Notions avancées : sous-programmes

1. Principes

- Un **programme principal** (ou programme **appelant**) écrit en COBOL peut appeler un (ou plusieurs) **sous-programme(s)** (ou programme(s) **appelé(s)** rédigés en COBOL ou dans un autre langage, et compilés séparément. Après édition de liens, l'ensemble constituera **une seule unité d'exécution**.

- Le contrôle est passé au sous-programme et, après son exécution, le traitement du programme principal reprend à l'instruction qui suit l'appel du sous-programme.

- Dans la norme **ANS74**, Les données traitées ou paramètres sont communes aux deux programmes; programme principal et sous-programme se transmettent, suivant un protocole fixé, les **adresses** des paramètres (passage par adresses ou références) ; il n'y a donc **PAS** de duplication des valeurs.

  La norme **ANS85**, permet au contraire le choix, lors de l'appel du sous-programme, entre le passage par par valeurs (`CONTENT`) et par adresses (`REFERENCE`).

- Dans le sous-programme, les paramètres doivent être décrits en `LINKAGE SECTION` ; dans le programme principal, les trois sections de la `DATA DIVISION` conviennent. Toutefois, en ANS74, que ce soit dans le programme principal ou dans le sous-programme, les descriptions des paramètres doivent être obligatoirement de niveau `01` ou `77`. ANS85 lève cette restriction en autorisant en paramètres des données élémentaires ou de groupe, de niveau quelconque.

2. Sous-programme COBOL

Un sous-programme est un programme (COBOL) classique dont il diffère éventuellement sur trois points :

- Il comprend une `LINKAGE SECTION` pour la description des paramètres dits **formels**.

- La déclaration de la division des traitements est complétée d'une clause `USING` fixant l'ordre des paramètres.

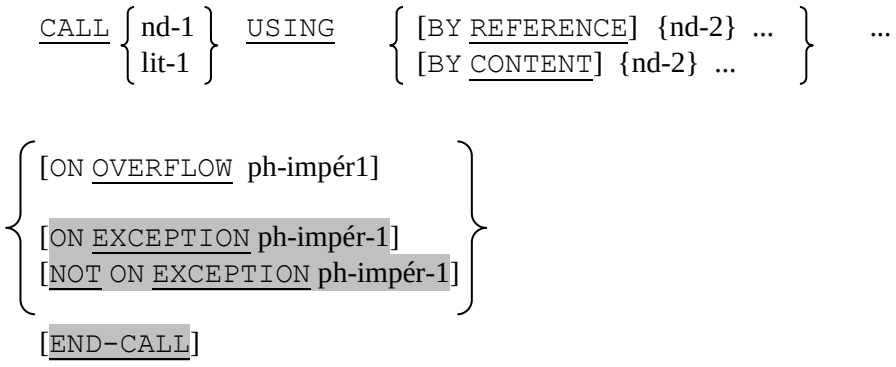
  `PROCEDURE DIVISION` [`USING` nd-1 [,nd-2]...].

Remarque : S'il n'y a aucun paramètre, `LINKAGE SECTION` et clause `USING` sont omises.

- L'instruction `EXIT PROGRAM` (ou `GOBACK`) doit être utilisée pour redonner le contrôle au programme principal.

Remarque : L'exécution d'une instruction `STOP RUN` à l'intérieur d'un sous- programme provoque l'arrêt immédiat et définitif du travail en cours (y compris le programme principal).

3. Appel d'un sous-programme



- lit-1 est le nom de programme appelé
- nd-1 est une donnée alphanumérique susceptible de contenir le nom du programme appelé.
- nd-2 ... sont les paramètres dits d'appels, décrits dans la `DATA DIVISION` du programme principal et de niveau 01 ou 77, (ou autre). Les index et les structures de fichier (FD) ne doivent pas être utilisés en paramètres.
- Option par défaut de passage des paramètres : `BY REFERENCE`
- Un mode de passage (`CONTENT` ou `REFERENCE`) est transitif sur tous les paramètres le suivant.
- Les listes de paramètres du programme principal et du sous-programme doivent être en concordance parfaite du point de vue ordre, type, représentation interne et n° de niveau des paramètres. Il n'y a aucune corrélation entre les noms de données du sous-programme et ceux du programme principal.
- Le sous-programme est initialisé lors du premier appel et conserve son état lors des appels ultérieurs, sauf s'il est doté de l'attribut `INITIAL`, ou si `CANCEL` a été exécuté dans le programme principal.
- Les clauses `OVERFLOW` et `EXCEPTION` sont validées si le programme appelé ne peut être chargé (ou ne réside pas) en mémoire, ou si la place disponible pour l'y charger est insuffisante.

4. Instructions spécifiques des sous-programmes

4.1. CANCEL

```
CANCEL { nd-1  } { nd-2  } ...
       { lit-1 } { lit-2 }
```

nd-1, nd-2, lit-1, lit-2,.... désignent des noms de sous-programmes. Cette instruction permet de libérer la zone mémoire occupée par le(s) sous-programme(s) cité(s) .

Pratiquement, elle permet de garantir que, lors du prochain appel, le sous-programme sera dans son **état initial**.

4.2 EXIT PROGRAM

```
EXIT PROGRAM.
```

- Marque la fin logique du sous-programme
- Cette instruction doit être la seule du paragraphe qui la contient
- Si le sous-programme possède l'attribut `INITIAL`, l'instruction est semblable à `CANCEL`, mais sans libération de ressources mémoire.
- Si le programme n'est pas un programme appelé, l'instruction est sans effet.
- Dans les dialectes GOBACK a un fonctionnement similaire.

5. Objets externes partagés

- Tout objet défini dans un programme COBOL lui est **interne**, et les ressources qu'il nécessite sont associées au seul programme.
- Un objet sera dit **externe** si ses ressources sont attachées non à un objet (programme) particulier, mais à l'unité d'exécution résultante. Pour cela, il faut le doter de l'attribut EXTERNE.
- Une clause `EXTERNAL` peut être associé à une spécification de fichier dans un programme, ou à un groupe de niveau `01` en `WORKING-STORAGE SECTION`. Tous les **éléments subordonnés** (zones de données) sont **automatiquement externalisés**.

  ```
  FD nom-fich [IS EXTERNAL]
  .....
  ```

  Dans ce cas, si un programme appelé contient des structures de mêmes caractéristiques (noms, formats), l'article chargé dans le programme principal lui sera accessible.

- La clause VALUE est interdite pour des objets externalisés (sauf nom-condition).

6. Exemple d'utilisation

MOY est un sous-programme permettant de calculer la moyenne de deux nombres.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MOY.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 SOMME PIC 9(5) COMP.

LINKAGE SECTION.
77 MOYENNE PIC 9(4)V99 COMP.
01 NOMBRES.
     02 N1 COMP-1.
     02 N2 PIC 9(4).


PROCEDURE DIVISION USING NOMBRES MOYENNE.
PO.  ADD N1 N2 GIVING SOMME
     DIVIDE 2 INTO SOMME GIVING MOYENNE.
P1.  EXIT PROGRAM.
```

Soit un programme principal comportant en DATA DIVISION les descriptions suivantes :

```
77 N1 PIC 9(4)V99 COMP.
01 COUPLE1.
     04 NB1 COMP-1.
     04 NB2 PIC 9(4).
01 COUPLE2.
     02 A1 COMP-1.
     02 A2 PIC 9(4).
```

On peut écrire en division des traitements :

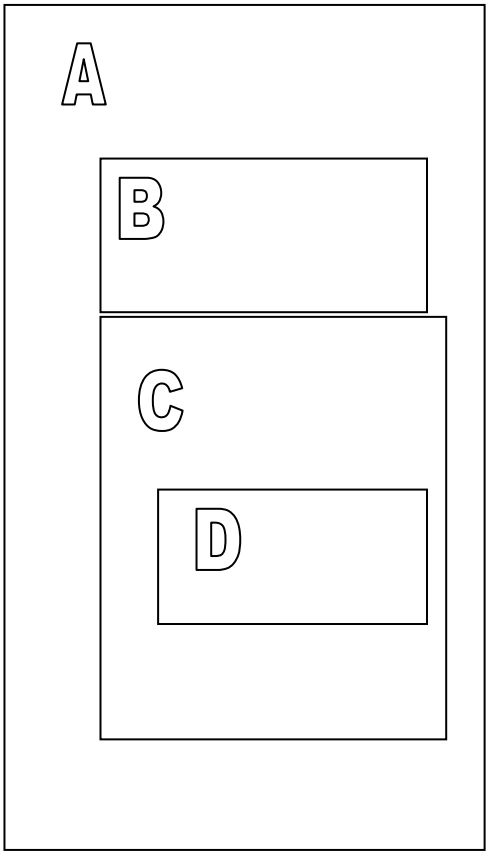
```
     '
     '
     CALL "MOY" USING COUPLE2 N1
     '
     '
     CALL "MOY" USING COUPLE1 N1
```

PROGRAMMES CONTENUS

En ANS85, un programme-source peut en **contenir** entièrement un (ou plusieurs) autres .
Cette disposition permet aux programmes de partager certaines ressources.

L'ensemble de ces programmes est compilé en même temps et ne donne création qu'à un seul programme-objet.
A ne pas confondre avec la notion classique de sous-programmes (programmes appelant et appelé étant des unités de compilations différentes, liées en une seule unité d'exécution).

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
.............

IDENTIFICATION DIVISION.
PROGRAM-ID. B.
END-PROGRAM  B.

IDENTIFICATION DIVISION.
PROGRAM-ID. C.
.............

IDENTIFICATION DIVISION.
PROGRAM-ID. D.
.............

END-PROGRAM  D.

END-PROGRAM  C.

END-PROGRAM  A.
```



♦ A contient **directement** B et C et **indirectement** D

C contient directement D.

♦ Dans cette structure l'en-tête `END-PROGRAM` nom est obligatoire.

♦ Les programmes contenus peuvent faire référence à des ressources des programmes contenants. Pour cela, il faut déclarer, **dans le programme contenant**, la clause `GLOBAL` pour des groupes de niveau `01` en `FILE SECTION` ou `WORKING-STORAGE SECTION`, ou des phrases `FD`.
Tout programme **contenu**, directement ou indirectement dans un tel programme-contenant pourra référencer la donnée sans la décrire à nouveau.
♦ Tous les éléments subordonnés à une structure globale sont globalisés.

♦ Normalement un programme **contenu** ne peut être appelé que par celui le contenant directement. La clause `COMMON`, dans son `PROGRAM-ID` permettra qu'il le soit également par des programmes de la même unité, autres que celui le contenant.