Sommaire

A Concepts de base

A1 Généralités **2**
A2 Concepts et terminologie **2**

A21 sous-programmes internes et externes **2**
A22 le transfert du contrôle **2**
A23 le passage des données **3**
A24 notion de variables globales et locales [clause GLOBAL] **6**
A25 données externes [clause EXTERNAL] **8**
A26 les sous-programmes communs [clause COMMON] **11**
A27 l'initialisation des sous-programmes (verbe CANCEL + clause INITIAL] **16**

B Formats **17**

B1 Le verbe CALL **17**
B2 La phrase PROCEDURE DIVISION USING **18**
B3 La phrase ENTRY..USING **19**
B4 La clause END PROGRAM **21**
B5 Le verbe CANCEL **21**
B6 Le verbe EXIT PROGRAM **22**
B7 Les clauses EXTERNAL et GLOBAL **23**
B8 Les attributs d'un sous-programme (INITIAL et COMMON) **24**
B9 Les formats pour les applications MULTI-LANGAGES **25**

C Exemples

C1 cas simple de S/P internes et externes avec données globales,locales,externes ... **29**
C2 cas avec un fichier déclaré «externe» **30**
C3 cas complet **34**

D La Modularistion avec le verbe PERFORM

D1 Généralités-concepts **37**
D2 Les formats **41**
D3 exemples **63**

A) Concepts de base

A1 Généralités

A11 définition

Un sous-programme est un programme qui est appelé par un autre (et qui peut lui-même faire appel à d'autres sous-programmes ...etc...) pour réaliser une action (=> algorithme) auquel on passe éventuellemnt des informations-paramètres et qui retourne éventuellement des informations-résultats.

Le nom «sous-programme» est très général ; il englobe les termes : «procédures» , «fonctions» , «modules» ... que l'on trouve dans la «littérature» spécifique aux langages et aux méthodes de développement.

En cobol , un sous-programme est un programme à part entière (=> composé des divisions et sections habituelles : IDENTIFICATION DIVISION , ENVIRONMENT DIVISION,DATA DIVISION ,....etc...).

A12 rôle du sous-programme

En principe , on écrit un sous-programme lorsque l'action réalisée par celui-ci peut être réutilisée par d'autres applications. (ce qui évite de réécrire du code inutilement et permet de gagner de la place en mémoire).

Remarques :

Il arrive également qu'on écrive des sous-programmes spécialement pour une (et une seule) application lorsqu'on veut sérier les problèmes et modulariser : en cobol , pour traiter ces cas , on ne fait pas réellement des sous-programmes (qui sont des programmes complets) mais on utilise l'instruction PERFORM qui permet de se brancher sur des «modules» (appartenant au même programme) placés après l'instruction STOP RUN.
[CF le paragraphe D]

A2 Concepts et terminologie

A21 sous-programmes internes ou externes

Les programmes appelants et appelés peuvent être placés dans le même fichier source ; dans ce cas on dit que les sous-programmes sont internes : ici ,après la compilation ,on obtient un seul programme «objet» (.obj) et après l'édition des liens , un seul (et relativement volumineux) exécutable (.exe).
Ce procédé est parfois appelé «modularisation statique».

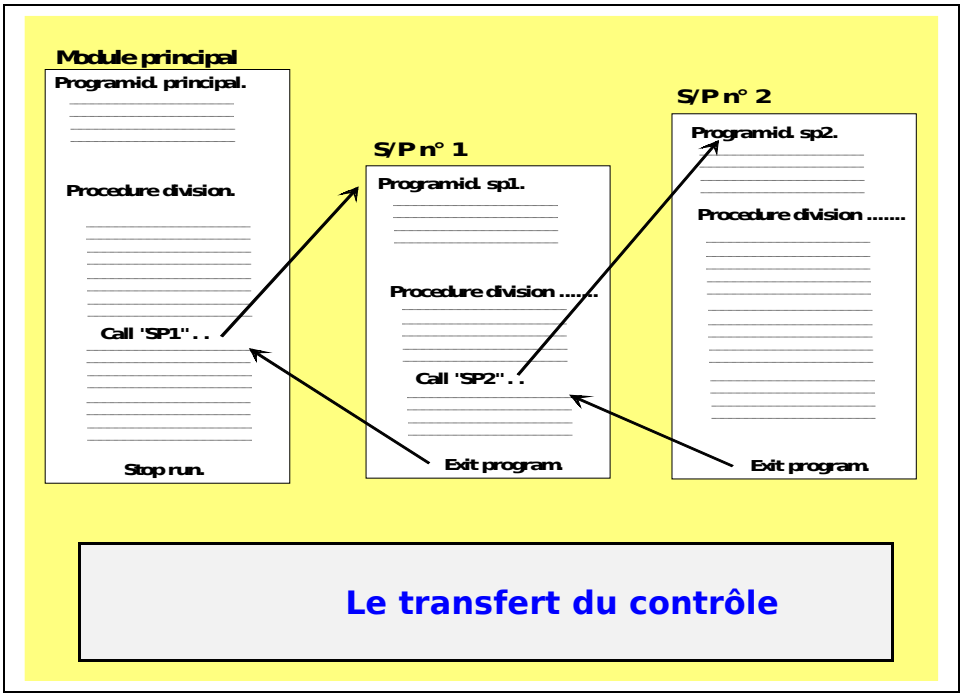
[NB : en cobol , les S/P internes se terminent par la clause END PROGRAM *nom-du-program-id*]

En réalité , il est souvent plus judicieux de travailler en «modularisation dynamique» qui consiste à faire un fichier source par sous-programme et à compiler et linker séparément tous les modules : ceux-ci sont alors appelés «sous-programmes externes».
Ensuite , on constitue des bibliothèques avec les programmes objet ou avec les exécutables et on les utilise à partir de n'importe quelle autre application (écrite éventuellement dans un langage différent).

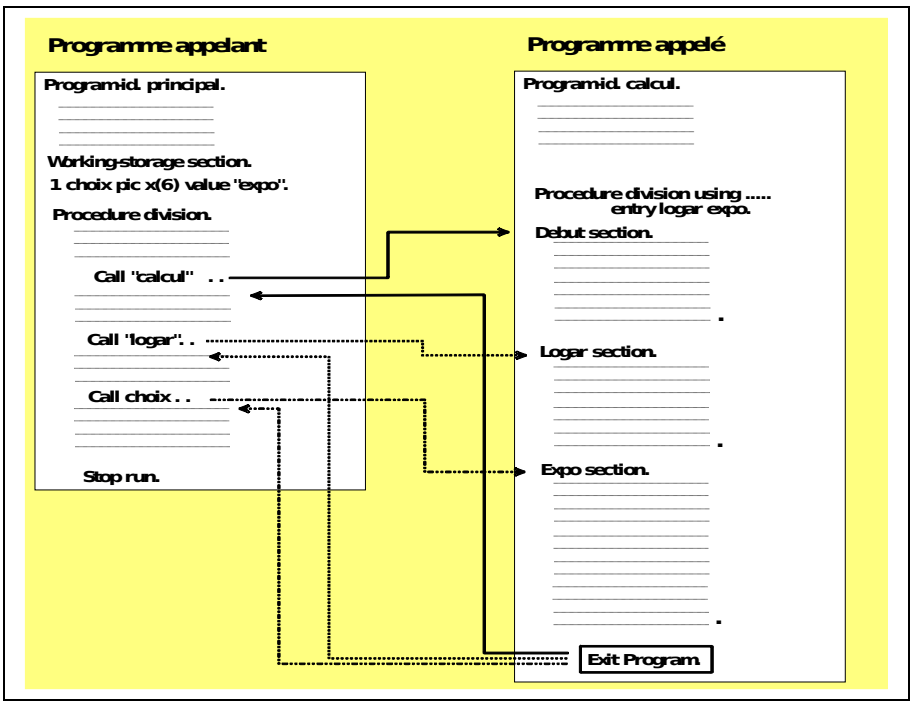
A22 le transfert du contrôle

Le programme appelant appelle et passe le contrôle au sous-programme par l'intermédiaire de l'instruction CALL. [CF format : paragraphe B1]

Quand son exécution est terminée , le sous-programme «rend la main» au programme appelant grâce à l'instruction EXIT PROGRAM : le module principal (ou maître ou appelant ..) continue en séquence à partir de l'instruction qui suit le verbe CALL.



NB : en cobol ,on peut également appeler et exécuter une partie d'un sous-programme (à partir d'un point d'entrée) néanmoins ce procédé (*résultant d'une mauvaise analyse du problème*) est rarement utilisé.



sous-programme à plusieurs points d'entrée

A23 le passage de données

En COBOL , comme dans tous les langages , on peut passer des paramètres à un sous-programme :

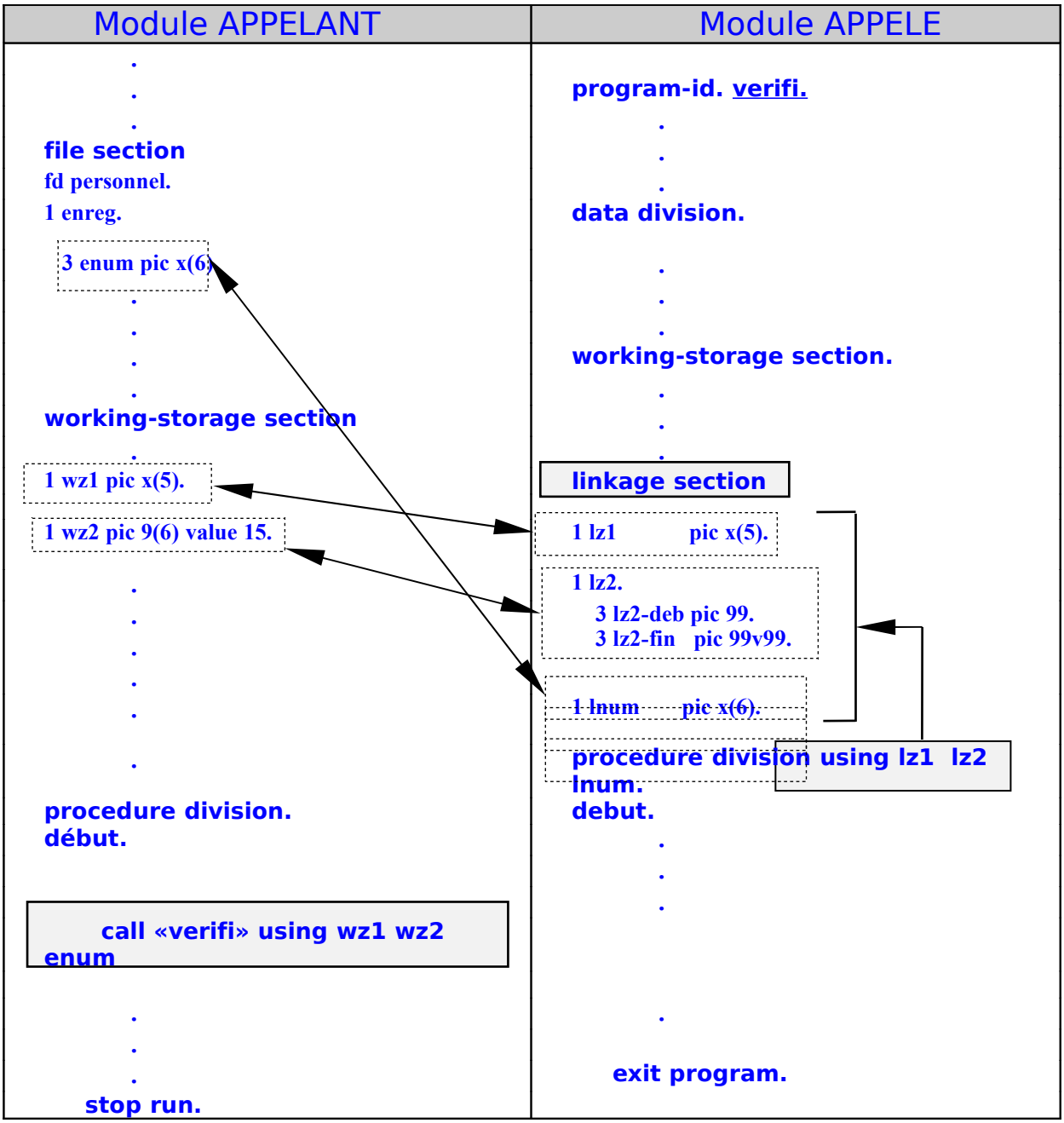
par valeur : au retour dans le programme principal , les zones passées ne sont pas modifiées.
par adresse : les zones passées (en fait c'est l'adresse de ces zones qui est transmise) peuvent être modifiées par le sous-programme.

Pour «passer»❶ des zones à un sous-programme , il suffit de les citer derrière la clause USING du verbe CALL en précisant les options «par valeur» (BY CONTENT) ou «par adresse» (BY REFERENCE).

❶ *en réalité , on ne passe aucune donnée ; le sous-programme utilise l'adresse des zones appartenant au module appelant [CF Principe de la linkage section - page suivante]*

Pour utiliser les zones «transmises» dans le sous-programme , il faut décrire celles-ci dans une LINKAGE SECTION placée en DATA DIVISION et les inscrire (dans le même ordre) dans la clause PROCEDURE DIVISION USING ...

schéma de principe



Principe de la linkage section

La linkage section est utilisée dans le sous-programme pour décrire des données qui existent dans le programme appelant.

Les descriptions en linkage section ne génèrent aucune zone réelle ; elles permettent d'énumérer les données communes entre le programme appelant et le programme applelé.

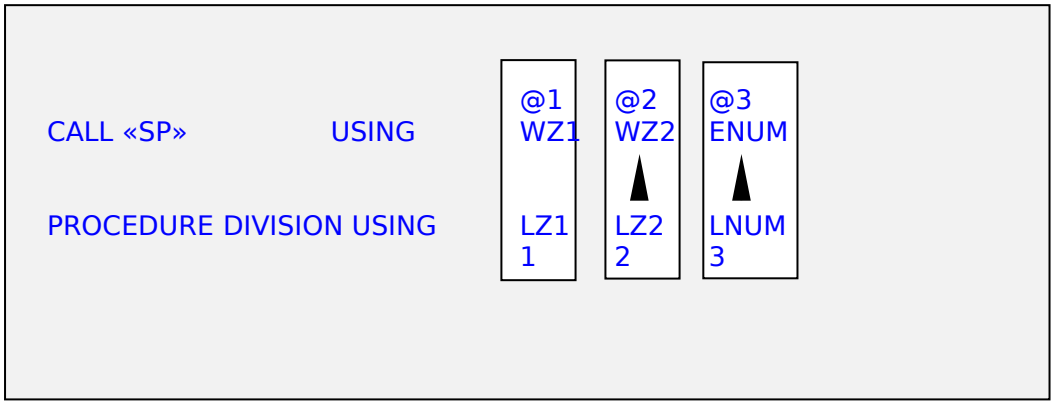
==> Aucune zone n'est allouée pour le sous-programme pour la linkage section :

l'adressage de ces données est résolu, lors de l'exécution, par correspondance entre la liste des zones mentionnées dans la phrase «PROCEDURE USING ...» et la liste des zones (réelles et uniques appartenant à la data division du programme principal) qui sont indiquées dans le «CALL USING ...» [NB : *parfois on utilise le terme «passage de paramètres par liste d'adresses»*]

Les données citées dans le «CALL USING...» et dans le «PROCEDURE DIVISION USING...» doivent être dans le même ordre.

Les noms-données inscrits dans la linkage section n'ont pas besoin d'être les mêmes que ceux du programme appelant (il est même préférable d'éviter de procéder ainsi) car le lien entre la donnée du programme appelant et celle de l'appelé ne se fait pas par le nom mais par l'emplacement (la position) dans la liste des noms cités dans la PROCEDURE USING»...

1ère position dans procedure division using => 1ère zone dans la liste du call.. using
2ème position dans procedure division using => 2ème zone dans la liste du call.. using
. . .



La structure de la linkage section est la même que celle de la working-storage section : chaque zone-groupe (niveau 1) ou zone élémentaire (niveau 77) doit être unique dans le programme.

Les données décrites dans une linkage section ne peuvent être utilisées par le sous-programme que si elles sont déclarées dans la clause «PROCEDURE USING ...» ou si elles sont subordonnées à celles-ci.

NB : dans la phrase PROCEDURE DIVISION USING.. les noms-données cités doivent correspondre à des nombres niveaux 1 ou 77 en linkage section.

les zones décrites en linkage section et celles déclarées dans le module appelant (et mises en commun par CALL USING..) doivent avoir la même longueur.
initialisation des zones de la linkage section : la clause VALUE est interdite en linkage section (sauf avec les nombres niveaux 88 => noms conditions et variables booléennes)

A24 Notion de variables globales et locales [clause GLOBAL]

Les termes «local» et «global» ne concernent que les programmes et leurs sous-programmes internes.

❶ Définition d'une variable locale

Implicitement (==> si on ne met aucune clause) , toutes les variables sont «locales» c'est-à-dire connues (ou «visibles») uniquement par le programme dans lequel elles sont décrites.

Une variable décrite dans un sous-programme , portant le même nom qu'une autre appartenant au programme appelant (ou à un autre sous-programme) est totalement indépendante de celle-ci .

==> quand une donnée locale et globale porte le même nom
c'est toujours la locale qui est prise en compte.

remarques spécifiques au cobol

A partir du moment où on décrit une variable dans un sous-programme , elle est systématiquement locale même si celle-ci est déclarée comme globale dans un autre module :

si dans un sous-programme, on veut utiliser une donnée décrite appartenent au programme appelant , il faut la déclarer «globale» dans celui-ci et ne faire aucune description dans l'appelé.

Inversement , si on ne décrit pas une donnée et qu'elle n'est pas déclarée globale ailleurs , on a une erreur à la compilation (du type «nom-donnée non déclaré»).

exemple

```
IDENTIFICATION DIVISION.
PROGRAM-ID. tp0.
WORKING-STORAGE SECTION.
1 a pic 99 value 12.
PROCEDURE DIVISION.                                   résultats obtenus
debut.
   display "au début      dans TP0 :  a = " a  ------>  a = 12
   call "plus5"
   display "après plus5   dans TP0 :  a = " a  ------>  a = 12
   call "moins2"
   display "après moins2  dans TP0 :  a = " a  ------>  a = 12
   stop run.

IDENTIFICATION DIVISION.
PROGRAM-ID. plus5 initial.
WORKING-STORAGE SECTION.
1 a pic 99 value 64.
PROCEDURE DIVISION.
debut.
   add 5 to a
   display "avant exit    dans PLUS5  a = " a  ------>  a = 69
   exit program.
end program plus5.

IDENTIFICATION DIVISION.
PROGRAM-ID. moins2 initial.
WORKING-STORAGE SECTION.
77 a pic -9V,99.
77 b pic 9v99 value 1.89.
PROCEDURE DIVISION.
debut.
   subtract 2 from b giving a
   display "avant exit    dans MOINS2 a = " a  ------>  a = -0,11
   exit program.
end program moins2.

end program tp0.
```

❷ Définition d'une variable globale

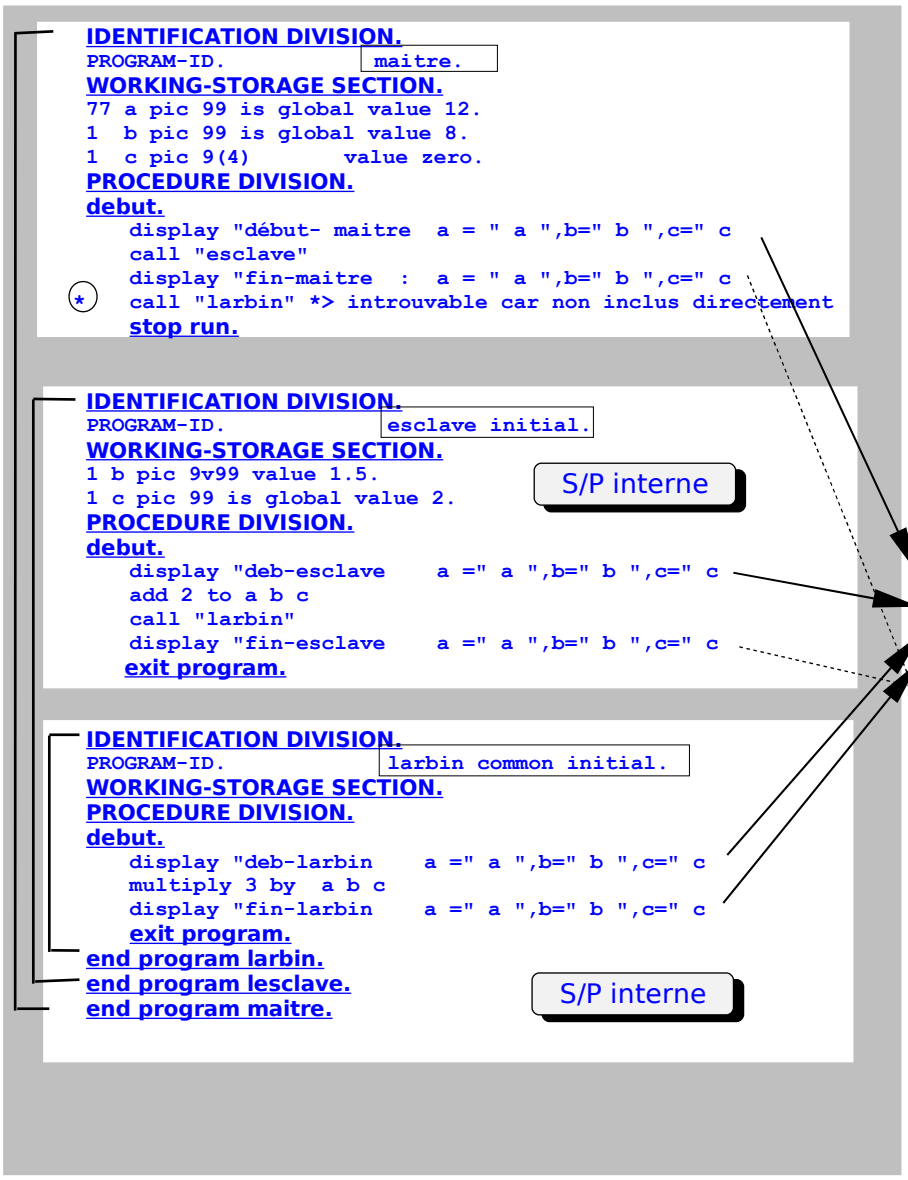
Une donnée est dite «globale» lorsqu'elle est «vue» ou «visible» (et donc accessible) par tous les sous-programmes internes.

remarques spécifiques au cobol

En cobol , pour qu'une donnée soit globale , il suffit de la déclarer avec la clause «..IS GLOBAL» dans le programme appelant ; dans tous les sous-programmes dans lesquels on veut l'utiliser , on ne fait aucune description.

Les variables déclarées globales dans un sous-programme qui contient lui-même des sous-programmes internes ne sont connues que par ceux-ci.

exemple



résultats obtenus

	a	b	c
debut-maitre	12	08	0000
deb-esclave	12	150	02
deb-larbin	14	08	04
fin-larbin	42	24	12
fin-esclave	42	350	12
fin-maitre	42	24	0000

NB : D'autres entités peuvent être déclarées globales (ex les fichiers,..)

CF parag B : règles et formats [clause GLOBAL et clause COMMON]

parag C : exemples sur le partage des fichiers

conseil :

L'utilisation des variables globales n'est pas conseillée car le sous-programme est lié à son module appelant et donc peu réutilisable.
D'autre part des modifications intervenant dans le source du programme appelant peuvent impliquer des mises à jour dans les sous-programmes utilisant les variables globales.

Un bon sous-programme doit être indédendant : une fois mis au point il ne doit plus être retouché : il réalise à l'aide de ses propres données (et celles qu'on lui a éventuellement passées comme arguments) l'action ou la fonction pour laquelle il a été prévu.

=> Plutôt que de déclarer des données globales , il vaut mieux les passer (par adresse ou par valeur) au sous-programme qui en a besoin.

A25 les données externes [clause EXTERNAL]

Le concept de «donnée externe» est sensiblement analogue à celui évoqué (ci-dessus) pour les données globales mais il concerne essentiellement les sous-programmes externes.

global : signifiant connu par tous les éléments du source considéré.
externe :signifiant global par rapport à l'ensemble des sources constituant l'application.

Les informations dites «externes» sont celles dont on a besoin dans un sous-programme externe et qui existent déjà ailleurs (déjà décrites et utilisées dans d'autres programmes ou sous-programmes de l'application).

❶ Définition

Une donnée externe est une information qui existe et qui a déjà été décrite dans un autre programme de l'application en cours de développement et qu'on veut pouvoir utiliser directement (sans la passer par valeur ou par adresse).

remarques spécifiques au cobol

En cobol , pour qu'on puisse utiliser une donnée externe , il faut la déclarer avec la clause «..IS EXTERNAL» dans le programme où elle existe (description initiale) ainsi que dans les sous-programmes dans lesquels on veut l'utiliser.

Les variables déclarées externes dans un programme (ou un sous-programme) sont «visibles» par tous les éléments internes ou externes qui les déclarent avec la clause IS EXTERNAL.

principe

Quand on a besoin d'utiliser une variable existant ailleurs , et qu'on la décrit avec IS EXTERNAL , le système sait qu'il s'agit d'une étiquette (référence externe) et non d'une description réelle par un nom-donnée : c'est à l'édition des liens qu'il la remplace par l'adresse réelle de la zone.

conseil :

Le problème est le même que pour les données globales : L'utilisation des variables externes n'est pas conseillée car le sous-programme considéré est lié au programme contenant la donnée réelle et donc peu réutilisable.
D'autre parts des modifications intervenant dans le source du programme contenant les données réelles peuvent impliquer des mises à jour dans les modules qui utilisent ces variables externes.

Un bon sous-programme doit être indédendant : une fois mis au point il ne doit plus être retouché : il réalise à l'aide de ses propres données (et celles qu'on lui a éventuellement passées comme arguments) l'action ou la fonction pour laquelle il a été prévu.

> => Plutôt que de déclarer des données externes , il vaut mieux les passer (par adresse ou par valeur) au sous-programme qui en a besoin.

En cobol , il existe néanmoins des cas où on ne peut pas passer l'information par valeur ou par adresse : c'est le cas des fichiers qui est traité dans l'exemple suivant.

Le partage des fichiers : principe et exemple:

Δ :
[*L'organisation interne des fichiers peut être spécifique à une machine et à son système d'exploitation par exemple : les fichiers utilisés sur BULL DPS8,8000,9000 avec leur «attribute region»,leurs «rcw» ,leurs «offset» n'ont aucun point commun avec ceux qui ont été manipulés sur PC pour écrire ce support.*
=> dans ce cours , toutes les informations-fichiers ne sont peut-être pas valables pour votre système.]

Chaque fichier est caractérisé par une zone nommée «File Connector» qui contient un buffer pour la lecture des données , un champ pour indiquer l'état du fichier (ouvert ,...) , un pointeur qui indique sur quel caractère on est positionné ,...etc...

En cobol , ce file-connector n'est pas transmissible à un sous-programme par valeur ou par adresse. (il est toujours possible de traiter ce problème dans un module écrit dans un autre langage tel que le «C» dans lequel le nom du fichier logique est en fait l'adresse de ce file-connector que l'on peut passer au sous-programme choisi.)

> ⇒ si on veut utiliser un fichier existant dans l'état où il se trouve , on ne peut que :
>
> - le déclarer en «global» si on veut y accéder dans un sous-programme interne
> - le déclarer en «externe» si on veut y accéder depuis un sous-programme externe.

exemple simple

A partir du fichier **c:\gestion\etu02.txt** décrit ci-dessous , on veut éditer toutes les personnes dont le nom commence par un groupe de n lettres.(par ex toutes les personnes dont le nom commence par «LAB»)

					NOM	PRENOM					
n°	1	02	ETU	KKKK054	BOBARD	JEAN	0000000000	560914	TEF	1225	%%%%%
n°	2	02	ETU	KKKK132	BOVEAU	JEAN	0000000000	590914	OS1	1622	%%%%%
n°	3	02	ETU	KKKK028	CARMICON	JEAN	0000000000	420914	TEF	1725	%%%%%
n°	4	02	ETU	KKKK030	CASBONBINI	MARCEL	0000000000	620914	TEF	1236	%%%%%
n°	5	02	ETU	KKKK050	CHIALOUX	FLORENCE	0000000000	410914	TEF	1725	%%%%%
n°	6	02	ETU	KKKK060	CHIASSOUX-BERNARD	ANTHONY	0000000000	520914	OS1	1825	%%%%%
n°	7	02	ETU	KKKK140	DESSALON	OLIVIA	0000000000	510914	IEF	1625	%%%%%
n°	8	02	ETU	KKKK026	DUFER	CASIMIR	0000000000	420914	IEF	1550	%%%%%
n°	9	02	ETU	KKKK056	FITROCHON	BENOIT	0000000000	560914	OS1	1642	%%%%%
n°	10	02	ETU	KKKK005	FOUMI	PIERRE	0000000000	400813	OS1	1700	%%%%%
n°	11	02	ETU	KKKK034	LABAF	JEAN	0000000000	700914	TEF	1625	%%%%%
n°	12	02	ETU	KKKK138	LABROUILLE	ISIDORE	0000000000	550914	IPE	1625	%%%%%
n°	13	02	ETU	KKKK130	LE CHAUVE	STANISLAS	0000000000	580914	OS1	1622	%%%%%
n°	14	02	ETU	KKKK032	MACARONETTINI	LEON	0000000000	690914	TEF	1350	%%%%%
n°	15	02	ETU	KKKK136	MULLER	AGATHE	0000000000	550914	IEF	1525	%%%%%
n°	16	02	ETU	KKKK058	MURINI	ANGE	0000000000	690914	OS1	1105	%%%%%
n°	17	02	ETU	KKKK018	N'GORE	JEAN	0000000000	560914	TEF	1650	%%%%%
n°	18	02	ETU	KKKK002	PIOLA	JEAN	0000000000	420914	TEF	1625	%%%%%
n°	19	02	ETU	KKKK062	PISSOUNET	MARCEL	0000000000	540914	OS1	1925	%%%%%

```
n° 20    02  ETU  KKKK016  RASPITOU          JEAN              0000000000  450914  TEF  1425  %%%%%
n° 21    02  ETU  KKKK052  ROUPIGNAC         THIERRY           0000000000  500914  TEF  1425  %%%%%
n° 22    02  ETU  KKKK134  ROYER-MOLLOUX     VERONIQUE         0000000000  550914  IEF  1624  %%%%%
```

Externa = Pg principal
nterne

```
IDENTIFICATION DIVISION.
PROGRAM-ID. externa.
ENVIRONMENT DIVISION .
input-output section.
file-control.
     select fetu02 assign to "c:\gestion\etu02.txt".
data division.
file section.
fd fetu02 is external.
1 enreg pic x(80).
WORKING-STORAGE SECTION.
77 wtrouve            pic 9     value 0.
   88 trouve                    value 1.
77 wdeb-nom           pic x(20) value "LAB".
77 wdeb-long          pic 9(5)  value 3.
1 i            external pic 999.
1 wctf1 external              pic 9.
        88 fin-fichier value 1.
procedure division .
debut.
    move 0 to wctf1 i
    open input fetu02
    read fetu02 at end
         set fin-fichier to true
    end-read
    perform until fin-fichier
       call   "recherch" using wtrouve wdeb-nom wdeb-long
       cancel "recherch"
       if trouve
          then
             display "n° " i "==> " enreg(13:20) enreg(33:20)
             read fetu02 at end
                  set fin-fichier to true
             end-read
       end-if
    end-perform
    close  fetu02
    stop run.
```

Δ

On aurait pu faire une application très simple, mais, le but étant de montrer l'utilisation des variables externes , on a sciemment utiliser une logique et une analyse du problème peu performantes.

La 1 ère lecture de Fetu02.txt a lieu dans ce module

Les autres sont effectuées dans «recherch» tant qu'on ne trouve pas de personne avec un nom commençant par «LAB»:

quand c'est le cas, on traite (retour au programme principal dans lequel est fait le display de l'élément trouvé)

et on relit à nouveau l'article suivant dans cette séquence.

Recherch = S/P externe
nterne

```
IDENTIFICATION DIVISION.
PROGRAM-ID. recherch.
ENVIRONMENT DIVISION.
input-output section.
file-control.
     select fetu02 assign to "c:\gestion\etu02.txt".
data division.
file section.
fd fetu02 is external.
1 ze  pic x(80).
WORKING-STORAGE SECTION.
77 wctf1          external pic 9.
   88 fin-fichier value 1.
1 i external pic 999.
linkage section.
77 ltrouve   pic 9.
   88 lok value 1.
77 ldeb-nom  pic x(20).
1  ldeb-long pic 9(5).
procedure division using ltrouve ldeb-nom ldeb-long.
debut.
    move 0 to ltrouve
    perform until fin-fichier or lok
       if ze(13:ldeb-long) = ldeb-nom(1:ldeb-long)
          then
             add 1 to i
             set lok to true
          else
             read fetu02 at end
                  set fin-fichier to true
             end-read
```

résultat obtenu

```
n° 1==> LABAF        JEAN
n° 2==> LABROUILLE   ISIDORE
```

```
    end-if
end-perform
exit program.
```

A26 les sous-programmes communs [clause COMMON]

Un sous-programme commun (option COMMON dans le paragraphe PROGRAM-ID) est un programme qui peut être appelé par tous les autres sous-programmes de l'application sauf par ceux correspondant aux cas suivants :

❶ Le sous-programme qui possède la clause COMMON ne peut s'appeler lui-même (CF le chapître sur la récursivité) ni être appelé par un sous-programme qu'il contient.

❷ Si le sous-programme qui possède la clause COMMON est directement inclus dans un autre , il ne peut être appelé que ceux qui sont inclus au même niveau ou par le programme contenant.
Par contre, ce même sous-programme «common» peut faire appel à des sous-programmes de niveau supérieur (par exemple ceux de même niveau que son contenant).

Exemple simple

sans la clause COMMON

```
PROGRAM-ID A
    PROGRAM-ID B
    END PROGRAM B

    PROGRAM-ID C
    END PROGRAM C

END PROGRAM A
```

A peut appeler B et C
B ne peut pas appeler C ou A
C ne peut pas appeler B ou A

avec la clause COMMON

```
PROGRAM-ID A
    PROGRAM-ID B
    END PROGRAM B

    PROGRAM-ID C COMMON.
    END PROGRAM C

END PROGRAM A
```

A peut appeler B et C
B peut appeler C
C ne peut pas appeler B

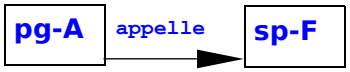
Exemple complet sur les sous-programmes internes avec ou sans «COMMON»

on a écrit les programmes pg-A, sp-B, sp-C, sp-D, sp-E, sp-F, sp-G avec les imbrications suivantes :

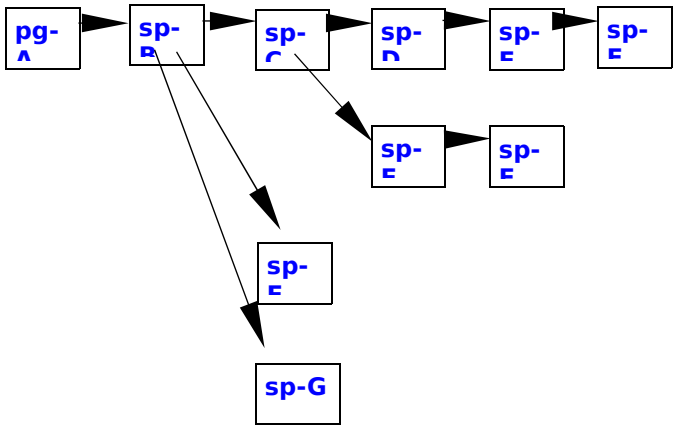
```
PROGRAM-ID. pg-A.

    PROGRAM-ID. sp-B.


    END PROGRAM sp-B.

    PROGRAM-ID. sp-C COMMON.


        PROGRAM-ID D COMMON.


            PROGRAM-ID. sp-E.


            END PROGRAM sp-E.

        END PROGRAM sp-D.

    END PROGRAM sp-C.

     PROGRAM-ID. sp-F COMMON.


    END PROGRAM sp-F.


    PROGRAM-ID. sp-G.


    END PROGRAM sp-G.

END PROGRAM pg-A.
```

et on a testé les cas suivants :

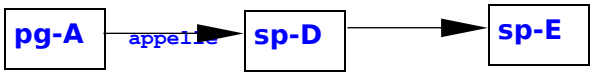
❶



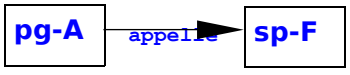
❷



❸



❹



programme cobol

programme principal pg-A

```
IDENTIFICATION DIVISION.
PROGRAM-ID. pg-a.
WORKING-STORAGE SECTION.
77 wqui pic xxxx value "pg-a".
77 i        pic 99 is global value 0.
1  m              is global .
   3 m1     pic x(18) value "   je suis dans : ".
   3 wcible  pic xxxx.
   3 m2     pic x(26) value " , j'ai été appelé par : ".
   3 wsource pic xxxx.
PROCEDURE DIVISION.
debut.
    add 1 to i
    move wqui   to wsource
    move "sp-f" to wcible
    call wcible
       on overflow

          move " , impossible d'appeler: " to m2          ❶
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-f"
-------------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-b" to wcible
    call wcible
       on overflow

          move " , impossible d'appeler: " to m2          ❷
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-b"
-------------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-d" to wcible
    call wcible
       on overflow

          move " , impossible d'appeler: " to m2          ❸
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-d"
-------------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-f" to wcible
    call wcible
       on exception

          move " , imposisble d'appeler: " to m2          ❹ = ❶
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-f"
    display "fin traitement"
    stop run.
```

essais de la clause COMMON

résultat obtenu

```
n° 01  je suis dans   :   sp-f ,   j'ai été appelé par   :   pg-a
n° 02  je suis dans   :   sp-b ,   j'ai été appelé par   :   pg-a
n° 03  je suis dans   :   sp-c ,   j'ai été appelé par   :   sp-b
n° 04  je suis dans   :   sp-d ,   j'ai été appelé par   :   sp-c
n° 05  je suis dans   :   sp-e ,   j'ai été appelé par   :   sp-d
n° 06  je suis dans   :   sp-f ,   j'ai été appelé par   :   sp-e
n° 07  je suis dans   :   sp-c ,   impossible d'appeler  :   sp-e
n° 08  je suis dans   :   sp-f ,   j'ai été appelé par   :   sp-b
n° 09  je suis dans   :   sp-b ,   impossible d'appeler  :   sp-g
n° 10  je suis dans   :   pg-a ,   impossible d'appeler  :   sp-d
n° 11  je suis dans   :   sp-f ,   j'ai été appelé par   :   pg-a
```

sous-programme sp-B (SANS COMMON)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. sp-b initial.
WORKING-STORAGE SECTION.
77 wqui pic xxxx value "sp-b".
PROCEDURE DIVISION.
debut.
    move " , j'ai été appelé par : " to m2
    move wqui to wcible
    display "n° " i m
 ----------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-c" to wcible
    call wcible
       on overflow
          move " , impossible d'appeler: " to m2
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-c"
 ----------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-f" to wcible
    call wcible
       on overflow
          move " , impossible d'appeler: " to m2
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-f"
 ----------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-g" to wcible
    call wcible
       on overflow
          move " , impossible d'appeler: " to m2
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-g"

    exit program.
end program sp-b.
```

sous-programme sp-C (AVEC COMMON)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. sp-c common initial.
WORKING-STORAGE SECTION.
77 wqui pic xxxx value "sp-c".
PROCEDURE DIVISION .
debut.
    move " , j'ai été appelé par : " to m2
    move wqui to wcible
    display "n° " i m
    ----------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-d" to wcible
    call wcible
       on overflow
          move " , impossible d'appeler: " to m2
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-d"
    ----------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-e" to wcible
    call wcible
       on overflow
          move " , impossible d'appeler: " to m2
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-e"
   exit program.
```

sous-programme sp-D (AVEC COMMON)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. sp-d common initial.
WORKING-STORAGE SECTION.
77 wqui pic xxxx value "sp-d".
PROCEDURE DIVISION.
debut.
    move " , j'ai été appelé par : " to m2
    move wqui to wcible
    display "n° " i m
-------------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-e" to wcible
    call wcible
       on overflow
          move " , impossible d'appeler: " to m2
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-e"
    exit program.
```

sous-programme sp-E (SANS COMMON)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. sp-e initial.
WORKING-STORAGE SECTION.
77 wqui pic xxxx value "sp-e".
PROCEDURE DIVISION.
debut.
    move " , j'ai été appelé par : " to m2
    move wqui to wcible
    display "n° " i m
-------------------------------------------------------------------------------
    add 1 to i
    move wqui   to wsource
    move "sp-f" to wcible
    call wcible
       on overflow
          move " , impossible d'appeler: " to m2
          display "n° " i  m1 wqui m2 wcible
    end-call
    cancel "sp-f"
     exit program.
end program sp-e.

end program sp-d.

end program sp-c.
```

sous-programme sp-F (AVEC COMMON)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. sp-f common initial.
WORKING-STORAGE SECTION.
77 wqui pic xxxx value "sp-f".
PROCEDURE DIVISION.
debut.
    move " , j'ai été appelé par : " to m2
    move wqui to wcible
    display "n° " i m
    exit program.
end program sp-f.
```

sous-programme sp-G (SANS COMMON)

```
IDENTIFICATION DIVISION .
PROGRAM-ID. sp-g initial.
WORKING-STORAGE SECTION.
77 wqui pic xxxx value "sp-g".
PROCEDURE DIVISION.
debut.
    move " , j'ai été appelé par : " to m2
    move wqui to wcible
    display "n° " i m
exit program.
end program sp-g.
```

```
end program pg-a.
```

A27 initialisation d'un sous-programme [verbe CANCEL + clause INITIAL]

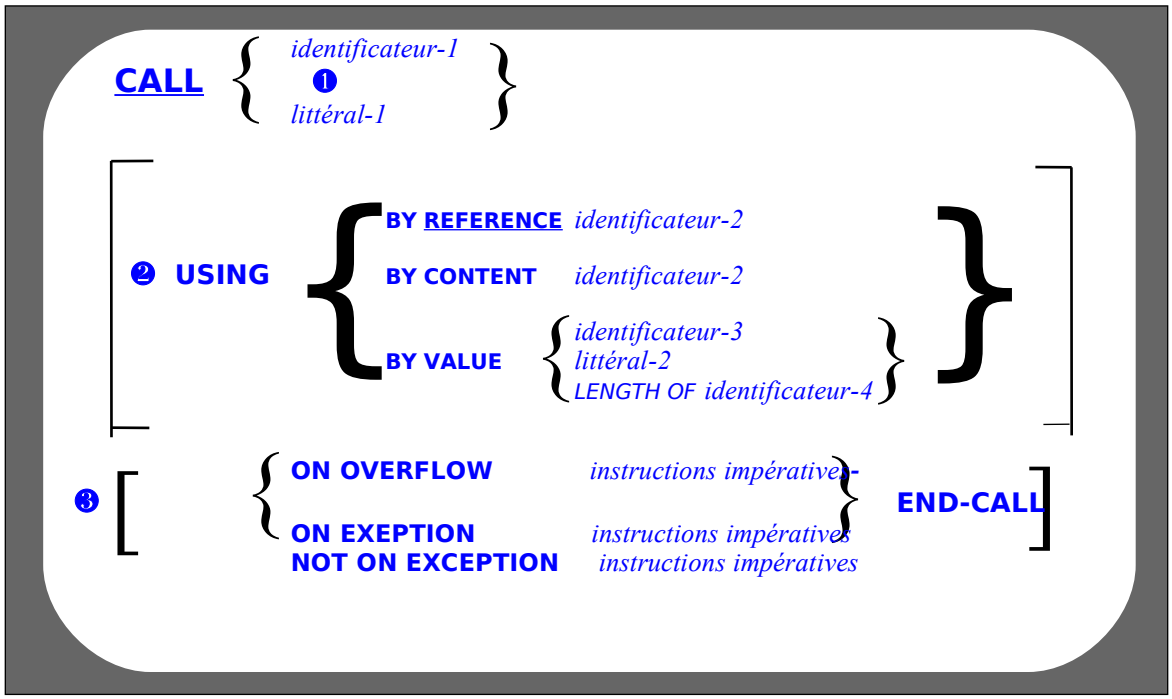
Un sous-programme est exécuté avec ses données à l'état initial (correspondant essentiellement aux clauses VALUE en working-storage section) quand:

- **il est applelé pour la première fois.**
- **il est applelé la première fois après l'exécution d'un CANCEL.**
- **à chaque fois qu'il est appelé , si le paragraphe PROGRAM-ID du sous-programme contient la clause INITIAL.**

B) Règles et formats

B1 le verbe CALL

b11 format



b12 règles

❶ Le nom du sous-programme appelé

Littéral-1 ou identificateur-1 doit être alphanumérique.

NB cobol microsoft : si le nom ou le littéral est un caractère numérique ou un caractère non ASCII , cela correspond à l'appel d'une routine système (ex call X«AF» using ...)

La valeur de littéral-1 ou identificateur-1 est le nom (PROGRAM-ID) du sous-programme que l'on veut appeler ou le nom d'un point d'entrée dans le sous-programme.

Il est possible d'appeler deux (ou plus) sous-programmes qui portent le même nom à condition que l'un soit inclus (interne) dans celui qui contient le CALL et que l'autre soit externe.

❷ Le passage de paramètres

La clause using ne doit être incluse que si dans le programme appelé , il y a une phrase PROCEDURE DIVISION USING .. ; dans ce cas , le nombre et l'ordre des paramètres doivent être les mêmes.

Tous les nom-données cités dans cette clause «USING» représentent des zones qui peuvent se trouver en FILE SECTION,WORKING-STORAGE SECTION,COMMUNICATION SECTION, LINKAGE SECTION du programme appelant.

Les nom-données cités dans la clause USING peuvent avoir des nombres niveaux 1 ou 77 ou compris entre 2 et 49.

L'option BY REFERENCE qui correspond au passage d'arguments par adresse (ou plus précisément par liste d'adresses - CF les concepts de base-) est implicite : les zones inscrites derrière cette clause peuvent être modifiées par le sous-programme appelé.
NB : si on souhaite qu'un sous-programme retourne des informations , il faut lui passer des zones BY REFERENCE dans lesquelles seront rangés les résultats.

❸ Traitement des anomalies

ON EXCEPTION est synonyme de ON OVERFLOW : les instructions inscrites dans cette clause sont exécutées lors de toute impossibilité d'appel du sous-programme (quelle que soit la raison : manque de ressource ,..).

NB Δ : sur certains systèmes ,quand le sous-programme appelé ne peut être chargé en mémoire (faute de place) , l'exécution du programme appelant se poursuit à l'instruction suivante : il est donc plus prudent de prévoir la clause ON EXCEPTION (ou NOT..).

B2 la phrase «PROCEDURE DIVISION ...»

b21 format

b22 règles

❶ Présence de la clause USING (*cf page 4,5*)

Elle n'est nécéssaire que si , dans le programme appelant , on a fait un CALL ... USING ... ; dans ce cas les nom-donnée-1 , ..., nom-donnée-n qui suivent correspondent (dans le même ordre) aux zones passées avec le USING du verbe CALL.

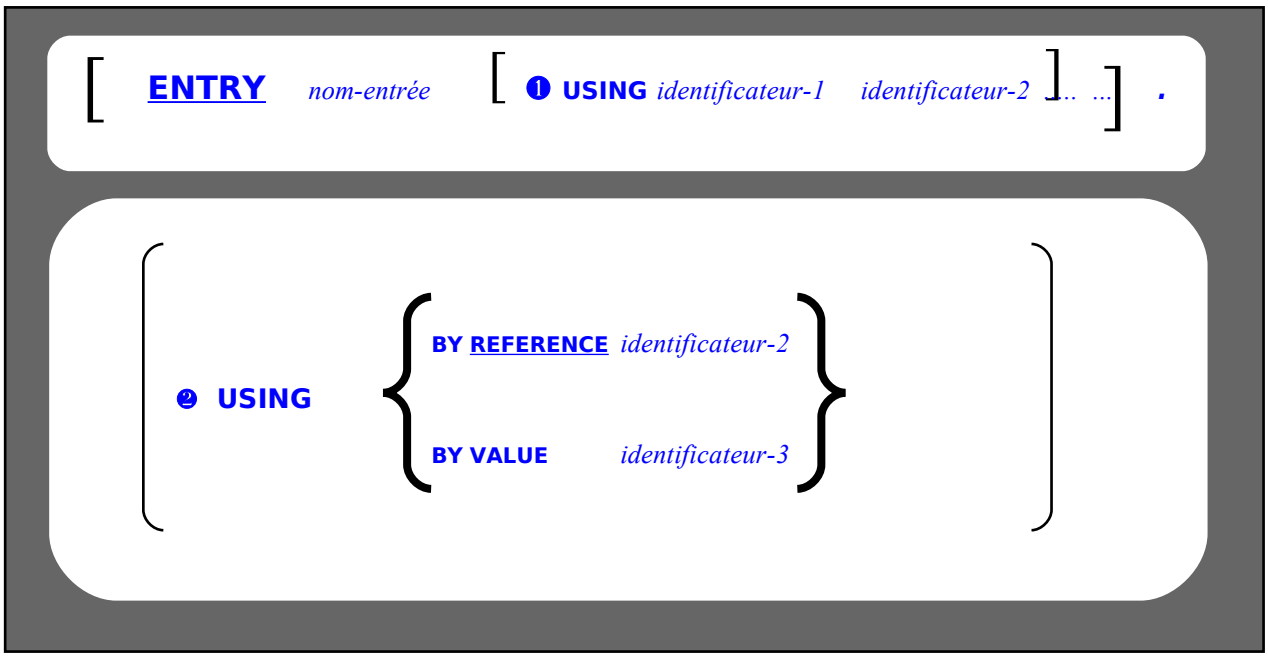
nom-donnée-1,..nom-donnée-n doivent avoir des niveaux 1 ou 77 en linkage section.

La linkage section qui contient la description des zones utilisées dans le sous-programme , c'est-à-dire celles définies dans le USING de PROCEDURE DIVISION , peut éventuellemnt contenir d'autres données supplémentaires (CF paragraphe suivant sur la clause ENTRY).

Si on a fait un « CALL .. USING a b c » et que dans le sous-programme appelé on oublie (volontairement ou pas) de mentionner certaines zones : par exemple « PROCEDURE DIVISION USING a c » ; les informations omises sont inaccessibles , mais cela ne génére aucune erreur.

B3 la phrase «ENTRY ...»

b31 format



b32 règles

Remarques

Le format le plus couramment utilisé est le ❶ , le n° ❷ qui est équivalent au USING du verbe CALL permet de choisir au niveau du point d'entrée le type de passage de paramètres.

La méthode pour définir des points d'entrée dans un sous-programme peut varier d'un système à l'autre (compilateur) : l'exemple général cité à la page 3 où des nom-procédure placés en colonne 8 constituent les points d'entrée n'est pas valable en cobol microsoft (CF l'exemple de la page suivante).

Règles *(cobol microsoft)*

Le nom de chaque point d'entrée (il peut y en avoir plusieurs - CF page 20 -) doit être un littéral alphanumérique ou un nom-donnée décrit en PIC X(.) ; seuls les 8 premiers caractères sont significatifs (6 seulement sur certaines machines).

Le nom choisi pour un point d'entrée doit être unique (il ne doit surtout pas être le même que celui d'un autre sous-programme de l'application).

Le point d'entrée est en fait une étiquette à partir de laquelle le sous-programme appelé s'exécute lorsqu'on rencontre l'instruction CALL «nom du point d'entrée»...

La clause USING a la même fonction que celle mentionnée dans PROCEDURE DIVISION : elle permet de définir les paramètres qui seront utilisables par cette portion de sous-programme (lesquels doivent se trouver en linkage section).
Avec le USING de ENTRY , on peut passer à chaque point d'entrée , des paramètres différents.

La clause ENTRY ne fonctionne pas avec des sous-programmes internes.

Pour accéder aux divers points d'entrée , il faut que le sous-programme ait été appelé au moins une fois par son nom de PROGRAM-ID.

Le USING pour un point d'entrée a un fonctionnement identique au USING de PROCEDURE DIVISION (la seule différence est que ce dernier concerne l'ensemble du sous-programme et non une partie)

NB cobol microsoft : il existe une directive de compilation «STICKY-LINKAGE» (ou «NOSTICKY-LINKAGE») qui peut influer sur le fonctionnement de la clause USING de ENTRY : elle permet aux autres points d'entrée l'accès aux paramètres des autres points d'entrée à condition qu'ils aient été appelé au moins une fois.

exemple :

```
IDENTIFICATION DIVISION.
PROGRAM-ID. entree.
WORKING-STORAGE SECTION.
77 a pic 999 value 16.
77 b pic 999 value 49.
77 c pic 999 value 3.
77 d pic 999 value 2.
77 z pic 9   value 0.
77 wchoix pic x(8) value "racin2".
PROCEDURE DIVISION.
debut.
    call "calcul" using z
    call "puiss2" using by reference a d
    display "cas n°" z "   a=" a ",b=" b ",c=" c
    move 0 to a
    call wchoix  using by reference b
    display "cas n°" z "   a=" a ",b=" b ",c=" c
    move 0 to b
    call "divid2" using by reference c
    display "cas n°" z "   a=" a ",b=" b ",c=" c
    stop run.
```

résultat SANS la directive STICKY-LINKAGE

```
bonjour , lz=0
cas n° 0        a=256,b=049,c=003
cas n° 0        a=000,b=001,c=003
cas n° 0        a=000,b=000,c=000
```

CALCUL = S/P externe

```
$set sticky-linkage
IDENTIFICATION DIVISION.
PROGRAM-ID. calcul.
WORKING-STORAGE SECTION .
linkage section.
77 la pic 999 .
77 lb pic 999.
77 lc pic 999.
77 ld pic 999.
77 lz pic 9.
PROCEDURE DIVISION using lz.
debut.
    display "bonjour ,lz=" lz
    exit program.
entry "puiss2" using la ld
    compute la = la ** ld
    add 1 to lz
    exit program.
entry "racin2" using lb
    compute lb = lb ** (1 / ld)
    add 1 to lz
    exit program.
entry "divid2" using lc
    compute lc = lc / ld
    add 1 to lz
    exit program.
```

résultat AVEC la directive STICKY-LINKAGE

```
bonjour , lz=0
cas n° 1        a=256,b=049,c=003
cas n° 2        a=000,b=007,c=003
cas n° 3        a=000,b=000,c=001
```

commentaires :

puiss2 (élévation à la puissance ld=2)	connait lz	si on a mis la directive STICKY-LINKAGE
racin2 (racine carrée)	connait lz et ld	si on a mis la directive STICKY-LINKAGE
divid2 (division par ld = 2)	connait lz et ld	si on a mis la directive STICKY-LINKAGE

B4 la clause END PROGRAM

b41 format

END PROGRAM *nom-du programme* *(dans program-id)*

b42 règles

❶ cas de plusieurs sous-programmes

Si le paragraphe «PROGRAM-ID» est placé entre le «PROGRAM-ID» et le «END PROGRAM» d'un autre module, la clause «END PROGRAM» doit précéder le «END PROGRAM» de celui-ci.

```
PROGRAM-ID A
  PROGRAM-ID B
  END PROGRAM B
END PROGRAM A
```

```
PROGRAM-ID A
END PROGRAM A
PROGRAM-ID B
END PROGRAM B
```

```
PROGRAM-ID A
  PROGRAM-ID B
END PROGRAM A
END PROGRAM B
```

interdit

B5 le verbe CANCEL

b51 format

CANCEL { *identificateur-1* / *littéral-1* } .....

b52 règles

Littéral-1 doit être un littéral alphanumérique et identificateur-1 un nom-donnée décrit en PIC X(.).

L'ordre CANCEL permet d'effacer en mémoire le sous-programme considéré ; cela garantit que lors du prochain appel , le système n'utilisera pas une copie mais le chargera dans son état initial.

Le CANCEL d'un programme s'applique également à tous les sous-programmes internes inclus.

Dans un sous-programme appelé , il est interdit de faire un CANCEL de l'appelant ou d'autres programmes de niveau supérieur.
Ce qui équivaut à dire qu'on ne doit jamais faire CANCEL pour un sous-programme qui n'est pas encore passé par EXIT PROGRAM (retour à l'applelant).

Le verbe CANCEL est un effacement « explicite » d'un sous-programme ; il existe un cancel automatique dit «implicite» qui se produit dans les cas suivant :

- avant l'appel du sous-programme si celui-ci contient la clause INITIAL.
- quand l'application (.exe) se termine (si on relance l'exécutable , tous les sous-programmes sont mis à l'état initial).

Si on fait un CANCEL d'un programme qui n'a pas été appelé , il n'y a aucune action : le contrôle est passé à l'instruction suivante.

LE contenu des informations « externes » d'un sous-programme ne change pas lorsqu'on fait un CANCEL.

Lorqu'un CANCEL intervient , un CLOSE (implicite) est généré pour chaque fichier ouvert (le file-connector interne de chaque fichier est remis à l'état fermé) : toutes les USE procedure associées à ces fichiers ne sont pas exécutées.

B6 le verbe EXIT PROGRAM

b61 format

EXIT PROGRAM.

b62 règles

Cette instruction permet de quitter un sous-programme pour revenir à l'instruction du programme appelant qui suit le CALL.

Avec certains compilateurs EXIT PROGRAM doit être la seule instruction d'un paragraphe.

Si on passe sur un EXIT PROGRAM alors qu'on n'a pas appelé aucun sous-programme , l'exécution se poursuit à partir de l'instruction suivante.

L'exécution d'un EXIT PROGRAM dans un sous-programme appelé qui possède l'attribut INITIAL génère un CANCEL de celui-ci.

B7 les clauses GLOBAL et EXTERNAL

b71 formats

pour les fichiers

❶ **FD** *nom-du fichier*
[is external]
[is global]
block contains ....
1 *nom-du record* [is global] ....

pour les zones en data division (notamment en working-storage section)

❷ **1** (ou autre nbre niveau) *nom-donnée*
[redefines *nom-donnée*]
[is external]
[is global]
pic ....

pour le report writer

❸ **RD** *report-name*
[is global]
code ....

b72 règles

❶ **cas des fichiers**

Si on veut qu'un fichier soit externe , il suffit de déclarer « EXTERNAL » au niveau de la phrase FD mais en aucun cas au niveau de la description de l'article.

Par contre , l'enregistrement (niveau 1) peut être GLOBAL si on veut l'utiliser dans un sous-programme interne.

GLOBAL au niveau de la phrase FD n'implique pas GLOBAL au niveau du record lu. (au niveau de la description en FD , on agit sur la zone file-connector du fichier.)

Si le fichier est une imprimante (c'est-à-dire qu'on a une clause LINAGE en FD) , le LINAGE-COUNTER est GLOBAL si on a mis cette clause en FD ; il est EXTERNAL si on a choisi cette option ; il est GLOBAL et EXTERNAL si les deux clauses ont été placées dans la phrase FD.

❷ cas des descriptions de données (en working)

Dans tous les cas , la clause EXTERNAL est imcompatible avec VALUE (sauf le VALUE des niveaux 88).

Si une zone-groupe (niveau 1) a un attribut GLOBAL ou EXTERNAL , les zones sous-groupes et élémentaires subordonnées héritent de ces proriétés.

On ne doit pas déclarer EXTERNAL une zone redéfinie (avec REDEFINES.) , par contre on peut redéfinir une donnée externe.

Une donnée GLOBALE ne doit être décrite que dans le programme appelant alors qu'une information EXTERNAL est déclarée dans les deux avec le même nom.

NB : la clause GLOBAL peut être mentionnée en linkage section.

Des zones en data division portant le même nom-donnée ne peuvent pas avoir l'attribut GLOBAL.

❸ cas du report writer)

Si la « report description » contient GLOBAL , les registres LINE-COUNTER et PAGE-COUNTER (CF le chapitre sur l 'éditeur d'état) sont globaux.

B8 les attributs d'un sous-programme : COMMON et INITIAL

b81 formats

b82 règles

Si le sous-programme est interne , le nom du programme doit être unique dans le source considéré.

COMMON ne concerne que les sous-programmes internes : (tout le principe est expliqué pages 11 à 15)

Si PROGRAM est spécifié , la clause COMMON et/ou INITIAL doit être présente.

Quand un sous-programme contenant la clause INITIAL est appelé , toutes les zones de sa data division sont remises à l'état initial .

B9 les formats pour les applications MULTI-LANGAGES

b91 généralités

La plupart des formats étudiés dans les paragraphes précédents ont été complétés au fur et à mesure de l'évolution des différentes versions des compilateurs , par des clauses permettant de gèrer la communication (appel , transfert de données , retour de code , ..) entre des sous-programmes écrits dans différents langages.

Ce domaine concerne les questions suivantes :

- comment passer en paramètre l'adresse d'un sous-programme [CF exemple C3]
- comment appeler un sous-programme à l'aide de son adresse. [CF exemple C3]
- comment appeler un sous-programme écrit (ou pas) dans un autre langage.
- comment envoyer des informations à un sous-programme écrit (ou pas) dans un autre langage.
- comment récupérer des données provenant d'un sous-programme écrit (ou pas) dans un autre langage.

Ces clauses que nous allons décrire succintement peuvent également servir pour une application écrite totalement en cobol. [CF exemple C3]

Elles font volontairement l'objet d'un paragraphe à part pour ne pas surcharger les notions essentielles vues précédemment car elles sont moins fréquemment utilisées .

Le but de ce paragraphe est d'énumérer (tout en expliquant brièvement) certaines phrases omises dans les paragraphes B1 à B8 afin que le lecteur ne soit pas surpris en les découvrant dans certains programmes. C'est pourquoi nous n'entrerons pas dans les détails de syntaxe ou de règles d'utilisation . Néanmoins , pour fournir un support relativement complet , nous avons testé ces éléments dans l'exemple commenté du paragraphe C3.

b92 la convention.

SPECIAL-NAMES..
CALL-CONVENTION *nombre* IS *nom mnémonique* .

Cette clause permet de choisir un nom mnémonique pour définir un type de langage (en principe , le langage dans lequel a été écrit le sous-programme que l'on veut appeler).

Le nombre doit être compris entre 0 et 65535.

[en binaire sur 16 bits ;en fait, seuls les bits 0,1,2 sont utilisés

bit 0 : 0 => analyse des paramètres de la droite vers la gauche
1 => gauche vers la droite
bit 1 : 0 => le programme appelant renvoie des paramètres
1 => appelé
bit 2 : 0 => le code retourné par le sous-programme sera mis dans le RETURN-CODE
1 => le registre RETURN-CODE ne sera pas mis à jour
les phrases RETURNING ne sont pas concernées par ce bit]

Si on ne met pas cette clause CALL-CONVENTION , par défaut le nombre est 0 qui signifie COBOL STANDARD.

EXEMPLE :

```
SPECIAL-NAMES.
       CALL-CONVENTION 0 is MICROSOFT-C
       CALL-CONVENTION 2 is FORTRAN
       CALL-CONVENTION 3 is PASCAL.
```

Le nom mnémonique est ensuite utilisé de la façon suivante :

❶ CALL MICROSOFT-C « nom-du-programme » USING paramètres ...
pour appeler un module écrit en C

❷ PROCEDURE DIVISION PASCAL USING paramètres..
pour spécifier la convention que ce programme pourrait nécéssiter pour être appelé par un autre écrit dans un autre langage.

b93 description et récupération de l'adresse d'un programme (cf C3)

les zones dans lesquelles on veut mettre l'adresse d'une procédure (programme) doivent être décrite avec la clause :

USAGE IS PROCEDURE-POINTER

un pointeur sur une procédure est une zone susceptible de contenir l'adresse d'un point d'entrée (nom fixé par la phrase ENTRY) ou d'un programme (nom qui se trouve dans le paragraphe PROGRAMM-ID).
Dans ce type de donnée , on peut récupérer l'adresse d'un programme écrit dans un autre langage.

Une zone en « usage procedure-pointer » doit toujours être élémentaire : si on met cette clause en niveau 1 , seules les données élémentaires subordonnées seront utilisables.

Une telle zone est renseignée à l'aide du verbe SET

```
SET    nom-donnée-1 (décrit en usage procedure-pointer)

TO    { nom-donnée-2 (usage procedure-pointer)              }
      { ENTRY   { identificateur }                          }
                { littéral       }
```

identificateur (ou littéral) est le nom du programme ou du point d'entrée.

b94 autre format du verbe CALL (cf C3)

```
CALL [nom-mnémonique] { identificateur-1                     }
                      { nom de pointeur sur procédure-1      }
                      { littéral-1                           }

[ USING { BY REFERENCE  nom de pointeur sur procédure-2                         } ]
[       { BY CONTENT    nom de pointeur sur procédure-3                         } ]
[       { BY VALUE      { nom de pointeur sur procédure-4                    }  } ]
[       {               { [littéral-2 [SIZE IS littéral-3]]                  }  } ]

[ GIVING     {  INTO          identificateur-2 } ]
[ RETURNING  {  ADDRESS OF    identificateur-3 } ]
```

La clause SIZE définit le nombre de caractères (pic X) à passer: si cette clause est présente , littéral-2 doit être numérique en COMP-5.

Si la clause SIZE n'est pas mentionnée , le nombre (par défaut) de bytes transmis est de 4 ou égal à la valeur fixée avec la directive LITVAL-SIZE.

Un paramètre passé BY VALUE ne peut dépasser 4 bytes.

ex : CALL MICROSOFT-C «essai» BY VALUE 6 SIZE 2. ==> passe la valeur 6 sur 2 bytes

Identificateur-3 doit être défini en LINKAGE section avec un niveau 1.

On ne peut pas faire CANCEL d'une zone en usage procedure-pointer.

[*« nb de l'auteur »:*

Mon compilateur (m-soft V4.5) ne semble pas connaître GIVING ; avec RETURNING qui semble être un synonyme : pas de problème]

Les clauses GIVING et VALUE sont réservées pour des valeurs numériques :

binaires : entiers	usage binary , comp,comp-4,comp-5
pointeurs	usage is pointer , procedure-pointer

Si on met la clause RETURNING *nom-de-zone* (ou GIVING) au niveau du verbe CALL , l'information retournée par le sous-programme appelé sera rangée dans *nom-de-zone* sinon elle sera stockée dans le registre spécial nommé RETURN-CODE.

Le registre RETURN-CODE est en principe en pic S9(4) COMP ; si on met la directive RTNCODE-SIZE«4» , il sera en pic S9(8) COMP.

b95 le verbe EXIT PROGRAM (cf C3)

EXIT PROGRAM

[{ GIVING / RETURNING } { ADDRESS OF nom-donnée-1 / Littéral-1 }]

Littéral-1 doit être un entier.

Nom-donnée-1 représente n'importe quelle zone en data division.

Ce format du EXIT PROGRAM permet au sous-programme appelé de retouner une valeur ou une adresse au programme appelant. (écrit dans n'importe quel langage).

Si le RETURNING (ou GIVING) n'a pas été mentionné dans le verbe CALL , le retour s'effectue dans le registre spécial RETURN-CODE sinon dans la zone suivant RETURNING (ou GIVING).

Les programmes qui utilisent ce format doivent être traité avec la directive système MF.

b96 les autres sorties de programme (*pour information*)

Le format de l'EXIT PROGRAM ci-dessus a été rajouté (avec les mêmes règles de fonctionnement) à tous les ordres cobol permettant de quitter un programme :

GOBACK

nom-donnée-1 { **GIVING** | **RETURNING** } { *ADDRESS OF* | *Littéral-1* }

GOBACK marque la fin logique d'un programme maître ou appelé.

Tout ce qui suit GOBACK n'est pas exécuté.

Si on passe sur un GOBACK dans un programme appelé , il se comporte comme un EXIT PROGRAM : le contrôle est rendu immédiatemment à l'appelant à partir de l'instruction qui suit le CALL.

Si on passe sur un GOBACK dans un programme maître , il se comporte comme un STOP RUN.

STOP RUN

nom-donnée-1 { **GIVING** | **RETURNING** } { *ADDRESS OF* | *Littéral-1* }

les sous-programmes externes

```
IDENTIFICATION DIVISION.
PROGRAM-ID. plus5 initial.
WORKING-STORAGE SECTION.
1 w1 pic 99.
1 w2 pic 99v99 is external.
1 w3 pic 9999  is global.
1 w4 pic 99v99 is global.
PROCEDURE DIVISION.
debut.
    add 5 to w1 w2 w3 w4
    exit program.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. plus8 initial.
WORKING-STORAGE SECTION.
1 w1 pic 99    is external.
1 w2 pic 99v99 is external.
1 w3 pic 9999  is external.
1 w4 pic 99v99 is global.
PROCEDURE DIVISION.
debut.
    add 8 to w1 w2 w3 w4
    exit program.
```

```
DENTIFICATION DIVISION.
PROGRAM-ID. plus1 initial.
WORKING-STORAGE SECTION.
1 w1 pic 99    is global external.
1 w2 pic 99v99 is external.
1 w3 pic 9999  is global external.
1 w4 pic 99v99.
PROCEDURE DIVISION.
debut.
    add 1 to w1 w2 w3 w4
    exit program.
```

C Exemples

C1 S/P internes,externes + données globales,locales externes

```
IDENTIFICATION DIVISION.
PROGRAM-ID. tp1.
WORKING-STORAGE SECTION.

1 w1 pic 99     value 12 is global.
1 w2 pic 99v99           is external.
1 w3 pic 9(4)            is global external.
1 w4 pic 99v99           is global.
```

résultats obtenus

	W1	W2	W3	W4
après init =>	12	1552	0789	2855
après mult2 =>	24	3104	1578	2855
après plus5 =>	24	3604	1578	2855
après plus8 =>	24	4404	1586	2855
après plus1	24	4504	1587	2855
apr moins3	21	4204	1584	2855
après div10	02	0420	0158	0285

```
PROCEDURE DIVISION.
debut.
     move 12    to w1
     move 15.52 to w2
     move 789   to w3
     move 28.55 to w4
     display "              w1  w2    w3   w4"
     display " après init.=> " w1 "  " w2 "  " w3 " " w4
     call "mult2" ❶
     display "après mult2=> " w1 "  " w2 "  " w3 " " w4

     call "plus5" ❷
     display "après plus5=> " w1 "  " w2 "  " w3 " " w4

     call "plus8" ❸
     display "après plus8=> " w1 "  " w2 "  " w3 " " w4

     call "plus1" ❹
     display "après plus1=> " w1 "  " w2 "  " w3 " " w4

     call "moins3" ❺
     display "apr  moins3=> " w1 "  " w2 "  " w3 " " w4

     call "div10" ❻
     display "après div10=> " w1 "  " w2 "  " w3 " " w4

   stop run.
IDENTIFICATION DIVISION.
PROGRAM-ID. mult2 initial.
WORKING-STORAGE SECTION.

1 w2 pic 99v99 external.
1 w4 pic 99v99 global.
PROCEDURE DIVISION.
debut.
    multiply 2 by w1 w2 w3 w4
    exit program.
    end program mult2.
IDENTIFICATION DIVISION.
PROGRAM-ID. moins3 initial.
WORKING-STORAGE SECTION.
1 w2 pic 99v99 external.
1 w3 pic 9999  external global.
1 w4 pic 99v99 global.
PROCEDURE DIVISION.
debut.
    subtract 3 from w1 w2 w3 w4
    exit program.
    end program moins3.
IDENTIFICATION DIVISION.
PROGRAM-ID. div10 initial.
WORKING-STORAGE SECTION.
1 w2 pic 99v99 external.
1 w3 pic 9999  external .
PROCEDURE DIVISION.
debut.
    divide 10 into w1 w2 w3 w4
    exit program.
    end program div10.
    end program tp1.
```

les sous -programmes internes

C2 le partage des fichiers : cas avec des variables globales et et des variables externes

C21 résultat souhaité

A partir de 2 fichiers décrits ci-dessous , on veut éditer toutes les personnes dont le nom commence par la lettre «P» ou «p».

NB :On aurait pu faire une application très simple, mais , le but étant de montrer l'utilisation des variables globales et externes , on a volontairement multiplié l'utilisation de sous-programmes totalement inutiles.

JEU d'ESSAI : c:\gestion\dir02.txt

					NOM	PRENOM					
n°	1	02	DIR	KKKK133	BISTOUKET	PIERRE	0000000000	590208	OS1	1204	%%%%%
n°	2	02	DIR	KKKK135	CASANOVIEN	ALICE	0000000000	550208	IPE	1250	%%%%%
n°	3	02	DIR	KKKK059	CHIASSOU	ZOE	0000000000	690208	IPE	1500	%%%%%
n°	4	02	DIR	KKKK053	DE LA CARTOUCHE	JACQUES	0000000000	500208	IPE	1345	%%%%%
n°	5	02	DIR	KKKK061	EBOUNARD	BERTRAND	0000000000	510208	IPE	1200	%%%%%
n°	6	02	DIR	KKKK055	FITROCHON	GISELE	0000000000	560208	IPE	1641	%%%%%
n°	7	02	DIR	KKKK137	JACQUOUX	JACQUES	0000000000	550208	OS1	1552	%%%%%
n°	8	02	DIR	KKKK131	LE CHAUVE	IRENE	0000000000	580208	IPE	1202	%%%%%
n°	9	02	DIR	KKKK057	MOMEAU	FERNAND	0000000000	690208	IPE	1160	%%%%%
n°	10	02	DIR	KKKK051	PASTISSE	ALPHONSE	0000000000	410208	IPE	1800	%%%%%
n°	11	02	DIR	KKKK063	PISSOUNET	JACQUES	0000000000	530208	OS1	1550	%%%%%
n°	12	02	DIR	KKKK139	SOMMIER-DUVALLON	MARCELLIN	0000000000	520208	OS1	1200	%%%%%
n°	13	02	DIR	KKKK017	ZYVANISEVITCHSKY	JULES	0000000000	420208	IPE	1550	%%%%%

JEU d'ESSAI : c:\gestion\etu02.txt

n°	1	02	ETU	KKKK054	BOBARD	JEAN	0000000000	560914	TEF	1225	%%%%%
n°	2	02	ETU	KKKK132	BOVEAU	JEAN	0000000000	590914	OS1	1622	%%%%%
n°	3	02	ETU	KKKK028	CARMICON	JEAN	0000000000	420914	TEF	1725	%%%%%
n°	4	02	ETU	KKKK030	CASBONBINI	MARCEL	0000000000	620914	TEF	1236	%%%%%
n°	5	02	ETU	KKKK050	CHIALOUX	FLORENCE	0000000000	410914	TEF	1725	%%%%%
n°	6	02	ETU	KKKK060	CHIASSOUX-BERNARD	ANTHONY	0000000000	520914	OS1	1825	%%%%%
n°	7	02	ETU	KKKK140	DESSALON	OLIVIA	0000000000	510914	IEF	1625	%%%%%
n°	8	02	ETU	KKKK026	DUFER	CASIMIR	0000000000	420914	IEF	1550	%%%%%
n°	9	02	ETU	KKKK056	FITROCHON	BENOIT	0000000000	560914	OS1	1642	%%%%%
n°	10	02	ETU	KKKK005	FOUMI	PIERRE	0000000000	400813	OS1	1700	%%%%%
n°	11	02	ETU	KKKK034	LABAF	JEAN	0000000000	700914	TEF	1625	%%%%%
n°	12	02	ETU	KKKK138	LABROUILLE	ISIDORE	0000000000	550914	IPE	1625	%%%%%
n°	13	02	ETU	KKKK130	LE CHAUVE	STANISLAS	0000000000	580914	OS1	1622	%%%%%
n°	14	02	ETU	KKKK032	MACARONETTINI	LEON	0000000000	690914	TEF	1350	%%%%%
n°	15	02	ETU	KKKK136	MULLER	AGATHE	0000000000	550914	IEF	1525	%%%%%
n°	16	02	ETU	KKKK058	MURINI	ANGE	0000000000	690914	OS1	1105	%%%%%
n°	17	02	ETU	KKKK018	N'GORE	JEAN	0000000000	560914	TEF	1650	%%%%%
n°	18	02	ETU	KKKK002	PIOLA	JEAN	0000000000	420914	TEF	1625	%%%%%
n°	19	02	ETU	KKKK062	PISSOUNET	MARCEL	0000000000	540914	OS1	1925	%%%%%
n°	20	02	ETU	KKKK016	RASPITOU	JEAN	0000000000	450914	TEF	1425	%%%%%
n°	21	02	ETU	KKKK052	ROUPIGNAC	THIERRY	0000000000	500914	TEF	1425	%%%%%
n°	22	02	ETU	KKKK134	ROYER-MOLLOUX	VERONIQUE	0000000000	550914	IEF	1624	%%%%%

C22 programme COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. tp3.
ENVIRONMENT DIVISION.
input-output section.
file-control.
     select fetu02 assign to "c:\gestion\etu02.txt".
     select fdir02 assign to "c:\gestion\dir02.txt".
     select fvidageassign to "c:\gestion\vidage.txt".
data division.
file section.
fd fvidage is global external linage is 66.
1 sligne pic x(120).
fd fdir02 is external.
1 enreg-dir2  pic x(80).
fd fetu02 is external.
1 enreg-etu2 global pic x(80).
WORKING-STORAGE SECTION.
77 wtrouve pic 9 value 0.
   88 trouve      value 1.
77 wlettre pic x value "P".
1 i     global external pic 999.
1 wctf1 external     pic 9.
        88 ff-dir     value 1.
1 wctf2 external     pic 9.
   88 ff-etu     value 1.
```

fichier utilisé dans le S/P interne «ecriture» sous forme globale

et dans le S/P externe «rechdir» => déclaration en «externe»

ces 2 fichiers sont utilisés dans des S/P externes => déclaration en EXTERNAL et l'article de etu02.txt est directement imprimé dans le sous-programme «écriture» => déclaration en global.

PARTAGE DE FICHIERS ENTRE PLUSIEURS PROGRAMMES

```
procedure division.
debut.
    move 0 to wctf1 wctf2 i
    open input fetu02 fdir02 output fvidage
    read fetu02 at end
         set ff-etu to true
    end-read
    perform until ff-etu
      call   "nomrech" using wtrouve wlettre
      cancel "nomrech"
      if trouve
        then
           display enreg-etu2
           call "ecriture"
           read fetu02 at end
                set ff-etu to true
           end-read
      end-if
    end-perform
    read fdir02
      at end
        display "fichier etu02.txt = vide"
      not at end
        call   "rechdir"
        cancel "rechdir"
    end-read
    close fdir02 fetu02 fvidage
    stop run.
```

La 1 ère lecture de Fetu02.txt a lieu dans ce module

Les autres sont effectuées dans «nomrech» tant qu'on ne trouve pas de personne avec un nom commençant par «P» ou «p» :

quand c'est le cas, on traite (appel S/P d'impression) et on relit à nouveau l'article suivant dans cette séquence.

quand le premier fichier est fini , on accède au 2 ème lequel sera gèré par le S/P externe «rechdir» et son sous-programme inclus «rechnom» décrits à la page suivante.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ecriture.
DATA DIVISION.
1 wligne .
  3        pic x(3).
  3 wnum    pic zzz.
  3        pic x(6).
  3 w108    pic x(108).
procedure division.
debut.
    add 1 to i   move i to wnum
    move enreg-etu2(1:2) to w108(1:5)
    move enreg-etu2(3:3) to w108(6:6)
    move enreg-etu2(6:7) to w108(12:10)
    move enreg-etu2(13:20) to w108(22:23)
    move enreg-etu2(33:20) to w108(45:23)
    move enreg-etu2(53:10) to w108(68:13)
    move enreg-etu2(63:6) to w108(81:9)
    move enreg-etu2(69:3) to w108(90:6)
    move enreg-etu2(72:4) to w108(96:7)
    move enreg-etu2(76:5) to w108(103:5)
    write sligne from wligne
```

Ce sous-programme interne utilise :

le fichier Fvidage (déclaré en global)

l'article du fichier Fetu02.txt également global

et la variable globale i (qui est également incrémentée dans le module «rechdir» et donc définie en external)

=> il n'y a aucune description à écrire concernant ces 2 entités.

=>

les données globales ne sont pas décrites dans le module interne appelé

```
    exit program.
    end program ecriture.
end program tp3.
IDENTIFICATION DIVISION.
PROGRAM-ID. rechdir.
ENVIRONMENT DIVISION.
input-output section.
file-control.
     select fdir02  assign to "c:\gestion\dir02.txt".
     select fvidage assign to "c:\gestion\vidage.txt".
data division.
file section.
fd fdir02 is  global external.
1 enreg-dir2  global pic x(80).
fd fvidage is external linage is 66.
1 sligne pic x(120).
WORKING-STORAGE SECTION.
77 wlettre pic x value "P".
77 wtrouve pic 9 value 0.
   88 trouve    value 1.
77 wctf1       global external pic 9.
   88 ff-dir value 1.
77 i    is  external pic 999.
1 wligne.
   3        pic x(3).
   3 wnum    pic zzz.
   3        pic x(6).
   3 w108    pic x(108).
procedure division.
debut.
    perform until ff-dir
    |  call "rechnom" using wtrouve wlettre
    |  cancel "rechnom"
    |  if trouve
    |  |  then
    |  |     display enreg-dir2
    |  |     add 1 to i
    |  |     move i to wnum
    |  |     move enreg-dir2(1:2)      to w108(1:5)
    |  |     move enreg-dir2(3:3)      to w108(6:6)
    |  |     move enreg-dir2(6:7)      to w108(12:10)
    |  |     move enreg-dir2(13:20) to w108(22:23)
    |  |     move enreg-dir2(33:20) to w108(45:23)
    |  |     move enreg-dir2(53:10) to w108(68:13)
    |  |     move enreg-dir2(63:6)     to w108(81:9)
    |  |     move enreg-dir2(69:3)     to w108(90:6)
    |  |     move enreg-dir2(72:4)     to w108(96:7)
    |  |     move enreg-dir2(76:5)     to w108(103:5)
    |  |     write sligne from wligne
    |  |     read fdir02 at end
    |  |     |     set ff-dir to true
    |  |     end-read
    |  end-if
    end-perform
    exit program.
IDENTIFICATION DIVISION.
PROGRAM-ID. rechnom.
WORKING-STORAGE SECTION.
77 wmaj pic x.
77 wmin pic x.
linkage section.
77 ltrouve pic 9.
   88 lok value 1.
77 llettre pic x.
procedure division using ltrouve llettre.
debut.
    move 0 to ltrouve
    move llettre to wmaj wmin
    call "cbl_toupper" using wmaj by value 1
    call "cbl_tolower" using wmin by value 1
    perform until ff-dir or lok
    |  if enreg-dir2(13:1) = wmaj or wmin
    |  |  then
    |  |     set lok to true
    |  |  else
    |  |     read fdir02 at end
    |  |     |     set ff-dir to true
    |  |     end-read
    |
```

```
            end-if
         end-perform
         exit program.
      end program rechnom.
      end program rechdir.
    $set NESTCALL
      IDENTIFICATION DIVISION.
      PROGRAM-ID. nomrech.
      ENVIRONMENT DIVISION.
      configuration section.
      input-output section.
      file-control.
          select fetu02
                 assign to "c:\gestion\etu02.txt".
      data division.
      file section.
      fd fetu02 is external.
      1 enreg-etu2  pic x(80).
      WORKING-STORAGE SECTION.
      77 wtrouve pic 9 value 0.
        88 trouve    value 1.
      77 wctf2          external pic 9.
        88 ff-etu value 1.
      77 wmaj pic x.
      77 wmin pic x.
      linkage section.
      77 ltrouve pic 9.
        88 lok value 1.
      77 llettre pic x.
      procedure division using ltrouve llettre.
      debut.
         move 0 to ltrouve
         move llettre to wmaj wmin
         call "cbl_toupper" using wmaj by value 1
         call "cbl_tolower" using wmin by value 1
         perform until ff-etu or lok
         |  if enreg-etu2(13:1) = wmaj or wmin
         |  | then
         |  |     set lok to true
         |  |  else
         |  |     read fetu02
         |  |        at end
         |  |           set ff-etu to true
         |  |     end-read
         |  end-if
         end-perform
      exit program.
```

C23 résultat obtenu

```
1     02   ETU   KKKK002   PIOLA              JEAN              0000000000   420914   TEF   1625   %%%%%
2     02   ETU   KKKK062   PISSOUNET          MARCEL            0000000000   540914   OS1   1925   %%%%%
3     02   DIR   KKKK051   PASTISSE           ALPHONSE          0000000000   410208   IPE   1800   %%%%%
4     02   DIR   KKKK063   PISSOUNET          JACQUES           0000000000   530208   OS1   1550   %%%%%
```

C3 Exemple complet comprenant les cas suivants :

utilisation de sous-programme internes et externes
utilisation de variables globales et locales
utilisation de variables externes
passage d'information par adresse
passage en paramètre de l'adresse réelle d'un programme
appel d'un sous-programme par son adresse
utilisation du code de retour

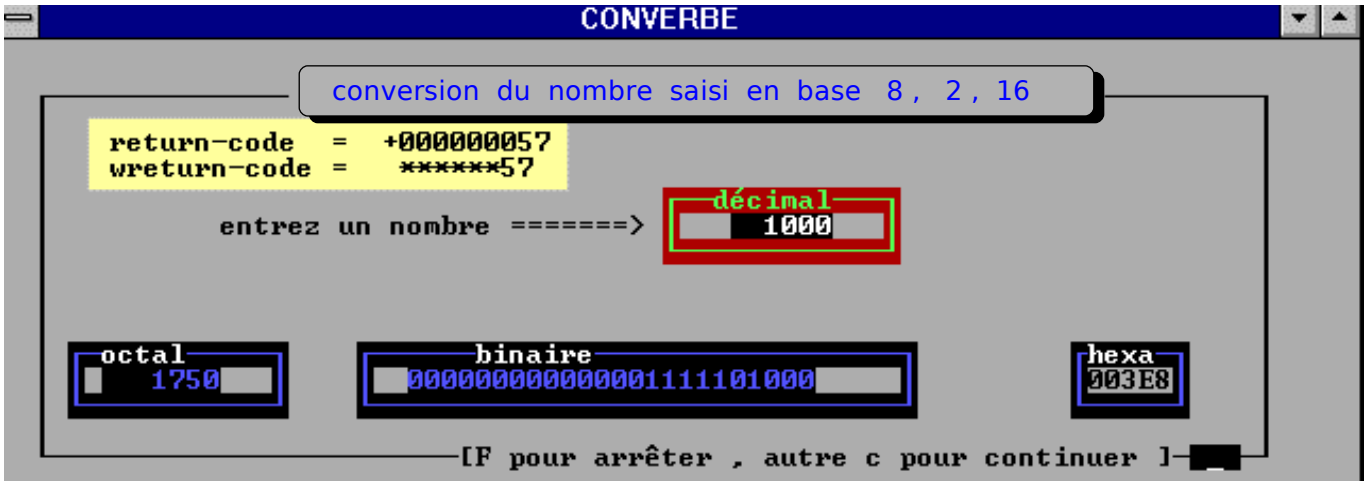
C31 résultat souhaité

Le masque d'écran ci-dessous nous décrit le résultat à obtenir : à partir d'un nombre décimal [9(6) maxi] saisi par l'utilisateur , on veut afficher sur l'écran les représentations de ce nombre en binaire, en hexadécimal et en octal.

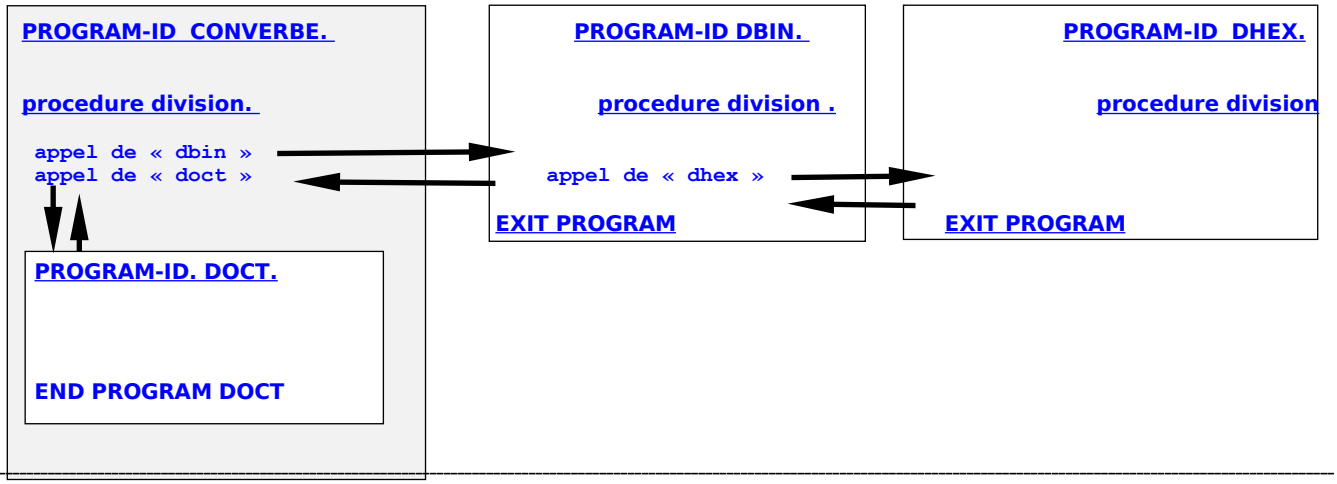
Les solutions pour résoudre ce problème sont nombreuses mais , on a imposé la réalisation de 3 sous-programmes :

« doct » qui convertit le décimal en octal
« dbin » qui convertit le décimal en binaire
« dhex » qui convertit le décimal en hexadécimal

Cet exercice étant généralement étudié en début de cours ,les élèves ne connaissent pas encore les tables, les clauses occurs , redefines , ... : on a donc choisi , pour les algorithmes des sous-programmes , de ne travailler qu'avec les verbes arithmétiques.



C32 enchaînement des sous-programmes



END PROGRAM CONVERBE.

C33 programmes cobol

```
$set RTNCODE-SIZE"4"
 IDENTIFICATION DIVISION.
 PROGRAM-ID. converbe.
 AUTHOR. IC LAFARGUE.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 1 wad-dhex usage is procedure-pointer.
 1 wdecimal is global external pic 9(6).
 1 woctal   is global pic 9(7) value 0.
 1 wbina value zero.
   3            pic x(12).
   3 wbinaire pic x(24).
 1 whexa          external pic xxxxx.
 1 wreponse pic x value " ".
 1 wreturn-code pic -*(7)9.
 SCREEN SECTION.
   COPY "c:\laf\cobol\ex1\conver.ss".
 PROCEDURE DIVISION.
 10.
     set wad-dhex to entry "dhex"
     perform until wreponse = "F" or "f"
        move space to wreponse
        display magrille
        accept  gdecimal
        call "dbin" using wbina wad-dhex ❶
*       cancel "dbin"
        move return-code to wreturn-code
        display "return-code  = "   at line 6 col 10
                 return-code        at line 6 col 26
        display "wreturn-code = "   at line 7 col 10
                 wreturn-code       at line 7 col 26
        if return-code = 57 ❸
        | then
        |    call "doct" ❷
        |    cancel "doct"
        |
        end-if
        display goctal
        display gbinaire
        display ghexa
        accept greponse
     end-perform
     stop run.
```

```
01 magrille.
  02 BACKGROUND-COLOR 7 FOREGROUND-COLOR 0.
   03 BLANK SCREEN.
   03 LINE 6 COL 3 VALUE "+-------------- ".
   03 COL 19 BACKGROUND-COLOR 3 VALUE "conversion du nombre saisi
-" en base 8 , 2 , 16".
          .         .         .         .
          .         .         .         .
   03 COL 71 BACKGROUND-COLOR 0 FOREGROUND-COLOR 7 HIGHLIGHT
       VALUE " ".
   03 COL 73 BACKGROUND-COLOR 0 FOREGROUND-COLOR 7 HIGHLIGHT  VALUE
" ".
   03 COL 74 VALUE "-+".
   03 gdecimal LINE 11 COL 44 BACKGROUND-COLOR 0 FOREGROUND-
          COLOR 7 HIGHLIGHT PIC Z(5)9 TO wdecimal PROMPT ".".
   03 goctal LINE 17 COL 7 BACKGROUND-COLOR 0 FOREGROUND-COLOR
             1  HIGHLIGHT PIC Z(6)9 FROM woctal.
   03 gbinaire LINE 17 COL 25 BACKGROUND-COLOR 0 FOREGROUND-
               COLOR   1 HIGHLIGHT PIC X(24) FROM wbinaire.
   03 ghexa LINE 17 COL 65 PIC X(5) FROM whexa.
   03 greponse LINE 20 COL 72 BACKGROUND-COLOR 0 FOREGROUND-

               COLOR  7 HIGHLIGHT PIC X USING wreponse.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. doct initial.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 i          pic 9     value 0.
 77 wdividende pic 9(6) value 0.
 77 wdiviseur  pic 9(6) value 8.
 77 wquotient  pic 9(6) value 0.
 77 wreste     pic 9(7) value 0.
 PROCEDURE DIVISION.
 10.
     move 0 to woctal
     move wdecimal to wquotient
     perform until wquotient <= 0
        move wquotient to wdividende
        divide wdividende by wdiviseur
           giving wquotient remainder wreste
        compute woctal = woctal+ wreste * (10 ** I)
        add 1 to i
     end-perform
     exit program.
 end program doct.
 end program converbe.
```

❶
on appelle « dbin » en lui passant l'adresse du S/P « dhex » qu'on a mis dans wad-dhex avec SET.
« dbin » se servira de cette adresse pour appeler « dhex »

❷
appel de « doct » (interne) sans passer de paramètres qui sont globaux.

❸
on vérifie le code de retour renvoyé par « dbin » au moyen de la commande exit program returning.. : comme le call de « dbin » ne contenait pas de returning , l'information retournée se met dans le RETURN-CODE.
(on a mis la valeur 9 dans le code retour mais celui-ci contient 57 qui est en fait le code ASCII de « 9 » en décimal)

```
*$set RTNCODE-SIZE"4"
 IDENTIFICATION DIVISION.
 PROGRAM-ID. dbin initial.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 i         pic 9(6)  value 0.
 77 wdividende pic 9(6) value 0.
 77 wdiviseur  pic 9(6) value 2.
 77 wquotient  pic 9(6) value 0.
 77 wreste     pic 9    value 0.
 1 wresultat.
  3 wresu     pic 9(18) value 0.
 1 wdecimal    pic 9(6) external.
 1 lad-i usage is pointer.
 1 werreur     pic 9 value 0.
 linkage section.
 1 lbina.
   3         pic x(18).
   3 lresu pic 9(18).
 1 ladresse-dhex usage is procedure-pointer.
 PROCEDURE DIVISION using lbina ladresse-dhex.
 10.
     move zero         to lbina
     move wdecimal to wquotient
     perform until wquotient <= 0
     |  move wquotient to wdividende
     |  divide wdividende by wdiviseur
     |        giving wquotient remainder wreste
     |  compute lresu = lresu + wreste * (10 ** I)
     |  add 1 to i
     end-perform
     call ladresse-dhex returning into lad-i
     move 9 to werreur
     exit program returning werreur.



 IDENTIFICATION DIVISION.
 PROGRAM-ID. dhex initial.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 i            pic 9       value 5.
 77 wdividende pic 9(6)       value 0.
 77 wdiviseur  pic 9(6)       value 16.
 77 wquotient  pic 9(6)       value 0.
 77 wreste     pic 99       value 0.
 1 wdecimal external pic 9(6).
 1 whexa         external pic x(5).
 PROCEDURE DIVISION.
 10.
     move zero         to whexa
     move wdecimal to wquotient
     perform until wquotient <= 0
     |  move wquotient to wdividende
     |  divide wdividende by wdiviseur
     |        giving wquotient remainder wreste
     |  evaluate wreste
     |  | when 0 thru 9
     |  |    move wreste(2:1) to whexa(i:1)
     |  | when 10
     |  |    move "A" to whexa(i:1)
     |  | when 11
     |  |    move "B" to whexa(i:1)
     |  | when 12
     |  |    move "C" to whexa(i:1)
     |  | when 13
     |  |    move "D" to whexa(i:1)
     |  | when 14
     |  |    move "E" to whexa(i:1)
     |  | when 15
     |  |    move "F" to whexa(i:1)
     |  end-evaluate
     |  subtract 1 from i
```

```
end-perform
exit program returning  address of i.
```