sur l'environnement dans lequel le programme s'exécute, composée de 2 sections :

- **CONFIGURATION SECTION :** informations sur le système (2 §)

  SOURCE-COMPUTER : système utilisé pour la compilation

  OBJECT-COMPUTER : système utilisé pour l'exécution

- **INPUT-OUTPUT SECTION :** informations sur les fichiers (2 §)

  FILE-CONTROL : noms et organisation des fichiers (par SELECT)

  I-O-CONTROL : optimisation de la place occupée par les fichiers

Structure d'un programme (5/8)

- **Exemple :**

  ENVIRONMENT DIVISION.

  CONFIGURATION SECTION.

  SOURCE-COMPUTER. IBM-Z-OS.

  OBJECT-COMPUTER. IBM-Z-OS.

  INPUT-OUTPUT SECTION.

  FILE-CONTROL.

  SELECT FICH1 ASSIGN TO "fichier1.txt"

  ORGANIZATION IS SEQUENTIAL.

Structure d'un programme (6/8)

3) DATA DIVISION : utilisée pour définir les variables du programme, composée de 4 sections possibles :

- **FILE SECTION :** utilisée pour définir la structure d'enregistrement des fichiers ("buffer" d'accès), exemple :

```
DATA DIVISION.
FILE SECTION.
FD FICH1.
01 ENREG PIC X(50).
```

Structure d'un programme (7/8)

- **WORKING-STORAGE SECTION :** utilisée pour déclarer des variables (et des structures de fichiers) "permanentes", exemple :

  WORKING-STORAGE SECTION.

  01 WS-NOM PIC A(10).

  01 WS-NBRE PIC 9(6).

- **LOCAL-STORAGE SECTION :** semblable à la WORKING-STORAGE SECTION mais avec des variables allouées et initialisées à chaque exécution du programme, exemple :

  LOCAL-STORAGE SECTION.

  01 LS-VAL PIC 9(5).

Structure d'un programme (8/8)

- **LINKAGE SECTION :** utilisée pour décrire les variables des données reçues d'un programme externe (appelant), exemple :

  LINKAGE SECTION.

  01 LS-QUANTITE PIC 9(3).

  4) PROCEDURE DIVISION : contient les traitements du programme composés d'instructions (au moins une), les noms des sections et des paragraphes étant libres dans cette division

- La dernière instruction exécutée doit être **STOP RUN** pour la fin d'un programme principal ou **EXIT PROGRAM** pour la fin d'un sous-programme (appelé)

2. Eléments de syntaxe

Jeu de caractères

- Pour sa syntaxe (hors chaînes de caractères), **COBOL reconnaît 78 caractères :**

  A-Z a-z 0-9

  + - * /

  , ; . : ' "

  $ ( ) = > <

- **Principaux séparateurs de "mots" :** blanc / espace, virgule, point, apostrophe, guillemets, parenthèses

Format d'un source

- Historiquement (avant COBOL 2002), le source d'un programme est formaté en **colonnes de 80 caractères par ligne :**

  1-6 : numéros de lignes (ignorés par le compilateur)

  7 : blanc ou indicateur **'*' (ligne de commentaire, '*>' en libre)**, '/' (ligne de commentaire avec saut de page), '-' (continuation d'une instruction), 'D' ou 'd' (ligne prise en compte en "debugging mode")

  8-11 : FD, niveaux de base, entêtes divisions/sections/paragraphes → *Area A*

  12–72 : suite de la ligne d'instruction → *Area B*

  73-80 : indications libres pour le programmeur

- **Rappel :** les compilateurs modernes acceptent le **format libre**

Valeurs littérales

- **Valeurs brutes (données utiles)** alphanumériques ou numériques

- **Alphanumériques valides (160 caractères maximum) :**

  "Un texte" 'Un texte'

  "C'est un texte" 'C''est un texte' 'Avec guillemets "'

- **Numériques valides (18 caractères maximum) :**

  99 +99.9 -99.9

  Invalides : 1,1 1. 1.1-

Mots définis et réservés (1/2)

- Mots de 30 caractères maximum
- **Mots définis par le programmeur :**

  Noms des fichiers, zones / variables, sections, paragraphes,...

  Composés de lettres, chiffres, tirets '-' → *mots réservés interdits*
- **Mots réservés COBOL (prédéfinis) :**

  Mots clés : PIC, DISPLAY, MOVE, ADD, COMPUTE, IF, PERFORM,...

  Caractères spéciaux : +, -, *, /, =, <, <=,...

  Constantes figuratives : valeurs spéciales ZERO, SPACES,...
- **Rappel :** les compilateurs modernes sont **insensibles à la casse**

Mots définis et réservés (2/2)

- **Constantes figuratives :** → *singulier ou pluriel au choix*

  ZERO / ZEROS / ZEROES : zéro(s)

  SPACE / SPACES : espace(s)

  HIGH-VALUE / HIGH-VALUES : 1 binaire(s) X'FF'

  LOW-VALUE / LOW-VALUES : 0 binaire(s) X'00'

  NULL / NULLS : indéfini(s)

  TRUE / FALSE : booléens vrai / faux

  QUOTE / QUOTES : " ou ' (option du compilateur)

  ALL caractère : répétition du caractère précisé (entre '')

3. Description des données

Syntaxe générale

- **Rappel :** les données sont **définies dans la DATA DIVISION** (avant de pouvoir être utilisées dans la PROCEDURE DIVISION)

- **Syntaxe générale (exemple) :**

```
   01      NOM-ZONE      PIC X(10)      VALUE 'PAR DEFAUT'.
    |          |             |                 |
    |          |             |                 |
Niveau       Nom      Clause picture      Clause valeur
```

Niveaux (1/3)

- **Nombres à 2 chiffres** commençant à 01 et permettant de **hiérarchiser les données** dans des enregistrements
- Utilisés pour différencier les zones élémentaires des zones groupes
- **Zone élémentaire :** ne peut pas être divisée et définit la donnée / variable par son niveau, nom, clause picture et éventuellement clause valeur
- **Zone groupe :** composée de zones élémentaires (ou d'autres zones groupes) et définie – en général – par un niveau et un nom
  → structures imbriquées

  Dans un même groupe, **les nombres identiques sont au même niveau hiérarchique** (de même pour les niveaux 01)

Niveaux (2/3)

- **01 :** niveaux de base classiques (les plus hauts)
- **02-49 :** sous-niveaux classiques (choix libres)
- **77 :** niveaux indivisibles (zones élémentaires, variables simples)
- **88 :** sous-niveaux de zones conditionnelles (une seule à TRUE, les autres à FALSE)
- **66 :** niveaux symboliques regroupant plusieurs zones contiguës sous un même nom (aujourd'hui déconseillés)

- Numéroter les sous-niveaux par palier → *sous-groupes futurs*
- Utiliser une indentation par sous-niveau → *meilleure lisibilité*

Niveaux (3/3)

- **Exemple :**

```
01 WS-NOM PIC X(20).
01 WS-PRENOM PIC X(20).
01 WS-AGE PIC 9(3).
01 WS-ADRESSE.
    05 WS-NUMERO PIC 9(3).
    05 WS-VOIE PIC X(20).
    05 WS-COMMUNE PIC X(20).
```

Noms

- Les variables (zones de données) sont **globales au programme**
- Leur nom est de 30 caractères maximum (lettres, chiffres, tirets)
- **Noms acceptés (au moins une lettre) :** NOM, N10, 2T10, DATE-JOUR, WS-ADRESSE → *adopter des règles de nommage*
- En principe, chaque zone possède un nom unique : les noms en double doivent appartenir à des groupes (structures) différents

  (dans le code : pour utiliser des variables ayant le même nom, on doit préciser leur groupe respectif par OF ou IN)

- **FILLER :** nom spécial pour désigner des zones de "remplissage" dans les zones groupes → *réserver des emplacements fixes*

Clauses pictures (1/2)

- **Clause PIC :** permet de **préciser le format (type et taille)** des zones / variables / données

  A : alphabétique (lettres et espace)

  X : alphanumérique (tout caractère)

  N : alphanumérique dans le jeu de caractères utilisé (national)

  9 : numérique (0 à 9)

  V : virgule décimale implicite (emplacement du point numérique → *non stocké*)

  S : signe (+ ou - numérique, avant le nombre)

  1 USAGE BIT : bit booléen (0 ou 1)

Clauses pictures (2/2)

- **Exemples avec tailles :**

  PIC X(10) équivalent à PIC XXXXXXXXXX

  PIC S9(5)V9(3) équivalent à PIC S99999V999

- **Numériques édités (caractères "en dur", non calculables) :**

  B (espace) Z (zéro non significatif affiché en blanc) 0 (zéro affiché) / + - , .

  Exemples : -ZZZ9,99 (blanc si +) 99/99/9999 (date)

- La taille d'une zone groupe est la **somme de toutes ses zones**
- Le type d'une **zone groupe est toujours alphanumérique**

Clauses valeurs

- **Clause VALUE :** option utilisée pour **initialiser les données** (zones élémentaires et zones groupes)
- Peut prendre des **valeurs littérales alphanumériques et numériques ou des constantes figuratives** (ZERO, SPACES,...)
- **Exemples :**

  01 WS-TEXTE PIC X(10) VALUE 'ABC 123'.

  01 WS-NOMBRE1 PIC 99V99 VALUE IS 1.2.

  01 WS-NOMBRE2 PIC 999 VALUE ZERO.

- **Voir TP « 01-datadivision.cob »**

Clauses redéfinitions

- **Clause REDEFINES :** permet de **redéfinir une zone élémentaire ou une zone groupe** (et ses éléments) avec un nouveau nom, type et taille → *personnalisation du besoin*
- La redéfinition et la zone redéfinie possèdent les **mêmes valeurs**
- Leurs niveaux de base doivent être les mêmes (88 et 66 interdits)
- La redéfinition doit suivre et ne peut pas avoir de clause VALUE
- **Exemple :**

  01 WS-TEXTE PIC X(20) VALUE '1234'.

  01 WS-NOMBRE REDEFINES WS-TEXTE PIC 9(6).
- **Voir TP « 02-redefinitions.cob »**

Clauses renommages

- **Clause RENAMES :** permet de définir un **nom de zone de niveau 66 regroupant des zones élémentaires contiguës** et définies au préalable (début et fin au même niveau : 01, 77, 88, 66 interdits)

```
01 WS-CONTACT.
    05 WS-PRENOM PIC X(10) VALUE 'Pierre'.
    05 WS-NOM PIC X(10) VALUE 'MARTIN'.
    05 FILLER PIC X(3).
    05 WS-AGE PIC 9(2) VALUE 45.
66 WS-SOUSINFO RENAMES WS-NOM THRU WS-AGE.
```

- **Renommer une seule zone :** 66 WS-AUTRE RENAMES WS-NOM.

Clauses usages (1/2)

- **Clause USAGE :** permet d'identifier le **type de représentation interne utilisé pour enregistrer les données** (mémoire et fichiers) → *optimisation de la place occupée par les variables (notamment numériques)*
- Par défaut, toutes les données sont stockées en ASCII sur 1 octet (USAGE IS DISPLAY)
- **Usages numériques les plus utilisés :** DISPLAY (ASCII : 1 chiffre sur 8 bits), COMP-3 (octets : taille+1 / 2 arrondi supérieur), PACKED-DECIMAL (base 10 : 1 chiffre sur 4 bits), BINARY (base 2)

- **Pour une zone groupe :** l'usage spécifié s'applique à toutes ses zones élémentaires (niveaux 88 et 66 interdits)

Clauses usages (2/2)

- **Exemples :**

  PIC S9(7) USAGE IS COMPUTATIONAL-3.

  ou

  PIC S9(7) COMP-3. → *octets = (7+1)/2 = 4*

  PIC S9(5)V99 COMP-3. → *octets = (5+2+1)/2 = 4*

  PIC S9(6) COMP-3. → *octets = (6+1)/2 = 3.5, arrondi à 4*

  PIC 9(7) COMP-3. → *octets = (7+1)/2 = 4*

  PIC 9(6) COMP-3. → *octets = (6+1)/2 = 3.5, arrondi à 4*

  → calculs indispensables pour connaître la taille réelle d'un enregistrement

Constantes

- Zones élémentaires dont **la valeur n'est pas modifiable**

- **Deux descriptions possibles (exemples) :**

  78 PI VALUE 3.14. → *niveau non standard*

  01 TVA CONSTANT 20.0.

Clauses copies

- **Clause COPY :** permet d'inclure aux endroits voulus de la DATA DIVISION des **descriptions de données enregistrées dans des fichiers externes** au programme (copies)

  → capitalisation des descriptions types pour une réutilisation dans plusieurs programmes différents

- **Syntaxe :** COPY nomfichiercopy.
- **Remarque :** les fichiers copies n'ont pas besoin d'être compilés, uniquement le(s) programme(s) concerné(s) *→ à impacter lors de modifications*

- **Voir TP « 03-copyadresse.cob » et « 03-redefinitions.cob »**

4. Instructions de base

Entrées / Sorties standards (1/2)

→ *Instructions (verbes) de traitements en PROCEDURE DIVISION*

- **Instruction ACCEPT :** permet de **récupérer des valeurs en entrée standard** (clavier / console, SYSIN / JCL IBM, données système,...)

- **Exemples de syntaxes :**

  ACCEPT WS-ZONE. → *clavier / console (ou SYSIN / JCL IBM)*

  ACCEPT WS-ZONE FROM CONSOLE. → *clavier / console*

  ACCEPT WS-ZONE FROM SYSIN. → *SYSIN / JCL IBM*

Entrées / Sorties standards (2/2)

ACCEPT WS-DATE FROM DATE. → *date système (YYMMDD)*

ACCEPT WS-JOUR FROM DAY. → *jour système (YYDDD)*

ACCEPT WS-HEURE FROM TIME. → *heure système (HHMMSSss)*

- **Instruction DISPLAY :** permet d'**afficher des valeurs en sortie standard** (écran, file de sortie,...) → *ASCII*
- **Exemples de syntaxes :**

  DISPLAY WS-ZONE. → *WITH NO ADVANCING : sans retour à la ligne*

  DISPLAY "La date est " WS-DATE " et l'heure " WS-HEURE.
- **Voir TP « 04-acceptdisplay.cob »**

Initialisations des zones

- **Instruction INITIALIZE :** permet d'**initialiser les zones élémentaires et les zones groupes** (sauf clauses RENAMES)

  Alphabétiques et alphanumériques : **par des SPACES**

  Numériques : **par des ZEROES**

- **Exemple :** INITIALIZE WS-TEXTE WS-NOMBRE WS-ADRESSE.
- **Permet également une initialisation avec valeurs :**

  INITIALIZE WS-ADRESSE REPLACING NUMERIC DATA BY 9 ALPHANUMERIC DATA BY "-".
- **Voir TP « 05-initialisations.cob »**

Affectations des zones

- **Instruction MOVE :** permet d'**affecter des valeurs, ou d'autres zones, à des zones** (élémentaires ou groupes)
- **Syntaxe :** MOVE valeur/zone TO zone(s).

  → *MOVE source TO destination.*

- **Attention aux formats** (types et tailles) entre les valeurs / zones → *relativement permissif*
- **ZONE(p:n) :** 'n' caractères depuis la position 'p'
- **OF ou IN :** référence d'une zone élémentaire dans un groupe
- **Voir TP « 06-affectations.cob »**

Opérations arithmétiques (1/5)

- **Instruction ADD :** additions

- **Exemples :**

  ADD N1 N2 TO N3 N4.

  → N3 = N1 + N2 + N3 et N4 = N1 + N2 + N4

  ADD N1 N2 TO N3 GIVING N4.

  → N4 = N1 + N2 + N3

Opérations arithmétiques (2/5)

- **Instruction SUBTRACT :** soustractions

- **Exemples :**

  SUBTRACT N1 N2 FROM N3 N4.

  → N3 = N3 - N1 - N2 et N4 = N4 - N1 - N2

  SUBTRACT N1 N2 FROM N3 GIVING N4.

  → N4 = N3 - N1 - N2

Opérations arithmétiques (3/5)

- **Instruction MULTIPLY :** multiplications

- **Exemples :**

  MULTIPLY N1 BY N2 N3.

  → *N2 = N1 x N2 et N3 = N1 x N3*

  MULTIPLY N1 BY N2 GIVING N3.

  → *N3 = N1 x N2*

Opérations arithmétiques (4/5)

- **Instruction DIVIDE :** divisions

- **Exemples :**

  DIVIDE N1 INTO N2.

  → N2 = N2 / N1

  DIVIDE N1 BY N2 GIVING N3 REMAINDER N4.

  → N3 = N1 / N2 reste N4

Opérations arithmétiques (5/5)

- **Instruction COMPUTE :** calculs directs (toutes opérations)

- **Exemple :**

  COMPUTE N4 = (123.4 * N1) - (N2 / 3) + N3.

  → N4 = (123.4 x N1) - (N2 / 3) + N3

- **Voir TP « 07-calculs.cob »**

Opérations sur les chaînes (1/5)

- **Fonction LENGTH OF :** donne la **longueur d'une zone** non numérique

- **Exemple :**

  MOVE LENGTH OF WS-TEXTE1 TO WS-NOMBRE1.

- **Instruction INSPECT :** permet de **compter ou remplacer des caractères dans une chaîne** (opérations de gauche à droite)

Opérations sur les chaînes (2/5)

- **Avec option TALLYING :** compter → *sensible à la casse*
- **Exemple :**

  INSPECT WS-TEXTE1 TALLYING WS-NOMBRE1 FOR ALL 'e'.

- **Avec option REPLACING :** remplacer → *sensible à la casse*
- **Exemple :**

  INSPECT WS-TEXTE1 REPLACING ALL 'e' BY 'E' AFTER 'de'.

- Clauses **AFTER / BEFORE** possibles

Opérations sur les chaînes (3/5)

- **Et aussi :**

```
INSPECT WS-TEXTE1
CONVERTING 'abcdefghijklmnopqrstuvwxyz'
TO 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.

MOVE FUNCTION LOWER-CASE(WS-TEXTE1) TO WS-TEXTE1.

MOVE FUNCTION UPPER-CASE(WS-TEXTE1) TO WS-TEXTE1.
```

Opérations sur les chaînes (4/5)

- **Instruction STRING :** permet de **concaténer plusieurs chaînes** en une autre chaîne
- **Exemple :**

  STRING WS-TEXTE1 DELIMITED BY SIZE → *limite = taille zone*

  'qui est ' DELIMITED BY SIZE

  WS-TEXTE2 DELIMITED BY SPACE → *limite = 1er espace*

  INTO WS-TEXTE3 → *zone résultat*

  WITH POINTER WS-NOMBRE1 → *décalage et taille résultat*

  ON OVERFLOW DISPLAY "OVERFLOW" → *message si dépassement*

  END-STRING.

Opérations sur les chaînes (5/5)

- **Instruction UNSTRING :** permet d'**éclater une chaîne** en plusieurs autres chaînes

- **Exemple :**

  UNSTRING WS-TEXTE2 DELIMITED BY SPACE → *séparateur espace*

  INTO WS-TEXTE4, WS-TEXTE5, WS-TEXTE6 → *zones résultats*

  END-UNSTRING.

- Options **WITH POINTER** / **ON OVERFLOW** possibles

- **Voir TP « 08-chaines.cob »**

5. Structures de contrôles

Structures conditionnelles (1/4)

- **Instruction IF :** si, alors, sinon, fin si

- **Syntaxe générale :**

  IF condition THEN

  instructions exécutées si **condition vraie (TRUE)**

  ELSE → *si nécessaire*

  instructions exécutées si **condition fausse (FALSE)**

  END-IF.

  → *imbrications possibles (pas de '.' aux lignes imbriquées)*

Structures conditionnelles (2/4)

- **Opérateurs de comparaisons (numériques, alphanumériques, alphabétiques) :**

  EQUAL TO (=), GREATER THAN (>), LESS THAN (<),

  GREATER THAN OR EQUAL (>=), LESS THAN OR EQUAL (<=)
- **Opérateurs logiques :** NOT, AND, OR → *différent : NOT =*
- **Opérateurs typiques :**

  ZERO, POSITIVE, NEGATIVE, NUMERIC, ALPHABETIC,

  ALPHABETIC-LOWER, ALPHABETIC-UPPER

  → *le mot IS est optionnel*

Structures conditionnelles (3/4)

- **Instruction EVALUATE :** permet d'**évaluer une série de conditions / cas** → *forme plus synthétique que l'instruction IF*

- **Syntaxe générale :**

EVALUATE zone|condition|TRUE|FALSE

    WHEN valeur|condition|TRUE|FALSE → *X THRU Y possible*

        instructions

    ...

    WHEN OTHER instructions autres cas

END-EVALUATE.

Structures conditionnelles (4/4)

- **Zones conditionnelles :** sous-niveaux 88 – rattachés à un niveau inférieur – dont **un seul est à TRUE (autres à FALSE)**
- **Description (exemple) :**

  77 WS-CHOIX PIC X.

  88 OUI VALUE 'O', 'o'. → *clause PIC interdite (THRU possible)*

  88 NON VALUE 'N', 'n'.
- **Utilisation (exemple) :**

  SET OUI TO TRUE. → *jamais TO FALSE*

  IF OUI THEN …
- **Voir TP « 09-conditions.cob »**

Appels de paragraphes (1/3)

- **Instruction PERFORM / THRU :** permet de **localiser un traitement dans le code** ou d'**exécuter les instructions entre des noms de paragraphes** (avec début et fin)

  → découpage des traitements (attention à la structuration)

- **Exemples :**

  PERFORM

    DISPLAY "Traitement localisé" *→ pas de '.' aux lignes imbriquées*

    DISPLAY "Traitement localisé (suite)"

  END-PERFORM.

Appels de paragraphes (2/3)

```
PERFORM PARA-1.
PERFORM PARA-1 THRU FIN-PARA-1.
STOP RUN.
PARA-1.
   DISPLAY "Traitement PARA-1".
   DISPLAY "Traitement PARA-1 (suite)".
FIN-PARA-1.
   EXIT. → important
```

- **Voir TP « 10-paragraphes.cob »**

Appels de paragraphes (3/3)

- **Instruction GO TO :** permet d'**aller directement à un paragraphe** (débranchement sans retour)

- **Syntaxe :**

  GO TO nomparagraphe.

  → programmation non structurée (à proscrire)

Structures itératives (1/3)

- **Instruction PERFORM / TIMES :** permet d'**exécuter un paragraphe un certain nombre de fois**

- **Exemples :**

```
PERFORM 3 TIMES
   DISPLAY "Traitement localisé"
END-PERFORM.

PERFORM PARA-1 THRU FIN-PARA-1 WS-MAX TIMES.
```

Structures itératives (2/3)

- **Instruction PERFORM / UNTIL :** permet d'**exécuter un paragraphe jusqu'à une condition vraie**
- **Exemples :**

```
PERFORM UNTIL WS-N > 2
   DISPLAY "Traitement localisé : " WS-N
   ADD 1 TO WS-N
END-PERFORM.
PERFORM PARA-2 THRU FIN-PARA-2 UNTIL WS-N > 6.
```

- Clauses **WITH TEST BEFORE** / **WITH TEST AFTER** possibles

Structures itératives (3/3)

- **Instruction PERFORM / VARYING :** permet d'**exécuter un paragraphe jusqu'à une condition vraie** avec contrôle des itérations
- **Exemples :**

```
PERFORM VARYING WS-N FROM 1 BY 1 UNTIL WS-N > 4
   DISPLAY "Traitement localisé : " WS-N
END-PERFORM.
PERFORM PARA-3 THRU FIN-PARA-3
VARYING WS-N FROM 3 BY 2 UNTIL WS-N > 7.
```

- **Voir TP « 11-iterations.cob »**

6. Appels de sous-programmes

Généralités (1/2)

- Les sous-programmes permettent de **structurer le code des applications par encapsulation des fonctionnalités récurrentes** (donc réutilisables)

  → *programmation procédurale*

- Des données peuvent être échangées entre le programme appelant et le sous-programme appelé grâce aux **passages de paramètres par référence (BY REFERENCE) ou par contenu (BY CONTENT)** → *en fonction des besoins (mix possible)*

Généralités (2/2)

- **Dans le sous-programme appelé, la LINKAGE SECTION** de la DATA DIVISION (juste après la WORKING-STORAGE SECTION) **permet de décrire les paramètres reçus** → *sans VALUE*

- Le programme appelant **appelle le sous-programme par CALL USING** qui récupère les **paramètres transmis détaillés par PROCEDURE DIVISION USING** → *nombre et ordre identiques*

- Contrairement au programme "principal" appelant qui se termine par l'instruction STOP RUN (classique), **le sous-programme appelé doit se terminer par l'instruction EXIT PROGRAM** (et ne peut donc pas s'exécuter seul) → *retour à l'appelant*

Paramètres passés par référence

- **Par référence (BY REFERENCE) :** si les valeurs des paramètres transmis sont modifiées dans le sous-programme appelé, **ces modifications sont renvoyées au programme appelant**

  → mode par défaut si pas précisé

- **Programme appelant (exemple) :**

  01 WS-NOMPROG PIC X(8) VALUE "SOUSPROG". *→ dynamique*

  01 WS-CONTACT.

  05 WS-NOM PIC X(10) VALUE "MARTIN".

  → séparateur ',' si plusieurs paramètres à passer :

  CALL WS-NOMPROG USING BY REFERENCE WS-CONTACT.

Paramètres passés par contenu

- **Par contenu (BY CONTENT) :** si les valeurs des paramètres transmis sont modifiées dans le sous-programme appelé, **ces modifications ne sont pas renvoyées au programme appelant**
- **Programme appelant (exemple) :**

  01 WS-NOMPROG PIC X(8) VALUE "SOUSPROG". → *dynamique*

  01 WS-CONTACT.

      05 WS-NOM PIC X(10) VALUE "MARTIN".

  → séparateur ',' si plusieurs paramètres à passer :

  CALL WS-NOMPROG USING BY CONTENT WS-CONTACT.

Sous-programme appelé

- **Exemple :** → *PROGRAM-ID. SOUSPROG. (attention au nom)*

  LINKAGE SECTION.

  01 LS-CONTACT.

      05 LS-NOM PIC X(10).

  PROCEDURE DIVISION USING LS-CONTACT. → *',' si plusieurs*
- **Voir TP « 12-appelssousprog.cob » et « sousprog.cob »**
- **Exemple de compilation (OpenCOBOL / GnuCOBOL) :**

  cobc -free -c sousprog.cob → *objet « .o »*

  cobc -free -x 12-appelssousprog.cob sousprog.o

Cas de la LOCAL-STORAGE SECTION

- **WORKING-STORAGE SECTION :** utilisée pour déclarer des **variables (et des structures de fichiers) "permanentes"**

- **LOCAL-STORAGE SECTION :** semblable à la WORKING-STORAGE SECTION mais avec des **variables allouées et initialisées à chaque exécution du programme**

  → utile pour les sous-programmes avec appels multiples

- **Voir TP « 13-appelssousprog.cob » et « sousprog2.cob »**

7. Utilisation de tableaux

Déclarations (1/2)

- Les tableaux (tables) **se déclarent en DATA DIVISION avec la clause OCCURS / TIMES** qui détermine la répétition de leurs éléments (taille maximum)
- OCCURS ne peut s'utiliser **qu'en niveaux 02 à 49 sans REDEFINES**

- **Déclaration d'un tableau à 1 dimension (exemple) :**

```
01 WS-CONTACTS.
    02 WS-CONTACT OCCURS 10 TIMES.
        05 WS-NOM PIC X(10).
```

Déclarations (2/2)

- **Déclaration d'un tableau à 2 dimensions (exemple) :**

```
01 WS-TABLE.
    02 WS-LIGNE OCCURS 5 TIMES.
        03 WS-COLONNE OCCURS 10 TIMES.
            05 WS-VALEUR PIC X(10) VALUE "Vide".
```

Accès (1/2)

- Avec **indice(s) positif(s) de 1 au nombre maximum d'OCCURS / TIMES** → *subscript(s)*

- **Accès à un tableau à 1 dimension (exemples) :**

```
MOVE "MARTIN" TO WS-NOM(3).

MOVE "Pierre" TO WS-PRENOM(3).

MOVE 50 TO WS-AGE(3).

DISPLAY "WS-CONTACTS   : " WS-CONTACTS.

DISPLAY "WS-CONTACT(3) : " WS-CONTACT(3).
```

Accès (2/2)

- **Accès à un tableau à 2 dimensions (exemples) :**

```
MOVE "Valeur" TO WS-VALEUR(2,7).
DISPLAY "WS-TABLE              : " WS-TABLE.
DISPLAY "WS-LIGNE(2)           : " WS-LIGNE(2).
DISPLAY "WS-COLONNE(2,7) : " WS-COLONNE(2,7).
DISPLAY "WS-VALEUR(2,7)    : " WS-VALEUR(2,7).
```

→ attention aux dépassements de tailles

Tableaux indexés (1/5)

- Avec **variable(s) d'index** utilisable(s) par les instructions SET, PERFORM / VARYING, SEARCH, SEARCH ALL

  → organisation et optimisation internes des tableaux

- **Déclaration d'un tableau indexé (exemple) :**

```
01 WS-COULEURS.
    02 WS-COULEUR PIC X(10) OCCURS 10 TIMES INDEXED BY I
                                VALUE "Blanc".
```

Tableaux indexés (2/5)

- **Instruction SET :** permet d'**affecter, incrémenter et décrémenter un index**
- **Utilisation d'un tableau indexé (exemples) :**

```
SET I TO 3.

SET I UP BY 4.

SET I DOWN BY 2.

MOVE "Rouge" TO WS-COULEUR(I).

PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
   DISPLAY "WS-COULEUR(" I ") : " WS-COULEUR(I)
END-PERFORM.
```

Tableaux indexés (3/5)

- **Instruction SEARCH :** permet de **rechercher un élément dans un tableau indexé** (trié ou non) → *recherche linéaire*
- **Exemple :**

```
01 WS-TEXTE.
    02 WS-LETTRE PIC X OCCURS 15 TIMES INDEXED BY J.
...
SET J TO 1. → initialisation obligatoire
SEARCH WS-LETTRE AT END DISPLAY "Lettre 'T' non trouvée"
WHEN WS-LETTRE(J) = 'T' DISPLAY "Lettre 'T' trouvée en " J
END-SEARCH. → s'arrête quand trouvé
```

Tableaux indexés (4/5)

- **Instruction SEARCH ALL :** permet de **rechercher un élément dans un tableau indexé avec clé triée**

  → *recherche dichotomique*

- **Exemple :**

```
01 WS-ELEMENTS.
    02 WS-ELEMENT OCCURS 9 TIMES ASCENDING KEY IS WS-CLE
       INDEXED BY K.
       05 WS-CLE PIC 9.
       05 WS-VAL PIC X.
```

Tableaux indexés (5/5)

MOVE "1G2T3E4F5Z6L7B8R9U" TO WS-ELEMENTS.

DISPLAY "WS-ELEMENTS : " WS-ELEMENTS.

SEARCH ALL WS-ELEMENT AT END DISPLAY "Elément non trouvé"

WHEN WS-CLE(K) = 7 DISPLAY "Elément trouvé " WS-CLE(K) "-"

WS-VAL(K)

END-SEARCH. → *s'arrête quand trouvé*

- **Remarque :** « SET K TO 1. » inutile car obligatoirement trié

- **Voir TP « 14-tableaux.cob »**

8. Accès aux fichiers

Notion de fichier en COBOL (1/3)

- **Différente d'autres langages comme C/C++ :**

  "pas" de simples fichiers textes, mais des **fichiers PS (Physical Sequential, traités ici) et VSAM** (Virtual Storage Access Method, proches mais spécifiques aux Mainframes IBM z/OS-MVS)

- **Terminologie COBOL :**

  → orientée description des données propre au langage (comme déjà vue)

- **Champ (field) :** correspond à une **zone de donnée élémentaire avec format** (type et taille) **et attribut suivant**

Notion de fichier en COBOL (2/3)

Clé primaire (primary key) : constituée de champ(s) permettant de trouver un enregistrement unique

Clé secondaire (secondary key) : constituée de champ(s) permettant de trouver un enregistrement unique ou non

Descripteur (descriptor) : zone de donnée classique constituant l'enregistrement (non clé)

- **Enregistrement (record) :** correspond à l'**ensemble des champs décrivant l'entité de données** ayant une taille totale d'enregistrement (= somme des champs)

  Un fichier peut avoir des **enregistrements à taille fixe ou à taille variable** (si différentes structures)

Notion de fichier en COBOL (3/3)

Enregistrement physique (physical record) : correspond à l'information réelle stockée sur disque ("block")

Enregistrement logique (logical record) : correspond à l'information traitée par le programme (avec sa propre description)

- **Fichier (file) :** correspond à l'**ensemble des enregistrements décrits** (à taille fixe ou à taille variable)

Organisations des fichiers (1/3)

- Indiquent **comment les enregistrements sont organisés** dans les fichiers → *efficacité des accès*

- **Organisation séquentielle :** accès linéaire (lecture depuis le début et écriture à la fin), mise à jour possible mais tri impossible
- **Exemple de déclaration (en ENVIRONMENT DIVISION) :**

  INPUT-OUTPUT SECTION.

  FILE-CONTROL.

  SELECT CONTACTS ASSIGN TO "contacts.txt"

  ORGANIZATION IS SEQUENTIAL.

Organisations des fichiers (2/3)

- **Organisation indexée :** accès direct possible (par clés primaire et secondaire), fichier constitué de 2 parties (enregistrements séquentiels + clés/adresses correspondantes triées)
- **Exemple de déclaration (en ENVIRONMENT DIVISION) :**

  INPUT-OUTPUT SECTION.

  FILE-CONTROL.

  SELECT CONTACTS ASSIGN TO "contacts.txt"

  ORGANIZATION IS INDEXED

  RECORD KEY IS INDICE → *clé primaire*

  ALTERNATE RECORD KEY IS NOM. → *clé secondaire*

Organisations des fichiers (3/3)

- **Organisation relative :** accès par clé relative (avec adresses calculées par rapport au début de fichier), donc insertion possible et tri implicite

- **Exemple de déclaration (en ENVIRONMENT DIVISION) :**

  INPUT-OUTPUT SECTION.

  FILE-CONTROL.

  SELECT CONTACTS ASSIGN TO "contacts.txt"

  ORGANIZATION IS RELATIVE

  RELATIVE KEY IS INDICE. → *clé relative*

Modes d'accès aux fichiers (1/3)

- Indiquent **comment les enregistrements sont accédés** dans les fichiers → *dépendent des organisations*
- **Accès séquentiel :** linéaire (pour les 3 types d'organisations)
- **Exemples de déclarations (en ENVIRONMENT DIVISION) :**

  ORGANIZATION IS SEQUENTIAL

  ACCESS MODE IS SEQUENTIAL → *par ordre des insertions (ajouts)*

  ORGANIZATION IS INDEXED

  ACCESS MODE IS SEQUENTIAL → *par valeur des clés (index)*

  ORGANIZATION IS RELATIVE

  ACCESS MODE IS SEQUENTIAL → *par valeur des clés (relatives)*

Modes d'accès aux fichiers (2/3)

- **Accès variable (random) :** par clés (organisations indexées et relatives)

- **Exemples de déclarations (en ENVIRONMENT DIVISION) :**

  ORGANIZATION IS INDEXED

  ACCESS MODE IS RANDOM → *par valeur des clés (index)*

  ORGANIZATION IS RELATIVE

  ACCESS MODE IS RANDOM → *par valeur des clés (relatives)*

Modes d'accès aux fichiers (3/3)

- **Accès dynamique :** permet à la fois l'accès séquentiel (linéaire) et l'accès variable / random (par clés) pour le même fichier

  → en fonction des instructions d'accès utilisées (partie suivante) et des clés existantes (ou pas) pour le fichier accédé

- **Déclaration (en ENVIRONMENT DIVISION) :** pour les 3 types d'organisations

  ACCESS MODE IS DYNAMIC

FILE SECTION

- Utilisée pour **définir la structure d'enregistrement des fichiers**

  → "buffer" des données lors des accès de lecture / écriture

- **Exemple de description (en DATA DIVISION) :**

  FILE SECTION.

  FD CONTACTS. *→ nom du fichier déclaré en SELECT*

  01 CONTACT. *→ niveau d'enregistrement du fichier*

      05 INDICE PIC 9(6).

      05 NOM PIC A(20).

      05 PRENOM PIC A(20).

Instructions d'accès (1/8)

- **Instruction OPEN :** ouverture d'un fichier → *accessible*

- **Syntaxe :** OPEN mode nomfichier.

- **Différents modes :**

  INPUT : en lecture seule (fichier existant)

  OUTPUT : en écriture seule (insertion)

  I-O : en lecture / (ré)écriture d'enregistrement(s)

  EXTEND : en insertion séquentielle uniquement (ajout à la fin)

Instructions d'accès (2/8)

- **Instruction READ :** lecture d'un enregistrement

- **Syntaxe en accès séquentiel (linéaire) :**

  READ nomfichier NEXT RECORD INTO wsstructure

  AT END instruction correspondante

  NOT AT END instruction correspondante

  END-READ.

Instructions d'accès (3/8)

- **Syntaxe en accès variable / random (par clé) :**

  READ nomfichier RECORD INTO wsstructure

  KEY IS cléfichier

  INVALID KEY instruction correspondante

  NOT INVALID KEY instruction correspondante

  END-READ.

Instructions d'accès (4/8)

- **Instruction WRITE :** écriture d'un enregistrement
- **Syntaxe en organisation séquentielle (linéaire) :**

  WRITE enregfichier [FROM wsstructure] → *ajout à la fin*

  END-WRITE.
- **Syntaxe en organisations indexée et relative (par clé) :**

  WRITE enregfichier [FROM wsstructure]

  INVALID KEY instruction correspondante

  NOT INVALID KEY instruction correspondante

  END-WRITE.

Instructions d'accès (5/8)

- **Instruction REWRITE :** mise à jour d'un enregistrement → *I-O*
- **Syntaxe en organisation séquentielle (linéaire) :**

  REWRITE enregfichier [FROM wsstructure]

  END-REWRITE.
- **Syntaxe en organisations indexée et relative (par clé) :**

  REWRITE enregfichier [FROM wsstructure]

  INVALID KEY instruction correspondante

  NOT INVALID KEY instruction correspondante

  END-REWRITE.

Instructions d'accès (6/8)

- **Instruction DELETE :** suppression d'un enregistrement → *I-O*

  → *impossible en organisation séquentielle*

- **Syntaxe en organisations indexée et relative (par clé) :**

  DELETE nomfichier RECORD → *avec cléfichier renseignée*

  INVALID KEY instruction correspondante

  NOT INVALID KEY instruction correspondante

  END-DELETE.

  → *dernier enregistrement lu en accès séquentiel*

Instructions d'accès (7/8)

- **Instruction START :** positionnement à un enregistrement

  → impossible en organisation séquentielle

- **Syntaxe en organisations indexée et relative (par clé) :**

  START nomfichier KEY IS [=, >, <, NOT, >=, <=] cléfichier

  INVALID KEY instruction correspondante

  NOT INVALID KEY instruction correspondante

  END-START.

Instructions d'accès (8/8)

- **Instruction CLOSE :** fermeture d'un fichier → *inaccessible*
- **Syntaxe :** CLOSE nomfichier.
- **Voir TP « 15-fichiers.cob »**

Instructions complémentaires

- **Instruction SORT :** permet de **trier un fichier** en entrée (selon une clé, avec fichier en sortie et fichier de travail en description SD), exemple :

  SORT WORK1 ON ASCENDING KEY INDICE1O

  USING INPUT1 GIVING OUTPUT1.

- **Instruction MERGE :** permet de **fusionner/trier des fichiers** en entrée (selon une clé, avec fichier en sortie et fichier de travail en description SD), exemple :

  MERGE WORK12 ON ASCENDING KEY INDICE12O

  USING INPUT1, INPUT2 GIVING OUTPUT12.

- **Voir TP « 16-sortmerge.cob »**

9. Accès aux bases de données (DB2)

Contexte (1/3)

- **Accès à DB2 UDB (Universal Data Base) :** système de gestion de base de données relationnelle (**SGBDR**) d'IBM principalement dédié aux mainframes et mini-systèmes, **avec langage de requêtes structurées SQL** → *accès similaire aux autres SGBDR*

- **Traité ici sur System i / IBM i (AS/400)** → *"Embedded SQL" (DB2 intégré)*
- **Programmes COBOL de type SQLCBLLE (ou SQLCBL) :**

  Instructions SQL imbriquées au code entre EXEC SQL / END-EXEC.

  Avec "host variables" : variables COBOL (préfixées par ':') pour échanges de données DB2

Contexte (2/3)

- **Description DDS de la table utilisée FICCO0P (exemple) :**

  → fichier physique (PF) avec enregistrements (ex : indices 1, 2,...)

```
A  R FICCOF                      TEXT('Fichier contacts')
A    COIND          6S 0         TEXT('Indice')
A    CONOM          20A          TEXT('Nom')
A    COPRE          20A          TEXT('Prénom')
A    COTEL          10A          TEXT('Téléphone')
A    COCPO           5A          TEXT('Code Postal')
A    COCOM          20A          TEXT('Commune')
```

Contexte (3/3)

- **Partie déclarative COBOL (exemple) :**

```
WORKING-STORAGE SECTION.
    EXEC SQL
    INCLUDE SQLCA → informations échangées avec DB2
    END-EXEC.          (SQLCODE, SQLSTATE,...)
01 WS-CONTACT. → "host variables" utilisées
    05 WS-IND PIC S9(6). → attention aux types / tailles
    05 WS-NOM PIC X(20).
    05 WS-PRE PIC X(20).
...
```

Requêtes SELECT (1/2)

- **Lectures de données (exemple) :**

```
MOVE 2 TO WS-IND.
EXEC SQL
   SELECT COIND, CONOM, COPRE
   INTO :WS-IND, :WS-NOM, :WS-PRE → "host variables"
   FROM FICCO0P
   WHERE COIND = :WS-IND → 1 seul enregistrement retourné
END-EXEC.
```

Requêtes SELECT (2/2)

- **Traitement du code retourné par DB2 (exemple) :**

```
IF SQLCODE = 0 THEN → pas d'erreur
    DISPLAY "Données lues : " WS-CONTACT
ELSE
    DISPLAY "Erreur de lecture : " SQLCODE "-" SQLSTATE
END-IF.
```

Requêtes INSERT

- **Insertions de données (exemple) :**

```
MOVE 3 TO WS-IND.
...
EXEC SQL
   INSERT INTO FICCO0P
   (COIND, CONOM, COPRE, COTEL, COCPO, COCOM)
   VALUES
   (:WS-IND, :WS-NOM, :WS-PRE, :WS-TEL, :WS-CPO, :WS-COM)
   WITH NONE → sans journalisation / isolation (par défaut)
END-EXEC.
```

Requêtes UPDATE

- **Mises à jour de données (exemple) :**

```
MOVE "Marie" TO WS-PRE.
EXEC SQL
   UPDATE FICCO0P
   SET COPRE = :WS-PRE
   WHERE COIND = :WS-IND
   WITH NONE
END-EXEC.
```

Requêtes DELETE

- **Suppressions de données (exemple) :**

```
EXEC SQL
   DELETE FROM FICCO0P
   WHERE COIND = :WS-IND
   WITH NONE
END-EXEC.
```

Utilisation de curseurs (1/3)

- Un curseur permet de **traiter un à un les enregistrements (lignes de table) retournés** par une requête SELECT
- **Déclaration et ouverture (exemple) :**

```
EXEC SQL
   DECLARE COCUR CURSOR FOR
   SELECT COIND, CONOM, COPRE FROM FICCO0P
END-EXEC.
EXEC SQL
   OPEN COCUR
END-EXEC.
```

Utilisation de curseurs (2/3)

- **Traitement des enregistrements un à un (exemple) :**

```
PERFORM UNTIL SQLCODE = 100 → fin des enregistrements
   EXEC SQL
      FETCH COCUR → enregistrement suivant
      INTO :WS-IND, :WS-NOM, :WS-PRE
   END-EXEC
   IF SQLCODE NOT = 100 THEN
      DISPLAY "Données lues : " WS-CONTACT
   END-IF
END-PERFORM.
```

Utilisation de curseurs (3/3)

- **Fermeture (exemple) :**

```
EXEC SQL
   CLOSE COCUR
END-EXEC.
```

- **Voir TP « 17-accesdb2.cob »**