COBOL, notes de cours

Michel Billaud <billaud@info.iuta.u-bordeaux.fr> Octobre 1998

Ce document n'est ni un manuel de référence COBOL, ni un cours de programmation. ($Revision: 1.3 $, $Date: 1998/10/20 09: 08: 34 $)

Table des matières

Chapitre 1

Structure des programmes et instructions de base

1.1 Structure d'un programme

Un programme COBOL comporte quatre *divisions* qui contiennent des *sections* formées de *paragraphes*. Chaque paragraphe commence par une *étiquette*, et contient des *phrases* terminées par des points. Chaque phrase est composée d'*instructions* (ou de *déclarations*) commençant par un *verbe* et comportant éventuellement des *clauses* optionnelles.

1.1.1 `identification division`

Cette division fournit une documentation minimale du programme. Elle contient le nom du programme, et des commentaires divers : l'auteur, notice d'installation, la date d'écriture du programme, etc.

1.1.2 `environment division`

Décrit l'environnement dans lequel ce programme est exploité (`configuration section`). Par exemple : jeu de caractères utilisé, signe monétaire, signe décimal... Décrit aussi les fichiers (`input-output section`) que le programme utilisera (chemin d'accès, type, méthode d'accès,...).

1.1.3 `data division`

Décrit les variables en mémoire. Variables associées aux fichiers (`file section`), variables de travail (`working-storage section`), paramètres formels des sous-programmes (`linkage section`), variables de communication (`communication section`) et du générateur d'états (`report section`).

1.1.4 `procedure division`

Contient les instructions du programme.

1.2 Instructions

1.2.1 Arrêt

L'instruction `stop run` met fin au déroulement d'un programme COBOL.

1.2.2 Affectation

Exemples :

```
move 0 to total.
move spaces to ligne1, ligne 2.
```

1.2.3 Calcul

On peut employer des verbes spécifiques (`add, subtract, multiply, divide`) ou bien `compute` qui enchaîne plusieurs opérations.

Exemples :

```
add 1 to compteur.
add TOTAL-MATIN, TOTAL-SOIR giving TOTAL-JOURNEE.
add TOTAL-JOURNEE to TOTAL-MOIS, TOTAL-ANNEE.
subtract REMISE from PRIX.
multiply QUANTITE, PRix-UNITAIRE giving PRIX.
divide POIDS-GATEAU by NOMBRE-PARTS giving POIDS-PART.
divide NB-FROMAGE by NB-SOURIS giving PART-SOURIS remainder RESTE.
compute PRIX = (PRIX-UNITAIRE * QUANTITE) * (1 - POURCENT-REMISE / 100)
```

Des clauses optionnelles permettent de demander un arrondi, et de préciser ce qu'il faut faire si il y a (ou pas) débordement.

```
divide POIDS-GATEAU by NOMBRE-PARTS giving POIDS-PART rounded.

compute PRIX = (PRIX-UNITAIRE * QUANTITE) * (1 - POURCENT-REMISE / 100)
        on size error display "Debordement"
                      display "Donnees probablement incorrectes"
                      move 17 to CODE-ERREUR
        not on size error
                      move 0 to CODE-ERREUR
end-compute.
```

1.2.4 Conditionnelles

`if ... then ... else ... end-if`

```
if NOM = spaces then move 17 to CODE-ERREUR.

if TOTAL-JOURNEE > TOTAL-MEILLEURE-JOURNEE
then
   move TOTAL-JOURNEE to TOTAL-MEILLEURE-JOURNEE
   move JOUR to MEILLEUR-JOUR
```

```
end-if.

if A > B then
   move A to MAX
   move B to MIN
else
   move A to MIN
   mode B to MAX
end-if.
```

Lorsque le délimiteur `end-if` est omis, le corps du `if` se prolonge jusqu'au prochain délimiteur, ou au point qui termine la phrase.

`Evaluate`

```
evaluate REPONSE
when "OUI"
when "oui"
      move 1 to CODE-REPONSE
when "NON"
when "non"
      move 2 to CODE-REPONSE
when other
      move 99 to CODE-REPONSE
end-evaluate.

evaluate MOY-GENERALE
when 10 thru 20 move "Recu"      to RESULTAT
when  8 thru 10 move "Repechage" to RESULTAT
when other      move "Colle"     to RESULTAT
end-evaluate

evaluate NOTE-EXAM  also NOTE-STAGE
when     10 thru 20 also 10 thru 20
                              move "recu" to DECISION
when     9  thru 10 also 12 thru 10
                              move "rattrape par le stage" to DECISION
when     14 thru 20 also  9 thru 10
                              move "rattrape par notes exam" to DECISION
when other
                              move "Elimine" to DECISION
end-evaluate.
```

1.2.5 Perform : appel de sous-programme et répétition

L'instruction `perform` sert à la fois à faire des appels de sous-programmes et des boucles. Elle peut prendre deux formes :

- `perform` *nom-sous-programme clauses* `[end-perform]`
- `perform` *clauses instructions* `[end-perform]`

L'usage du délimiteur de fin optionnel `end-perform` est recommandé pour la seconde forme (dite "perform en ligne").

Un appel simple de sous-programme s'écrit sans clauses:

```
perform CALCUL-REMISE.
```

La clause "`until` *condition*" permet de répéter une boucle jusqu'à ce qu'une condition devienne vraie. Si on la fait précéder par "`with test after`", la condition n'est pas testée à l'entrée dans la boucle, mais à la sortie. Exemples :

```
perform with test after until REPONSE = "OUI" or "NON"
   display QUESTION "(repondez par OUI ou NON) ? "
   accept  REPONSE
end-perform.

perform TRAITER-DONNEES
until FIN-DONNEES = VRAI
   or CODE-ERREUR not = 0.
```

La clause n `times` répète n fois (boucle sans compteur):

```
perform SAUTER-LIGNE 10 times.
perform 3 times
  display "Alerte !"
end-perform.
```

Une boucle avec compteurs s'obtient par la clause "`varying ... from ... by ... until ...`"

```
perform varying I from 1 by 1 until I > 10
        move spaces to TABLEAU(I)
end-perform.
```

1.3 Entrées-sorties

1.3.1 Entrées-sorties interactives

```
display "Donnez un nombre entre " MIN " et " MAX " : ".
accept   REPONSE.
```

1.3.2 Entrées-sorties sur fichiers séquentiels

Les autres types de fichiers seront montrés ultérieurement.

Déclaration de fichier

```
environment division.
   input-output section.
   file-control.
       select FICHIER-CLIENTS assign to "/home/compta/clients"
       organization line sequential
       access sequential.
```

Déclaration d'enregistrement

```
data division.
   file section.
   fd FICHIER-CLIENT.
   01 F1-ENR.
      02 F1-NOM            pic X(30).
      02 F1-ADRESSE        pic X(40).
      02 F1-TELEPHONE      pic 9(10).
```

Ouverture de fichier

```
open input FICHIER-CLIENT.
open output FICHIER-CLIENT.
open extend FICHIER-CLIENT.
```

Lecture, écriture

```
read FICHIER-CLIENT
  at end move VRAI to FIN-FICHIER-CLIENT
  not at end perform TRAITER-ENREGISTREMENT-CLIENT
end-read.

write F1-ENR.
```

Fermeture de fichier

```
close FICHIER-CLIENT.
```

1.4 Un exemple

1.4.1 Le programme

```
       IDENTIFICATION DIVISION.
       program-id. progtele.
       author. M. Billaud.
       date-written. 8/02/1998.

      *
      * consultation du programme tele de la semaine.
      *
      * - realise avec Personal Cobol (Microfocus)


       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       file-control.
           select F-FILMS assign to "semaine.txt"
               organization line sequential.


       DATA DIVISION.
```

```
       FILE SECTION.

       fd F-FILMS.
       01 F-ENR.
           02 JOUR       pic XXX.
           02 CHAINE     pic XXXX.
      *    heure de debut et duree en minutes
           02 HHMM       pic 9999.
           02 DUREE      pic 999.
      *    titre
           02 TITRE      pic X(60).

       WORKING-STORAGE SECTION.

      * constantes definies par commodite.
       77  VRAI          pic X   value "V".
       77  FAUX          pic X   value "F".

       77  FIN-MENU      pic X.
       77  FIN-FICHIER   pic X.

       77  CHOIX-MENU    pic X.

       01  LIGNE-AFFICHAGE.
           02            pic XX  value "* ".
           02 JOUR       pic XXX.
           02            pic X   value " ".
           02 CHAINE     pic X(4).
           02            pic XXX value " | ".
           02 HHMM       pic Z9B99.
           02            pic X   value " ".
           02 TITRE      pic X(30).
           02            pic XX  value " |".

       01  LIGNE-ETOILES pic X(50) value all "*".

       77  JOUR-CHOISI   pic X(3).
       77  NB-FILMS      pic 999.


       PROCEDURE DIVISION.
       PRINCIPALE SECTION.
       DEBUT.
           display "hello".
           move FAUX to FIN-MENU.
           perform MENU until FIN-MENU = VRAI.
           stop run.

       MENU.
             display "Films de la semaine"
             display " ".
             display "T. Tous les films"
             display "J. Films d'un jour"
             display "C. Films d'une chaine"
             display "H. Recherche par heure de debut"
```

```
           display "Q. Quitter".
           display " "
           display "Votre choix ?"
           accept CHOIX-MENU
           evaluate CHOIX-MENU
           when "T"
           when "t"
                 perform EDITER-TOUS-FILMS
           when "J"
           when "j"
                 perform EDITER-JOUR
           when "C"
           when "c"
                 perform EDITER-CHAINE
           when "H"
           when "h"
                 perform RECHERCHER-HORAIRE
           when "Q"
           when "q"
                 move VRAI to FIN-MENU
           when other
                 display "Choix invalide"
           end-evaluate.

       EDITER-TOUS-FILMS.
           move FAUX to FIN-FICHIER.
           open input F-FILMS.
           display LIGNE-ETOILES.
           move 0 to NB-FILMS.
           perform until FIN-FICHIER = VRAI
              move SPACES to F-ENR
              read F-FILMS
              at end
                 move VRAI to FIN-FICHIER
              not at end
                 add 1 to NB-FILMS
                 perform AFFICHER-ENREGISTREMENT
              end-read
           end-perform.
           close F-FILMS.
           display LIGNE-ETOILES.
           display NB-FILMS " films trouves.".
           display " ".

       AFFICHER-ENREGISTREMENT.
           move CHAINE of F-ENR to CHAINE of LIGNE-AFFICHAGE.
           move JOUR   of F-ENR to JOUR   of LIGNE-AFFICHAGE.
           move HHMM   of F-ENR to HHMM   of LIGNE-AFFICHAGE.
           move TITRE  of F-ENR to TITRE  of LIGNE-AFFICHAGE.

           display LIGNE-AFFICHAGE.

       EDITER-JOUR.
           display "Quel Jour ? (3 premieres lettres majuscules)".
           accept JOUR-CHOISI.
```

```

     move FAUX to FIN-FICHIER.
     open input F-FILMS.
     move 0 to NB-FILMS.
     display LIGNE-ETOILES.
     perform until FIN-FICHIER = VRAI
        move SPACES to F-ENR
        read F-FILMS
        at end
           move VRAI to FIN-FICHIER
        not at end
           if JOUR-CHOISI = JOUR of F-ENR
           then
             add 1 to NB-FILMS
             perform AFFICHER-ENREGISTREMENT
           end-if
        end-read
     end-perform.
     close F-FILMS.
     display LIGNE-ETOILES.
     display NB-FILMS " films trouves pour le jour "
             JOUR-CHOISI.
     display " ".

  EDITER-CHAINE.
           display "* EDITER-CHAINE : A completer".

  RECHERCHER-HORAIRE.
     display "* RECHERCHER-HORAIRE : A completer".
```

1.4.2 Les données : le fichier `semaine.txt`

```
SAMC+  0850095Touche pas a mon periscope
DIMC+  1800095Touche pas a mon periscope
JEUC+  1655095Touche pas a mon periscope
DIMC+  0225095Touche pas a mon periscope (VO)
LUNC+  1335095Touche pas a mon periscope
SAMC+  1020108Planete hurlante
LUNC+  1335108Planete hurlante
SAMC+  0335108La septieme demeure
DIMC+  0745087Dans les griffes de la momie
DIMC+  0920087Bienvenue dans l'age ingrat
DIMC+  1045097Esprits rebelles
LUNC+  2035097Esprits rebelles
DIMC+  1035124Carlas's song
MERC+  2230124Carlas's song (VO)
DIMARTE2045100The Horror Picture Show (VO)
DIMF2  2055140L. 627
DIMTF1 2250100La veuve noire
LUNF3  0020081La chanson d'une nuit
LUNF3  0140120Mais ne te promenes donc pas toute nue !
LUNC+  0354103Salut cousin
LUNC+  0525091Flipper
```

```
LUNLA5 1430100Knock
```

1.4.3 Travail à faire

- faire afficher la durée
- rectifier la présentation
- complèter le programme en ajoutant une sélection par chaîne, et une sélection par horaire (p.ex LUN 1230)

Chapitre 2

Les variables

2.1 Déclaration des variables

En COBOL les variables sont déclarées dans la `data division`. Selon leur fonction, elles sont déclarées dans la `file section` ou la `working-storage section`.

Une déclaration de variable comporte plusieurs volets, dont :

- un numéro de niveau (01 à 49 pour les groupes et leurs éléments), 77 pour les variables isolées, 88 pour les conditions,...
- un nom (facultatif) pour la variable
- une clause `picture` indiquant la représentation externe de la variable
- une clause `usage` indiquant si il s'agit d'une variable de calcul ou d'édition
- un nombre d'occurences (dimension d'un tableau)
- une valeur initiale

2.1.1 Groupes

Un groupe est une variable composée de plusieurs champs (variables subordonnées) qui sont déclarées avec des numéros de niveaux plus grands. Exemple :

```
01 PERSONNE.
  02 NOM     pic X(30).
  02 PRENOM  pic X(30).
  02 DATE-NAISSANCE.
    03 JOUR  pic 99.
    03 MOIS  pic 99.
    03 ANNEE pic 9999.
  02 ADRESSE pic X(30).
```

2.1.2 Variables isolées

Pour les variables isolées, on peut utiliser le numéro spécial 77 au lieu de 01.

```
77 TOTAL-JOUR  pic 9(6)V99.
```

2.1.3 Filler

`Filler` est un nom spécial pour désigner les variables “de remplissage”.

```
01 LIGNE-IMPRESSION.
  02 filler    pic XX     value "* ".
  02 NOM       pic X(20).
  02 filler    pic XXX    value " * ".
  02 MOYENNE   pic Z9.99.
  02 filler    pic XX     value " *".
```

Depuis Cobol 85, on peut simplement omettre le nom des champs de remplissage :

```
01 LIGNE-IMPRESSION.
  02           pic XX     value "* ".
  02 NOM       pic X(20).
  02           pic XXX    value " * ".
  02 MOYENNE   pic Z9.99.
  02           pic XX     value " *".
```

2.1.4 Tableaux

On déclare un tableau par la clause `occurs`:

```
01 MOYENNE-GENERALE  pic 99v99   occurs 25.
01 TAB-NOTES occurs 25.
   02 NOTE-MATHS     pic 99v99.
   02 NOTE-FRANCAIS  pic 99v99.
```

Les indices partent de 1. On désigne chaque élement en faisant figurer l’indice entre parenthèses : `MOYENNE-GENERALE(K)`, `NOTE-MATHS of TAB-NOTES(NUMERO-ELEVE)` ou plus simplement `NOTE-MATH(NUMERO-ELEVE)`.

2.1.5 Redéfinitions de zone

Une variable peut en redéfinir une autre. Ceci revient à proposer plusieurs découpages d’une même zone. Exemple :

```
01 LIGNE pic X(80).
01 LIGNE-1 redefines LIGNE.
  02 ALPHA pic x(10).
  02 BETA  pic x(70).
01 LIGNE-2 redefines LIGNE.
  02 BETA  pic X(20).
  02 GAMMA pic X(60).
```

Les redéfinitions d’une même zone doivent être de la même longueur. Certains compilateurs refusent de redéfinir un elément qui a la clause `occurs`, comme dans :

```
01 JOURS pic X(2) occurs 7.
01 JOURS-INIT redefines JOURS pic X(14) value "LuMaMeJeVeSaDi".
```

On peut facilement contourner cette restriction, par exemple ainsi :

```
01 T-JOURS.
  02 JOURS pic X(2) occurs 7.
01 JOURS-INIT redefines T-JOURS pic X(14) value "LuMaMeJeVeSaDi".
```

2.2 Variables numériques et alphanumériques

2.2.1 Alphanumériques

- Variables alphanumériques: `pic X(30)`
- les groupes sont considérés comme des alphanumériques.

2.2.2 Variables numériques

- Entiers non-signés: `pic 9(6)`.
- Entiers signés: `pic s9(6) sign leading, separate`. (occupe 7 caractères).
- Nombre décimaux: `pic 9(6)v99`. Le `v` indique la position virtuelle du point décimal. (Occupe 8 caractères).
- Nombre décimaux signés: `pic s9(6)v99 sign leading, separate`. (9 caractères).

2.2.3 Variables de calcul

Sont déclarées avec une clause `usage [comp|binary|packed decimal]`, selon le type choisi pour la représentation interne. La taille occupée par les variables dépend des implémentations.

2.3 Variables éditées

Destinées à être lues par l'utilisateur. On ne peut pas faire de calcul avec, mais elles peuvent apparaître comme résultat d'un `move`, d'un `add`, d'un `compute` etc. Exemples:

```
pic  9999.99
pic  ZZ79.99
pic  ***9.99
pic +ZZZ9.99
pic -ZZZ9.99
pic 99/99/9999
```

- à la différence de `v`, le point "." est réel.
- "`Z`" : le zéro non significatif est remplacé par un blanc.
- "`+`" : le signe apparaît.
- "`-`" : signe moins si négatif, espace si positif.
- les signes "`,`" "`/`" "`0`" s'insèrent dans la chaîne. La lettre "`b`" insère un espace.

Il existe des variables alpha-numériques éditées:

```
77   TRUC pic  xx/bxx0.
```

2.4 Conditions

Ce ne sont pas des variables, mais des conditions associées à des variables.

```
...
   05 MOYENNE-GENERALE pic 99v99.
     88 RECU value 10 THRU 20.
     88 COLLE value 0 thru 9.99.
...
   01 REPONSE pic X.
     88 OUI value "O", "o".
     88 NON value "N", "n".
```

On pourra alors écrire `accept REPONSE;  if OUI then ....` au lieu de `accept REPONSE;  if REPONSE = "O" or "o" then ....`

Chapitre 3

Fichiers indexés

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

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

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

3.1 Déclaration d'un fichier

```
environment division.
input-output-section.
file-control.
  select F-VOITURES
  assign to "/donnees/voitures"
  organization indexed
  access random
  record key PLAQUE.
...
data division.
file section.
fd F-VOITURES.
01 ENR-VOITURE.
  02 PLAQUE pic X(10).
  02 MARQUE pic X(10).
  02 MODELE pic X(10).
  02 COULEUR pic X(10).
```

3.2 Accès aléatoire (random)

3.2.1 open

Un fichier indexé peut être ouvert dans un des modes suivants : `input`, `output`, ou `i-o` (abréviation de input-output). Exemple:

```
open i-o F-VOITURES.
```

3.2.2 read

```
move "123 ABC 45" to PLAQUE.
read F-VOITURES
invalid key
  display "pas de voiture dans le fichier"
not invalid key
  display "modele = " MODELE
end-read
```

3.2.3 write

```
move "123 ABC 45" to PLAQUE.
move "SIMCA" to MARQUE.
move "1000" to MODELE.
...
write ENR-VOITURE
invalid key
  display "voiture deja enregistree."
not invalid key
  display "ok."
end-write.
```

3.2.4 rewrite

```
move "123 ABC 45" to PLAQUE.
read F-VOITURES
invalid key
  display "voiture absente"
not invalid key
  move "BLEU" to COULEUR
  rewrite ENR-VOITURE
    invalid key
      display "*** ERREUR INTERNE (rewrite)"
    not invalid-key
      display "ok."
  end-rewrite
end-read.
```

3.2.5 delete

```
move "123 ABC 45" to PLAQUE.
delete F-VOITURES
```

```
invalid key
  display "voiture absente"
not invalid key
  display "ok"
end-delete.
```

3.2.6 close

```
close F-VOITURES.
```

3.3 Accès séquentiel (`séquential`)

En accès séquentiel, l'ouverture peut se faire dans l'un des 4 modes : `input, output, i-o, extend`. `Start` permet de se positionner sur un enregistrement. L'instruction `delete` n'est pas autorisée.

3.3.1 read

```
read F-VOITURES next
at end
   ....
not at end
   ....
end-read.
```

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

3.3.2 write, rewrite

```
write ENR-VOITURE.
```

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

3.3.3 start

```
move "123 XY 89" to PLAQUE.
start F-VOITURES, key = PLAQUE
invalid key
   move FAUX to TROUVE
not invalid key
   read F-VOITURES next
   at end
     move FAUX to TROUVE
   not at end
     move VRAI to TROUVE
   end-read
end-start.
```

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

3.4 Accès dynamique (dynamic)

Un fichier `dynamic` ne peut pas être ouvert en mode `extend`.

Instructions disponibles

- `open fichier mode`
- `start fichier key .... invalid key ...`
- `read fichier next at end ...`
- `read fichier invalid key ...`
- `write enregistrement invalid key ...`
- `rewrite enregistrement invalid key ...`
- `delete fichier invalid key ...`
- `close fichier`

3.5 Clés multiples

Exemple:

```
environment division.
input-output-section.
file-control.
  select F-VOITURES
  assign to "/donnees/voitures"
  organization indexed
  access random
  record key PLAQUE
  alternate record key MARQUE with duplicates.
...
procedure division.
...
  move "PEUGEOT" to MARQUE.
  start F-VOITURES key = MARQUE
  invalid key
    display "Pas de Peugeot"
  not invalid key
    move FAUX to FIN-FICHIER
    perform until FIN-FICHIER = VRAI
      read F-VOITURES next
      at end
        move VRAI to FIN-FICHIER
      not at end
        if MODELE = "403" then
          display PLAQUE, MODELE
        end-if
      end-read
    end-perform
  end-start.
```

Chapitre 4

Fichiers relatifs

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

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

Les différences principales sont que :

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

4.1 Déclaration d'un fichier

```
environment division.
input-output-section.
file-control.
  select F-ARTICLES
  assign to "/donnees/catalogue"
  organization relative
  access random
  relative key NUMERO-ARTICLE.
...
data division.
file section.
fd F-ARTICLES.
01 ENR-ARTICLE.
  02 DESIGNATION pic X(10).
  02 PRIX-UNITAIRE pic 9(6)v99.
...
working-storage section.
77 NUMERO-ARTICLE pic 9(3).
```

4.2 Accès aléatoire (random)

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

4.3 Accès séquentiel (sequential)

Les opérations autorisées sont

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

4.4 Accès dynamique (dynamic)

Combinaison des modes séquentiel et aléatoire:

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

Chapitre 5

Les sous-programmes

Un sous-programme se distingue d'un programme ordinaire parce qu'il comporte des paramêtres qui sont déclarés dans la LINKAGE SECTION de la DATA DIVISION. L'ordre des paramètres est fixé par la clause USING de PROCEDURE DIVISION.

L'appel d'un sous-programme se fait par CALL. Par défaut c'est un passage par référence qui se produit, mais on peut demander un passage par valeur.

5.1 Exemple

```
identification division.
program-id.  somme.

data division.
working-storage section.
77 I pic 9(3).
77 S pic 9(6).
linkage section.
01 A pic 9(3).
01 B pic 9(3).
01 C pic 9(6).

procedure division using A, B, C.
principale section.
premiere section.
debut.
   move 0 to S.
   perform varying I from A by 1 until A > B
      add I to S
   end-perform.
   move S to C.
   exit program.
end program somme.

identification division.
program-id.  essai.

data division.
working-storage section.
```

```
01 PREMIER pic 9(3).
01 SECOND  pic 9(3).
01 TROISIEME pic 9(6).

procedure division.
premiere section.
debut.
  display "Donnez deux nombres"
  accept PREMIER, SECOND.
  call "SOMME" using
              by content PREMIER, SECOND,
              by reference TROISIEME.
  display TROISIEME.
  stop run.
end program essai.
```

5.2 À noter

- On peut mettre plusieurs programmes dans le même fichier source.
- Le premier argument de CALL peut être une variable qui contiendra, au moment de l'exécution, le nom du sous-programme à appeler. C'est une forme de chargement dynamique de sous-programmes.
- On peut se débarasser d'un sous-programme chargé en mémoire par l'ordre `CANCEL sousprogramme`.
- On peut partager des variables entre plusieurs sous-programmes. Utiliser l'attribut EXTERNAL dans la déclaration des données.