CHAPITRE 1 : INSTRUCTIONS ET TYPES ELEMENTAIRES

1 Introduction[1]

COBOL est un langage de programmation de troisième génération créé en 1959. Son nom est un acronyme (**COmmon Business Oriented Language**) qui révèle sa vocation originelle : être un langage commun pour la programmation d'applications de gestion. Le standard COBOL 2003 supporte en particulier la programmation orientée objet et d'autres traits des langages de programmation modernes.

Le langage COBOL était de loin le langage le plus employé dans les années 60 à 80, et reste toujours en utilisation dans des entreprises majeures, notamment dans les institutions financières qui disposent d'une vaste bibliothèque d'applications COBOL.

En 2005, le Gartner Group estimait que 75% des données du monde des affaires étaient traitées par des programmes en COBOL et que 15% des nouveaux programmes développés le seront dans ce langage.

2 Structure générale d'un programme Cobol

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.

Les entêtes sont optionnelles mais seront de plus en plus utiles pour la lisibilité du programme par l'utilisateur et finalement par le compilateur.

L'environnement de travail que nous utiliserons est Net Express de MicroFocus.
Documentation en ligne sur : http://supportline.microfocus.com/documentation/

```
program-id. nom-du-programme.          *>partie en-tête du programme
- - - - - - - -
[data division.]                       *>partie des déclarations
[working-storage section.]             *>variables locales
- - - - - - -
screen section.                        *>plages et champs d'écran
- - - - - - -
[procedure division.]                  *>partie des instructions
- - - - - - - -
- - - - - - - -
end program nom-du-programme.   *>fin du programme
```

3 Exemple détaillé n°1

3.1 Le programme

Enoncé du problème: Une automobile a parcouru une certaine distance en un certain temps et a consommé une certaine quantité de litres d'essence.
On recherche sa vitesse moyenne et sa consommation au kilomètre pour ce trajet.

Solution :
Les données :

- distance en km : **nbkm**
- durée en heures et centièmes : **nbhc**
- quantité d'essence consommée en litres et centièmes : **nblc**

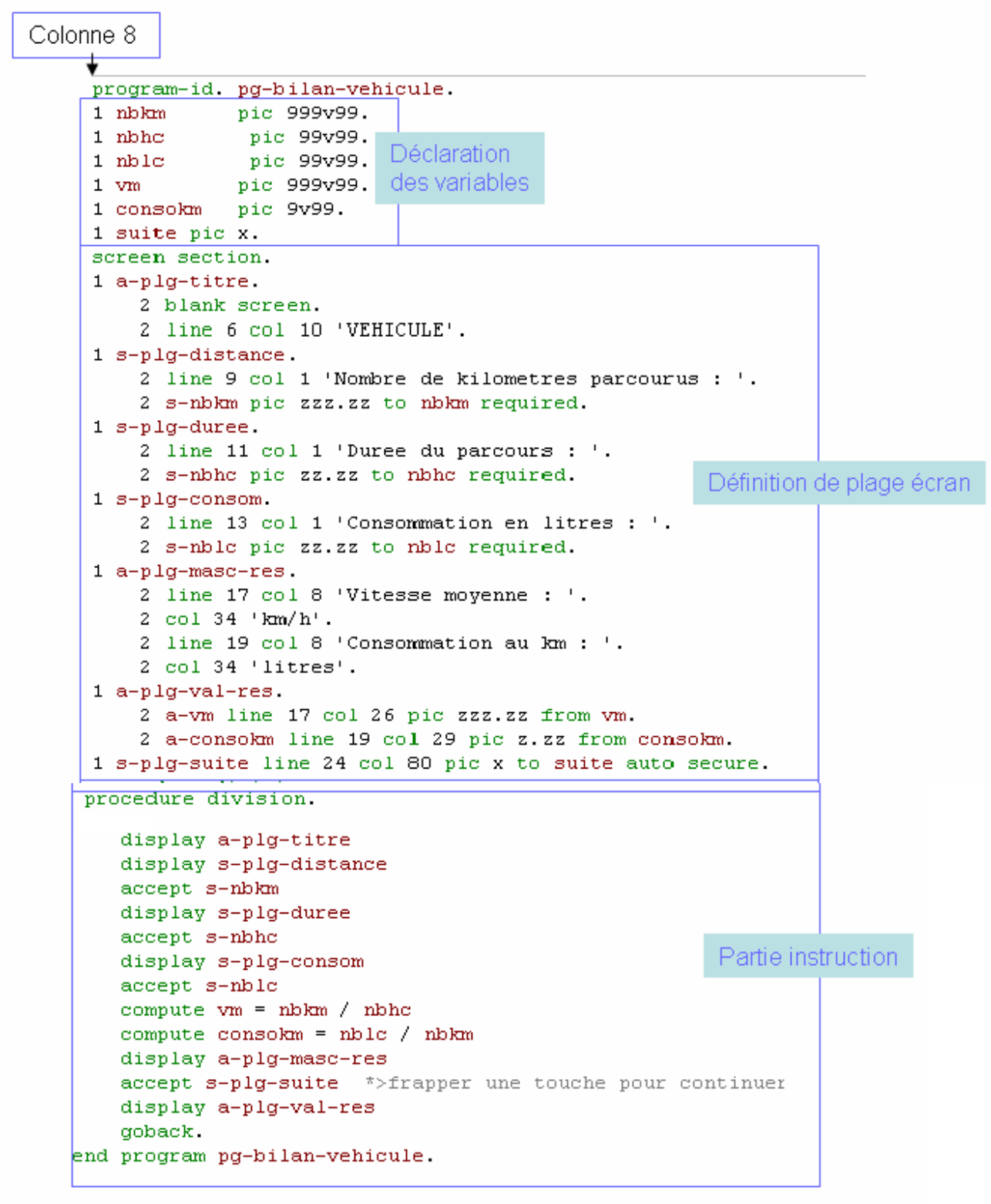
[1] Informations issues de Wikipédia fr.

Les résultats recherchés :

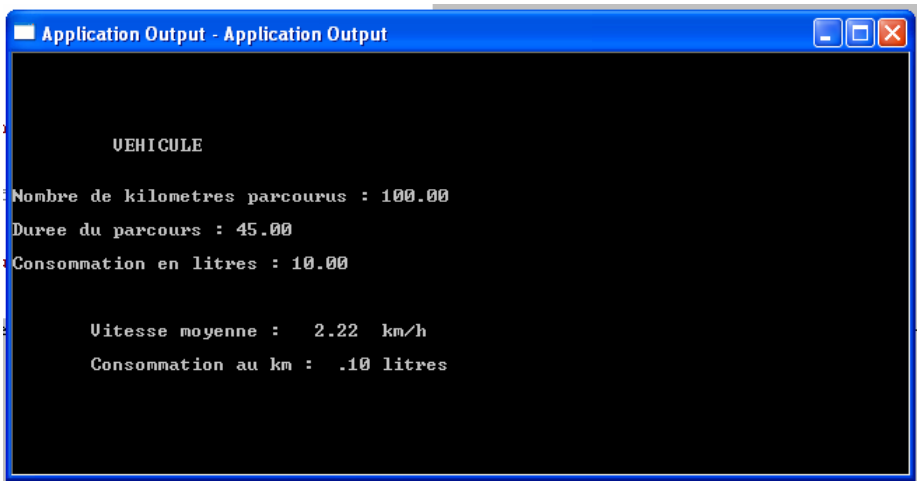
- vitesse moyenne en km et centième par heure : **vm**
- consommation au km en litres et centièmes : **consokm**

La démarche :

Lire les données : **nbkm, nbhc, nblc**
Calculer vm **: vm=nbkm/nbhc**
Calculer consokm : **consokm= nblc/nbkm**
Afficher les résultats **: vm, consokm**




```
program-id. pg-bilan-vehicule.
1 nbkm      pic 999v99.
1 nbhc       pic 99v99.
1 nblc       pic 99v99.
1 vm        pic 999v99.
1 consokm   pic 9v99.
1 suite pic x.
screen section.
1 a-plg-titre.
    2 blank screen.
    2 line 6 col 10 'VEHICULE'.
1 s-plg-distance.
    2 line 9 col 1 'Nombre de kilometres parcourus : '.
    2 s-nbkm pic zzz.zz to nbkm required.
1 s-plg-duree.
    2 line 11 col 1 'Duree du parcours : '.
    2 s-nbhc pic zz.zz to nbhc required.
1 s-plg-consom.
    2 line 13 col 1 'Consommation en litres : '.
    2 s-nblc pic zz.zz to nblc required.
1 a-plg-masc-res.
    2 line 17 col 8 'Vitesse moyenne : '.
    2 col 34 'km/h'.
    2 line 19 col 8 'Consommation au km : '.
    2 col 34 'litres'.
1 a-plg-val-res.
    2 a-vm line 17 col 26 pic zzz.zz from vm.
    2 a-consokm line 19 col 29 pic z.zz from consokm.
1 s-plg-suite line 24 col 80 pic x to suite auto secure.
procedure division.
   display a-plg-titre
   display s-plg-distance
   accept s-nbkm
   display s-plg-duree
   accept s-nbhc
   display s-plg-consom
   accept s-nblc
   compute vm = nbkm / nbhc
   compute consokm = nblc / nbkm
   display a-plg-masc-res
   accept s-plg-suite  *>frapper une touche pour continuer
   display a-plg-val-res
   goback.
end program pg-bilan-vehicule.
```



3.2 Les commentaires du programme

Ce programme comporte 3 parties :

- Une partie contenant la déclaration des variables. Il s'agit ici de 5 numériques :

pic 999v99 correspond à un numérique avec 3 chiffres avant le séparateur décimal et 2 après.

- Une partie « screen section » : dans laquelle on décrit précisément où et comment seront saisies/affichées les données.

1 a-plg-titre.
2 blank screen.
2 line 6 col 10 'VEHICULE'.

Indique qu'une plage nommée a-plg-titre est créée et comporte 2 éléments :

- Blank screen : permet un effacement de l'écran
- A la line 6 et colonne 10, la chaîne de caractère **'VEHICULE'** devra être affichée.

1 s-plg-consom.
2 line 13 col 1 'Consommation en litres : '.
2 s-nblc pic zz.zz to nblc required.

Indique que la plage **s-plg-consom** est créée et comporte 2 éléments :

- A la line 13 colonne 1 la chaîne **'Consommation en litres : '**est affichée.
- Le champ de saisie **s-nblc** servira à recueillir le contenu de la variable **nblc**. (required signifie que la saisie est obligatoire).

1 a-plg-val-res.
2 line 17 col 26 pic zzz.zz from vm.
2 a-consokm line 19 col 29 pic z.zz from consokm.

Indique que la plage **a-plg-val-res** est créée et comporte 2 éléments :

- Le champ **a-vm** à la line 17 colonne 26 ayant un format 3 chiffres, le séparateur décimal puis 2 chiffres. L'utilisation du 'z' au lieu de '9' permet la suppression des zéros non significatifs. (ex. le nombre 078.0678 sera affiché sous la forme 78.06). La valeur affichée sera le contenu de la variable vm.
- Le champ **a-consokm** sera affiché à la ligne 19 colonne 29 sous la forme de 2 chiffres avant le séparateur décimal suivi de 2 chiffres après. La valeur affichée sera le contenu de la variable **consokm.**

1 s-plg-suite line 24 col 80 pic x to suite auto secure.

Indique qu'un champ nommé **s-plg-suite** correspondant la position line 24 colonne 80 est créé et recueillera le contenu de la variable suite sous la forme d'un caractère. **Auto secure** signifie que la saisie n'apparaîtra pas à l'écran.

***debut** est une ligne de commentaire (* en colonne 7)

on peut aussi écrire ***> début** n'importe ou sur la ligne

display a-plg-titre
permet d'afficher la plage telle qu'elle a été définie en screen section

display s-plg-distance
accept s-nbkm
permet d'afficher la plage **s-plg-distance** telle qu'elle a été définie dans la screen section puis provoque la saisie du champ **s-nbkm**.

compute vm = nbkm / nbhc
instruction permettantde réaliser le calcul de l'expression **nbkm / nbhc** et d'affecter le résultat à la variable **vm**.

goback
instruction permet de retourner au programme appelant (ici le programme principal)

En résumé, pour écrire le programme en cobol, il faut :

Déclarer toutes les variables nbkm, nbh, nbl , vm, consokm en définissant leur type
Définir les plages d'écran pour la saisie et l'affichage des variables
Lire les données : nbkm, nbh, nbl en utilisant les plages de saisies
Calculer vm : vm=nbkm/nbh
Calculer consokm : consokm= nbl/nbh
Afficher les résultats : vm, consokm en utilisant sur les plages d'affichage

4 Partie des déclarations

Est une suite de lignes de déclarations.

4.1 Section des constantes et des variables (working-storage section)

4.1.1 Déclaration d'une constante

78 ***id-const*** **value valeur.**

Le '78' est situé en colonne 8 et 9.

4.1.2 Déclaration d'une variable

1 ***id-var*** { **[pic descr-type]** / **id-type** } **[value valeur] .**

dans ce cas la mémorisation est implicite : un caractère par octet (**display**)
id-type est le nom d'un type déclaré en amont.
'Valeur' est la valeur initiale.

Le **descripteur** de type (descr-type) est une combinaison permise de symboles, pour décrire :

a) des types numériques purs

'9' représente un chiffre décimal de 0 à 9 : mémorisé dans un octet
'V' ou 'v' symbolise une marque décimale virtuelle pour les calculs : ne prend pas de place mémoire supplémentaire
'S' ou 's' représente un signe pour les calculs : pas d'occupation mémoire supplémentaire

Exemple :
1 nbkm pic 999v99.
1 num pic9(3)v9(2).

b) des types alphabétiques

'A' ou 'a' représente une lettre ou un espace : mémorisé dans un octet

c) des types alphanumériques

'X' ou 'x' symbolise un caractère quelconque du jeu de caractères : occupe un octet

Remarque
Chaque symbole peut être suivi d'un facteur de répétition entre parenthèses.

4.1.3 Déclaration d'un type simple

1 id-type pic descr-type typedef .

4.1.4 Déclaration d'un booléen

En Cobol un booléen est toujours attaché à une variable. La valeur du booléen est déterminée par le fait, pour la variable associée, de prendre l'une des valeurs citées dans la déclaration du booléen.

nb-niv [*id-var-bool*] clause pic [clause value] .
88 *id-bool* value liste [] de { const | constj thru constk } [false const] .

Exemple :
1 resultat pic x.
88 juste value '1' false '0'.

La variable résultat est un caractère qui prend la valeur '0' ou '1'
Le booléen juste est attaché à la variable résultat et prend la valeur vrai si résultat contient la valeur '1' et faux sinon.

Exemple :
1 note_finale pic 99v99.
88 passable value 10 thru 11.99.
88 assez_bien value 12 thru 13.99.
88 bien value 14 thru 15.99.

Le booléen **passable** prend la valeur vrai dès que la variable **note_finale** prend une valeur dans l'intervalle 10 à 11,99. Dans tous les autres cas, **passable** prend la valeur faux.
Le booléen **bien** prend la valeur vrai dès que la variable **note_finale** prend une valeur dans l'intervalle 14 à 15,99. Dans tous les autres cas, **bien** prend la valeur faux.

4.2 Section des plages et champs d'écran : screen section

Est une suite de déclarations de plages et de champs d'écran.

4.2.1 Déclaration d'un champ d'écran constant

nb-niv [*id-chp-ecr*] [line nl] [col nc] const-ch.

avec nb-niv dans [1, 49]

4.2.2 Déclaration d'un champ d'écran variable

nb-niv [*id-chp-ecr*] [line nl] [col nc] [clause pic étendue] [clauses d'écran].

a) formats des clauses **pic** étendues

- Pour un champ d'écran de saisie :

pic descr-ed-type to *id-var*

- Pour un champ d'écran d'affichage :

pic descr-ed-type from *id-var*

- Pour un champ d'écran mixte :

pic descr-ed-type using *id-var*

Un **descripteur** de type édité (descr-ed-type) contient des symboles d'édition pour décrire :

a) des types numériques édités

'.'	pour l'insertion d'un point ('**.**') comme marque décimale réelle
'+' (ou '-')	placé une fois en tête du schéma, implique l'insertion du caractère '**+**' (ou ' **'**) si la valeur est positive sinon du caractère '**-**'
'**Z**' (ou '*****')	pour se substituer, en une suite continue, aux premiers symboles'**9**' à gauche du schéma et provoquer, dans les positions correspondantes, le remplacement d'un éventuel zéro non significatif de gauche par un caractère espace (' ') (ou '*****')

Exemples :

schémas de types	valeur contenue	valeur affichée
99.9	**12,3**	**12.3**
+99.9	**12,3**	**+12.3**
-99.9	**12,3**	**_12.3**
ZZ9.9	**12,3**	**_12.3**
***(4).9**	**12,3**	****12.3**

Remarque :

'+' (ou '-')	en une suite d'au moins deux occurrences joue le même rôle avec décalage du signe dans la position du zéro non significatif impliqué le plus à droite
'**,**' (ou '**/**')	pour l'insertion du caractère '**,**' (ou '**/**') dans chaque position du symbole correspondant au sein du schéma

b) des types alphanumériques édités

'B' (ou '**/**')	pour l'insertion d'un espace (ou d'un '**/**') dans la position correspondante

c) quelques clauses d'écran

auto, **background-color**, **foreground-color**, **blank-screen**, **blank-line**, **erase eos**, **erase eol**, **highlight**, **lowlight**, **prompt**, **required**, **secure**, **reverse-video**, **underline**, **. . . .**

Remarque :

Déclarer un nombre signé : s999.99

Afficher un nombre signé : +999.99 le signe n'apparaît dans tous les cas

ou bien -999.99 le signe n'apparaît que lorsque le nombre est négatif.

5 La partie des instructions

Correspond à la **procedure division** et est une suite d'instructions

5.1 Instructions d'affectation

En général,

move { ***id-var*** / **id-const** / **valeur** } **to liste [] de** ***id-var***

Exemple :
move 2 to nbhc

Pour l'affectation numérique,

compute liste [ ] de { ***id-var*** **[rounded]** } **= expr-arithm**

[size error bloc-instr-1] [not size error bloc-instr-2] end-compute

Exemple :
compute vm = nbkm / nbhc

Pour l'affectation d'un booléen,

set ***id-bool*** **to** { **true** / **false** }

Le positionnement à 'true' du booléen entraîne l'affectation automatique, à la variable associée, de la première valeur citée dans la clause 'value' de la déclaration du booléen. Idem pour le positionnement à 'false'.

5.2 Instruction d'initialisation

initialize liste [] de ***id-var*** [**replacing liste [] de**

({ **alphabetic** / **alphanumeric** / **numeric** / **alphanumeric-edited** / **numeric-edited** } **by [all]** { ***id-var*** / **const** })

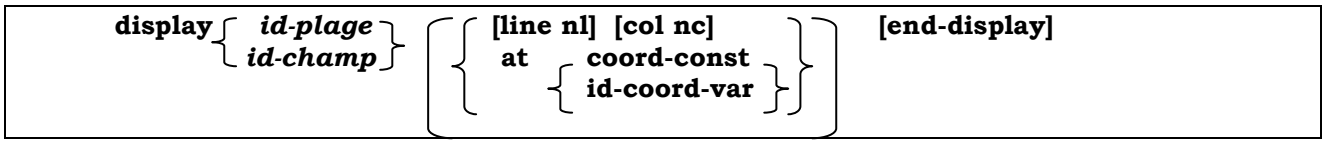
Cette instruction consiste, sauf indication différente avec l'option 'replacing', à initialiser à zéro tous les champs numériques et à blanc les autres. Cette instruction est également valable pour des types complexes tels que les structures et tableaux.

Exemple :
initialize adherent, numero, nom

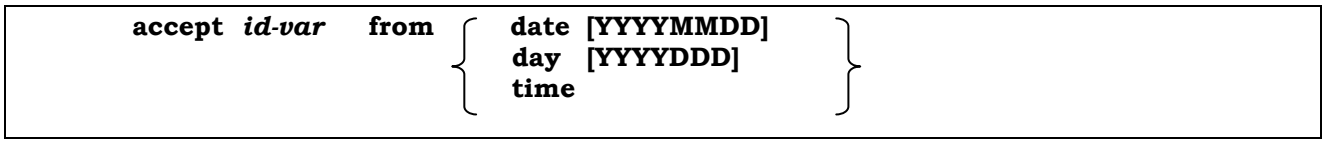
5.3 Instruction de saisie

accept { ***id-plage*** / ***id-champ*** } ({ **[line nl] [col nc]** / **at** { **coord-const** / **id-coord-var** } }) **[end-accept]**

5.4 Instruction d'affichage



5.5 Gestion des date et heure courantes



où id-var est une variable de type quelconque.

-option 'date'
La date du jour, extraite du registre DATE est affectée à id-var sous le format 'yymmdd', ou avec l'année sur 4 chiffres si on ajoute le schéma YYYYMMDD dans l'accept.

-option 'day'
Il y a affectation à id-var de la date du jour issue du registre DAY, avec l'année sur 2 caractères (ou 4 caractères si on ajoute le schéma YYYYDDD dans l'accept) et le numéro du jour dans l'année sur 3 chiffres.

-option 'time'
Le registre TIME donne l'heure sous le format 'hhmmsscc' c'est à dire en heures, minutes, secondes, centièmes.

Remarque :
- Les registres sont directement accessibles, sans déclaration.
- En ce qui concerne la date et l'heure courantes ne pas oublier la fonction, sans paramètre, CURRENT-DATE qui retourne 21 caractères alphanumériques dont
  - la date en format 'yyyymmdd' pour les 8 premiers caractères
  - l'heure en format 'hhmmsscc' pour les 8 suivants

La notion de référence modifiée étant aussi applicable à un appel de fonction alphanumérique pour accéder à une partie du résultat on peut, par exemple, écrire :

```
move function current-date (1 : 4) to date_courante(1 :4) pour affecter les 4 chiffres de l'année à la variable date_courante.
Move function current-date (5 : 2)  to date_courante(5 :2) pour affecter les 2 chiffres du mois à la variable date_courante.
```

```
1 date_courante pic 9(8).
```

ou bien (voir chapitre suivant, définition de structure).

```
1 date_courante.
        2 annee pic 9(4).
        2 mois pic 99.
        2 jour pic 99.
```

Exemple de calcul sur des dates : Ajout de 7 jours à la date courante.

```
        program-id. testdate.

        working-storage section.
        1 dte pic 9(8).
        1 val1 pic 9(8).
        1 val2 pic 9(8).

        screen section.
```

```
1 a-val1.
   2 line 6 col 10  pic 9(8) from dte.
1 a-val2.
   2 line 8 col 10  pic 9(8) from val2.

procedure division.
move function current-date(1:8) to dte(1:8)
display a-val1
compute val1= function integer-of-date(dte)
compute val1 = val1 + 7
compute val2 = function date-of-integer(val1)
display a-val2
end program testdate.
```

Exécution :

```
20081107
20081114
```

5.6 Instructions d'arrêt d'exécution d'un programme

goback : Assure l'arrêt de l'exécution du programme avec retour du contrôle au programme appelant, c'est-à-dire dans le cas d'un programme principal retour au système d'exploitation, donc arrêt définitif du programme.
stop run : Arrêt définitif du programme.

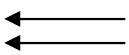
6 Complément sur l'utilisation de fichiers externes

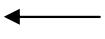
Il est possible d'utiliser des fichiers externes contenant des parties de code Cobol qui seront recopiées dans le code source du programme avant sa compilation.

Soit l'exemple suivant :

```
program-id . pg-racine-carree.
data division.
working-storage section.
1   x      pic  9(4)V99.
1   y      pic  99V99.
1   pause   pic  x.
screen section.
1 a-plg-titre line 2 col 20 'Calcul de racine carr' & x"82" & 'e'.
copy plage-saisie.
copy plage-res.
procedure division.
début.
  display  a-plg-titre
  copy mod-saisie.
  compute  y  =  function  sqrt(x)
  accept pause at 2480 with secure auto time-out after 2
  display  a-plg-res
  goback.
end program pg-racine-carrée.
```

Recopie des contenus des fichiers externes





Le contenu du fichier '**plage-saisie.cpy**' est :

```
1 m-plg-saisie.
      2 line 5 col 3 'Saisir un nombre r' & x"82" & 'el : '.
      2 s-chp-x pic z(4).99 to x required.
```

Le contenu du fichier '**plage-res.cpy**' est :

```
1 a-plg-res.
      2 line 8 col 3 'La racine carr' & x"82" & 'e est : '.
      2 a-chp-y pic zz.99 from y.
```

Le contenu du fichier '**mod-saisie.cpy**' est

```
display m-plg-saisie
        accept   s-chp-x
```

7 Notion de fonction

7.1 Les fonctions prédéfinies

Il existe de nombreuses fonctions prédéfinies (voir la documentation) que l'on appelle par l'instruction :

function ***id-fonc*** **(liste [,] de parm-eff)**

```
Exemple :     compute      y  =  function sin (x)
              Compute      Z =  function Pi *R**2      /*calcul de l'aire d'un cercle de rayon R*/
```

7.2 Les fonctions utilisateur

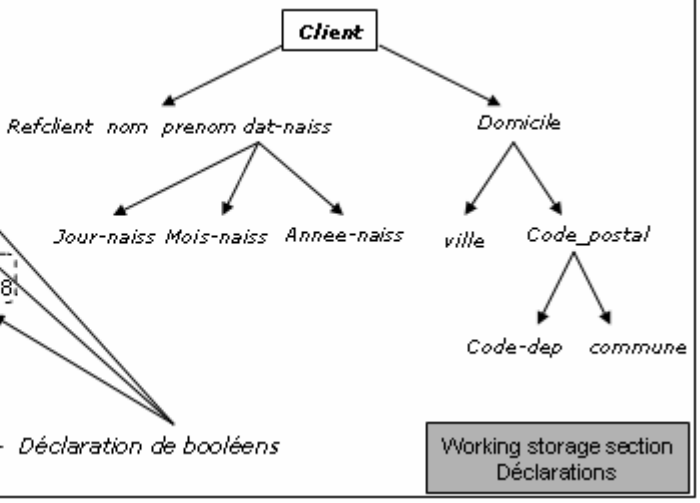
Seront détaillées dans le chapitre 7.

CHAPITRE 2 : TYPE STRUCTURE ET INSTRUCTIONS DE CONTROLE

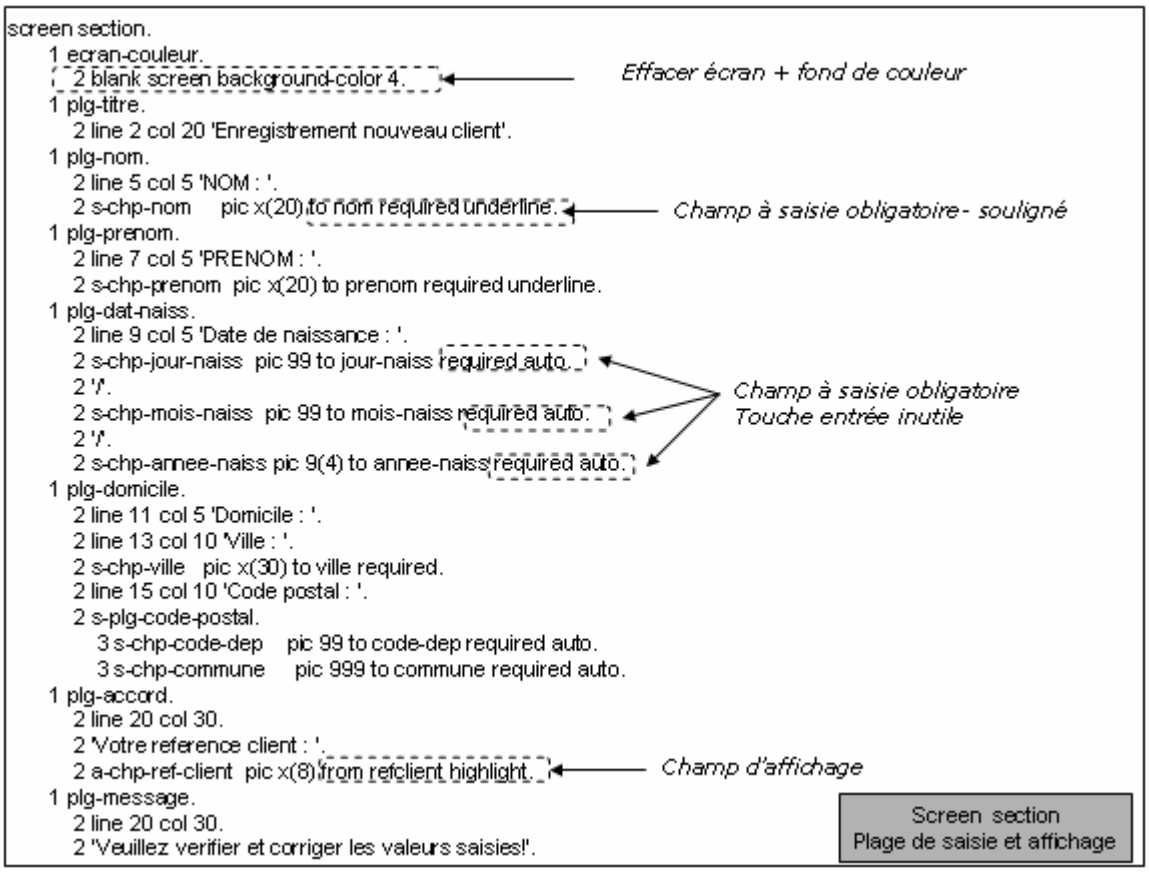
1 Exemple détaillé n°2

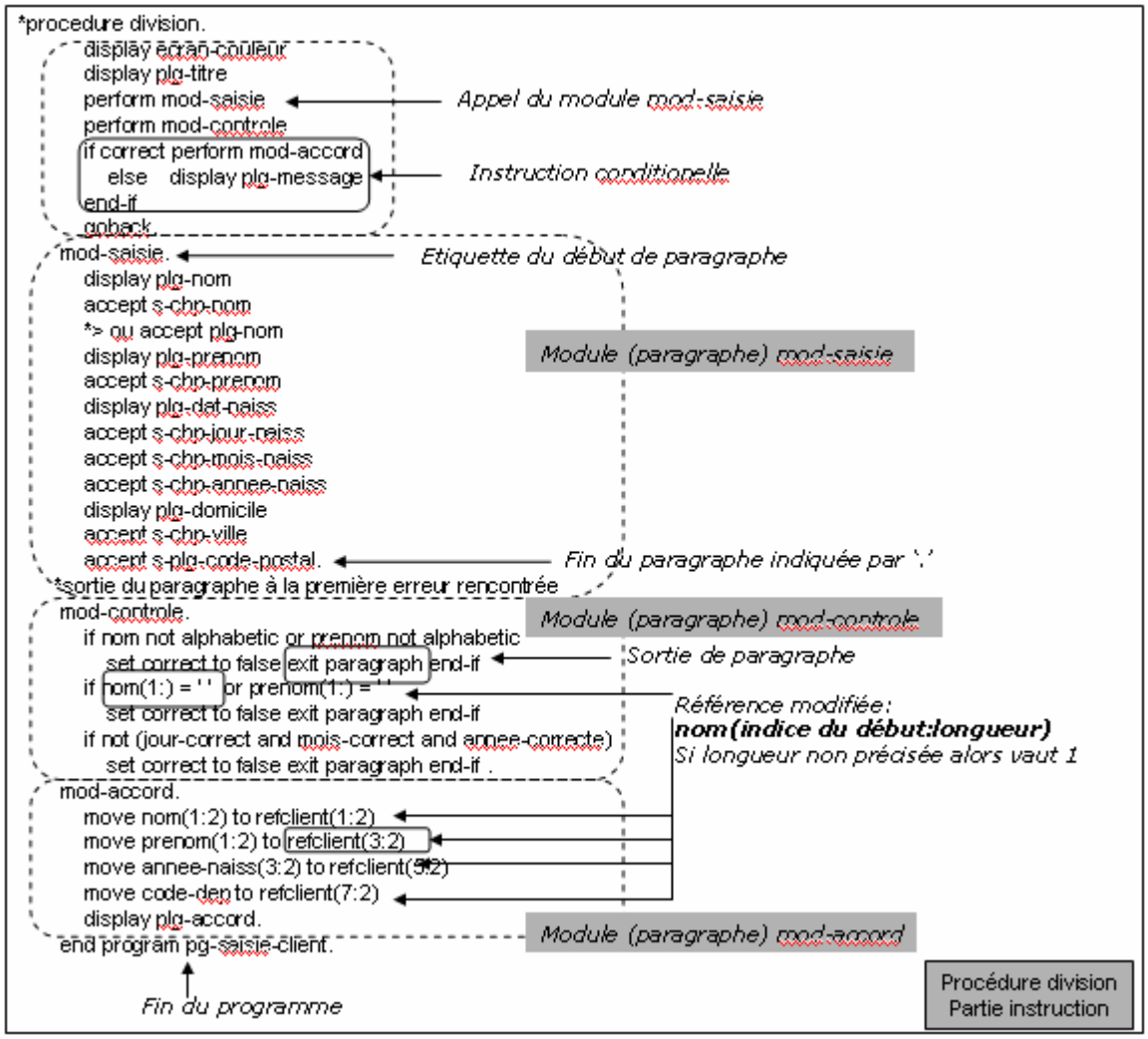
Soit le programme suivant qui permet la saisie d'information client et construit la référence du client.

```
program-id. pg-saisie-client.
*
working-storage section.
  1 client.
     2 refclient pic x(8).
     2 nom       pic x(20).
     2 prenom    pic x(20).
     2 dat-naiss.
        3 jour-naiss    pic 99.
        88 jour-correct value 1 thru 31.
        3 mois-naiss    pic 99.
        88 mois-correct value 1 thru 12.
        3 annee-naiss   pic 9(4).
        88 annee-correcte value 1900 thru 1988.
     2 domicile.
        3 ville         pic x(30).
        3 code-postal.
           4 code-dep  pic 99.
           4 commune   pic 999.
  1 pic x value 'o'.
  88 correct value 'o' false 'n'.
```




```
screen section.
     1 ecran-couleur.
       2 blank screen background-color 4.
     1 plg-titre.
        2 line 2 col 20 'Enregistrement nouveau client'.
     1 plg-nom.
        2 line 5 col 5 'NOM : '.
        2 s-chp-nom     pic x(20) to nom required underline.
     1 plg-prenom.
        2 line 7 col 5 'PRENOM : '.
        2 s-chp-prenom  pic x(20) to prenom required underline.
     1 plg-dat-naiss.
        2 line 9 col 5 'Date de naissance : '.
        2 s-chp-jour-naiss  pic 99 to jour-naiss required auto.
        2 '/'.
        2 s-chp-mois-naiss  pic 99 to mois-naiss required auto.
        2 '/'.
        2 s-chp-annee-naiss pic 9(4) to annee-naiss required auto.
     1 plg-domicile.
        2 line 11 col 5 'Domicile : '.
        2 line 13 col 10 'Ville : '.
        2 s-chp-ville   pic x(30) to ville required.
        2 line 15 col 10 'Code postal : '.
        2 s-plg-code-postal.
           3 s-chp-code-dep    pic 99 to code-dep required auto.
           3 s-chp-commune     pic 999 to commune required auto.
     1 plg-accord.
        2 line 20 col 30.
        2 'Votre reference client : '.
        2 a-chp-ref-client  pic x(8) from refclient highlight.
     1 plg-message.
        2 line 20 col 30.
        2 'Veuillez verifier et corriger les valeurs saisies!'.
```





2 Définition de structure

A effectuer en **working-storage section**.

2.1 Déclaration d'une structure

Une structure est une suite de lignes de déclaration des champs qui la constituent.
C'est-à-dire de la forme :

nb-niv [*id-champ*] { [clause **pic**] | id-type } [clause **value**] **.**

avec un nombre niveau (nb-niv) dans [1, 49]
Un arbre est, dans une imbrication de structures, la structure la plus externe. La racine d'un arbre (structure indépendante) est toujours de niveau **1** .

2.2 Déclaration d'un type structure

1 id-type-struct **typedef .**

_ _ _ _ _ _ _ _ _ _ _ _ _ _

suite des lignes de déclaration de la structure

_ _ _ _ _ _ _ _ _ _ _ _ _ _

```
Exemple :
1 petit-entier     pic 9(4)   typedef .     *>déclaration d'un type simple entier
1 grand-entier  pic 9(18)  typedef .
1 id-var1          petit-entier.              *>déclaration d'une variable de type petit-entier
1 id-var2          grand-entier.              *>déclaration d'une variable de type grand-entier
```

```
1 typ-adresse  typedef.          //déclaration du type structuré 'typ-adresse'
        2 num              pic 9(4).
        2 rue              pic x(60).
        2 codpostal        pic 9(5).
        2 ville            pic x(50).

Ce qui permet de déclarer 'adresse' dans adhérent par exemple : adhérent.
1…….
        2        adresse                  typ-adresse
```

2.3 Déclaration d'un type alternatif

```
1       id-type-alt      typedef .
        nb-niv-a         id-type-1.
        _ _ _ _ _ _ _ _ _ _ _ _ _ _
        suite des lignes de déclaration de la structure id-type-1
        _ _ _ _ _ _ _ _ _ _ _ _ _ _

liste [ ] de

        nb-niv-a         id-type-k  redefines  id-type-1.
        _ _ _ _ _ _ _ _ _ _ _ _ _ _
        suite des lignes de déclaration de la structure id-type-k
```

```
Exemple :

1 oper.
        2 codoper          pic x.
        2 refprodoper      pic x(5).
        2 prodacreer.
                3 designprodoper    pic x(20).
                3 qteprodoper       pic 9(4).
                3 prixprodoper      pic 9(4)v99.
                3 fourprodoper      pic x(20).
        2 prodamodif    redefines   prodacreer.
                3 qteprodoper    pic 9(4).
                3                pic x(46).
```

Il faut que la redéfinition d'un objet apparaisse directement après sa définition. On peut redéfinir plusieurs fois un même objet.

2.3.1 Déclaration d'une variable de type alternatif

```
nb-niv id-var  id-type-alt
```

2.3.2 Déclaration d'une désignation de structure bornée

```
decl-struct
66 id-struct-bornée  renames  id-champ-deb  thru  id-champ-fin .
```

où *id-champ-deb* et *id-champ-fin* sont des champs de la structure déclarée immédiatement au-dessus.

Exemple :

```
1 client.
        2 nom          pic x(30).
        2 prenom       pic x(30).
        2 adresse.
                3 numero  pic 9(3).
                3 rue       pic x (40).
                3 code postal   pic x(5).
                3 ville      pic x(20).
66   libelle renames nom thru prenom.
```

3 Notion de paragraphe

Un paragraphe commence par une étiquette (l'identificateur du paragraphe) placée en colonne 8 et suivie d'un point. Un paragraphe est constitué d'une suite d'instructions et se termine à la rencontre d'une autre étiquette de paragraphe ou par la fin physique du programme.

Le premier paragraphe de la procédure division, le seul dont l'étiquette n'est pas obligatoire, est le **paragraphe principal** , les autres sont les **paragraphes secondaires**.

```
étiq-par.
        _ _ _ _ _ _ _ _
        liste [] d'instructions
        _ _ _ _ _ _ _ _ .
```

3.1 Appel d'un paragraphe

```
perform  etiq-de-parag
```

L'appel d'un paragraphe, en un point de la procédure division est équivalent à la recopie de son contenu en ce point. Il est possible d'appeler un enchaînement de paragraphes défini par un paragraphe début et un paragraphe fin, dans le sens de l'ordre d'exécution de ceux-ci.

```
perform  etiq-de-parag1  thru  etiq-de-parag2
```

L'instruction suivante permet l'exécution d'un bloc d'instructions (une instruction composée).

```
perform
        _ _ _ _ _ _ _ _
        liste [] d' instructions
        _ _ _ _ _ _ _ _
        end-perform
```

3.2 Sortie d'un paragraphe

```
exit paragraph
```

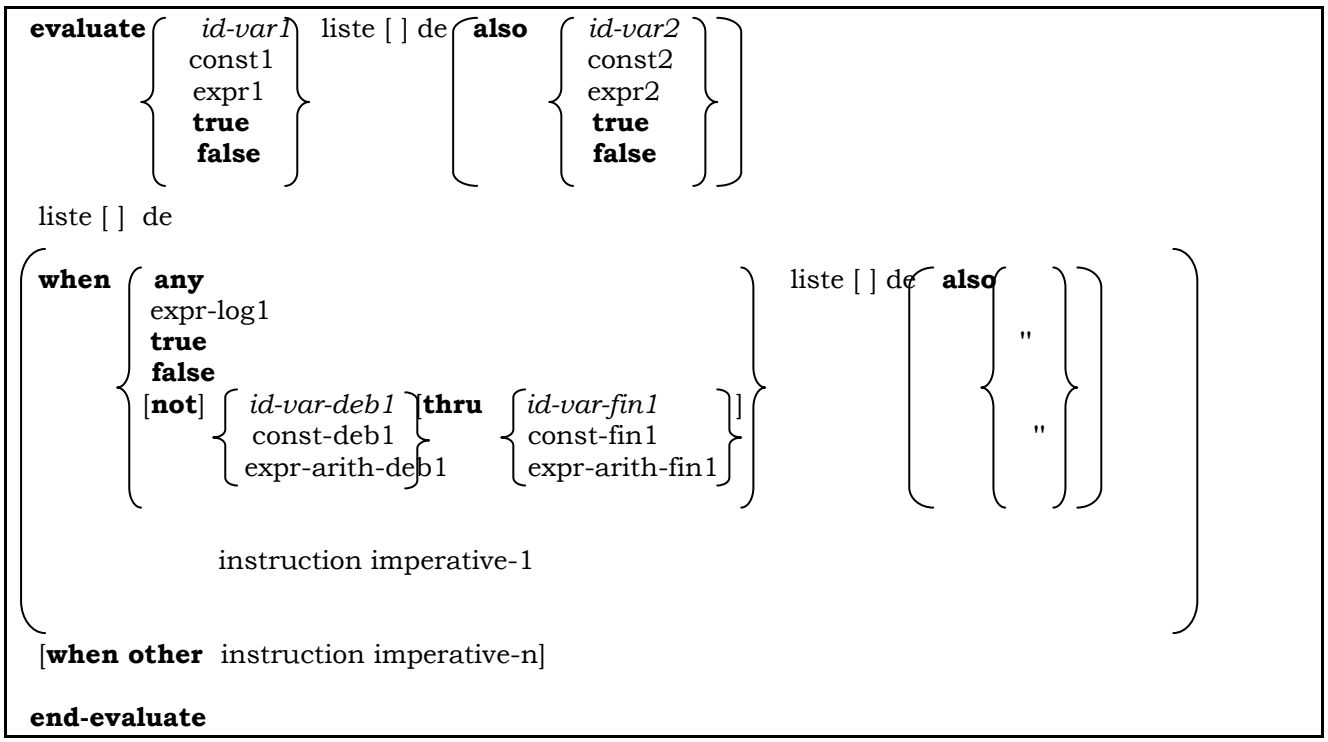
Le contrôle est transféré à la fin du paragraphe.

4 Les instructions de contrôle

```
if expr-bool  then  [perform]
                       suite d'instructions
                    [end-perform]
               [ else  [perform]
                       suite d'instructions
                    [end-perform]  ]
end-if
```

Exemple :

```
if correct then perform mod-accord
           else  display plg-message
        end-if
```



Exemples :

```
evaluate  id-var
          when 1  inst-imper1
          when 2  inst-imper2
          when other  inst-imper3
end-evaluate

evaluate true
          when expr-bool1  inst-imper1
          when expr-bool2  inst-imper2
          when other  inst-imper3
end-evaluate

evaluate  expr-bool1  also  expr-bool2
          when true  also  true  inst-imper1
          when  true  also  false  inst-imper2
          when false  also  true  inst-imper3
          when  false  also  false  inst-imper4
end-evaluate
```

CHAPITRE 3 : LES ITERATIONS

1 Itération portant sur une instruction composée

Les itérations peuvent s'exprimer de différentes manières suivant les situations.

1.1 Itération un nombre de fois connu

perform { *id-var-ent* / const-ent } **times**

_ _ _ _ _ _ _ _ _

suite d'instructions

_ _ _ _ _ _ _ _ _

end-perform

La suite *d'instruction* est répétée le nombre de fois indiqué par la valeur de l'entier ou de la constante.

1.2 Itération avec condition

perform [**test** { **before** / **after** }] **until** { cond-arrêt / **exit** }

_ _ _ _ _ _ _ _ _

suite d'instructions

_ _ _ _ _ _ _ _ _

end-perform

La suite d'instruction est répétée jusqu'à ce que la condition d'arrêt soit vérifiée.

Le test de la condition peut s'effectuer soit avant l'itération soit après : test before ou test after.

1.2.1 Test avant

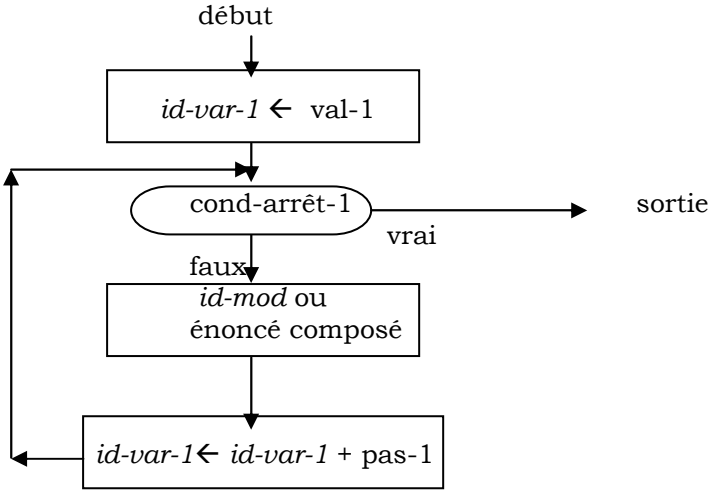


*Figure1 : cas d'une seule variable avec '**test before**'*

1.2.2 Test après

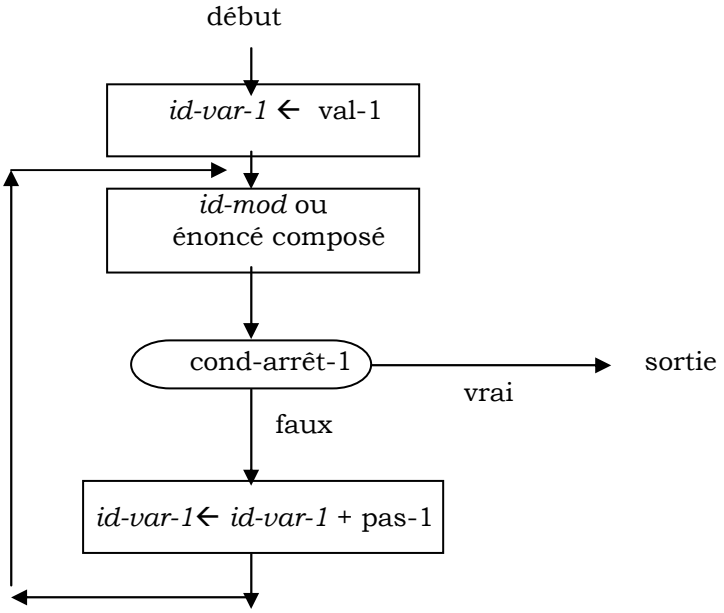


*Figure2 : cas d'une seule variable avec '**test after**'*

1.2.3 Avec 2 variables et test after

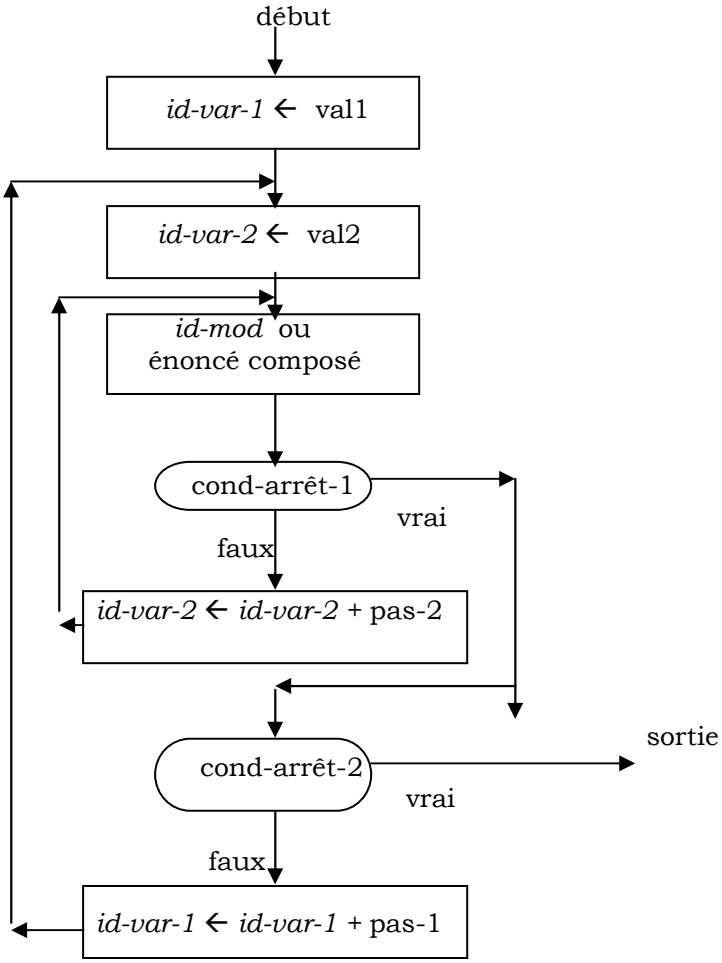


*Figure3 : cas de deux variables avec '**test after**'*

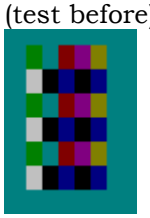
1.3 Itération avec gestion d'une variable

```
perform [ test { before } ]
                { after  }
        varying id-va-r1 from val-1 by pas-1 until cond-arrêt-1

        liste [ ] de [ after id-var-j from val-j by pas-j until cond-arrêt-j ]
  _ _ _ _ _ _ _ _ _
  suite d'instructions
  _ _ _ _ _ _ _ _ _
end-perform
```

Exemple 1 :

```
program-id. pg-rect-coul.
 1 nl     pic 99.          *>numéro courant de ligne
 1 nc     pic 99.          *>numéro  courant de colonne
 1 bgc    pic 9 value 1.       *>variable couleur de fond

screen section.
1 effacer-ecran blank screen background-color 3.
1 a-chp-car-coul line nl col nc '  ' background-color bgc.
display effacer-écran

perform test before varying nl from 5 by 1 until nl > 10
          after nc from 5 by 1 until nc = 10
       compute bgc = bgc + 1
      display a-chp-car-coul
end-perform.
end program pg-rect-coul.
```

L'exécution est la suivante :
(test before)

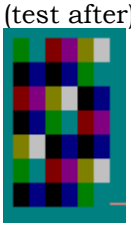


6 lignes et 5 colonnes

Exemple 2 :

```
program-id. pg-rect-coul.
 1 nl     pic 99.          *>numéro courant de ligne
 1 nc     pic 99.          *>numéro  courant de colonne
 1 bgc    pic 9 value 1.       *>variable couleur de fond

screen section.
1 effacer-ecran blank screen background-color 3.
1 a-chp-car-coul line nl col nc '  ' background-color bgc.
display effacer-écran

perform test after varying nl from 5 by 1 until nl > 10
          after nc from 5 by 1 until nc = 10
       compute bgc = bgc + 1
      display a-chp-car-coul
end-perform.
end program pg-rect-coul.
```

L'exécution est la suivante :
(test after)



7 lignes et 6 colonnes

Exemple 3 :

```
perform  until  exit
_ _ _ _ _
          exit perform
_ _ _ _ _
end-perform
```

Sortie d'une itération sans fin.

2 Instructions d'itération portant sur un module bloc

Un module bloc est un paragraphe (*etiq-par*) ou une suite de paragraphes caractérisée par un paragraphe début (*etiq-par-deb*) et un paragraphe fin (*etiq-par-fin*) et dont l'exécution commence avec la première instruction de *id-par-deb* et se termine avec la dernière instruction de *id-par-fin*.

Une suite de paragraphes peut être placée dans une section, unité de composition au plus haut niveau d'une procédure division.

Une procédure division est soit entièrement composée de sections soit entièrement composée de paragraphes.

On désignera par *id-mod* un module :

etiq-par
{ *etiq-par-deb* **thru** *etiq-par-fin*
{ *etiq-sec*

où *etiq-sec* désignera une section définie comme suit :

etiq-sec **section .**
liste [] de paragraphes

2.1 Itération d'un module bloc

perform *id-mod* { *id-var-ent* / const-ent } **times**

id-mod est répété le nombre de fois indiqué par la valeur de l'entier ou de la constante.

perform *id-mod* [**test** { **before** / **after** }] **until** { cond-arrêt / **exit** }

id-mod est répété jusqu'à ce que la condition d'arrêt soit vérifiée.

perform *id-mod* [**test** { **before** / **after** }]
varying *id-var-1* **from** val-1 **by** pas-1 **until** cond-arrêt-1
liste [ ] de **after** [*id-var-j* **from** val-j **by** pas-j **until** cond-arrêt-j]

2.2 Instructions de sortie d'un appel itératif d'un module bloc

Cas où le module bloc est un paragraphe :

exit paragraph

Cas où le module bloc est une section

exit section

3 Exemple détaillé n°3

Le programme suivant permet de calculer la racine carrée d'une suite de nombre. Le calcul s'arrête quand l'utilisateur tape 'N' ou 'n' à la question 'Voulez-vous continuer ?'

```
program-id. pg-suite-racine-carre.
data division.
working-storage section.
1   x      pic  9(4)V99.
1   y      pic  99V99.
1   choix  pic  x.
screen section.
1 a-plg-titre line 2 col 20 'Calcul de racine carr' & x"82"
                       & 'e'.
1 m-plg-saisie.
   2 line 5 col 3 'Saisir un nombre r' & x"82" & 'el : '.
   2 s-chp-x pic z(4).99 to x required.
1 a-plg-res.
   2 line 8 col 3 'La racine carr' & x"82" & 'e est : '.
   2 a-chp-y pic zz.99 from y.
1 m-plg-choix.
   2 line 20 col 50 'Voulez-vous continuer : ?'.
   2 s-chp-choix pic x to choix.
procedure division.
début.
  display  a-plg-titre
  perform test after until function upper-case(choix)='N'
  display m-plg-saisie
  accept   s-chp-x
  compute  y  =  function  sqrt(x)
  display a-plg-res
  display m-plg-choix
  accept s-chp-choix
  end-perform
  goback.
end program pg-suite-racine-carre.
```

CHAPITRE 4 : LE TYPE CHAINE DE CARACTERES

Une chaîne de caractères est une séquence de caractères entre quotes '...' ou entre guillemets "...".

1 Partie Déclaration (en working-storage section)

1.1 Descripteur de type chaîne de caractères

pic x (*id-const-ent* | val-ent) ou *id-type-ch*

1.2 Déclaration d'une constante de type chaîne

78 *id-const-ch* **value** val-ch **.**

1.3 Déclaration d'une variable simple *(variable alphanumérique)*

nb-niv *id-var-ch* { descr-type-ch | id-type-ch } **.**

Le nombre niveau est **1** si la variable est indépendante sinon il traduit la position (2 à 49) de la variable dans la structure d'appartenance.

1.4 Déclaration d'une variable de type structure *(cas particulier de variable chaîne)*

nb-niv *id-var-struct* **.**
_ _ _ _ _ _ _
suite de lignes de déclaration des composants
_ _ _ _ _ _ _

Remarque importante : Une variable de type structure est toujours de type chaîne de caractères, sa valeur est la concaténation des valeurs des feuilles de la structure.

1.5 Déclaration d'un type chaîne de caractères

1 id-type-ch descr-type-ch **typedef.**

2 Manipulation de chaîne de caractère

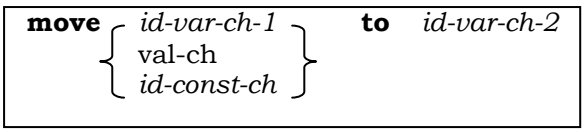
2.1 Accès à une sous-chaîne (référence modifiée)

id-var-ch **(** rang-deb **:** long-ss-chaîne **)**

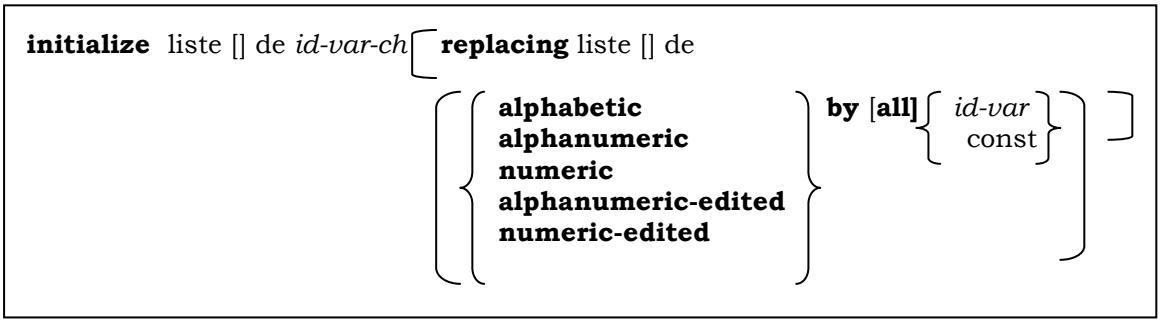
l'accès au caractère de rang k s'écrit donc *id-var-ch* **(** k **:** 1 **)**

2.2 Affectation d'une valeur à une variable de type chaîne de caractères

2.2.1 Instruction d'affectation



2.2.2 Instruction d'initialisation



2.3 Comparaison de chaînes de caractères

Dans la comparaison de deux chaînes de caractères, la longueur physique des chaînes n'intervient pas, seuls les caractères contenus sont examinés. La première inégalité rencontrée, lors de la comparaison deux à deux des caractères de même rang dans les deux chaînes, détermine le résultat de la comparaison

2.4 Quelques fonctions prédéfinies sur les chaînes de caractères

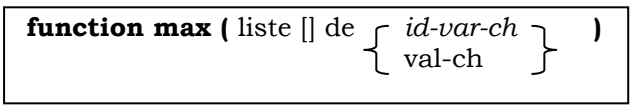
2.4.1 Fonction retournant la longueur d'une chaîne

donne la valeur de la longueur totale de la chaîne y compris les blancs (telle que définie en pic) ou encore :

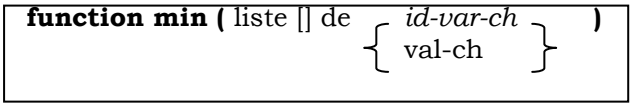
Exemple :

```
move length of chaine1 to lg
```

2.4.2 Fonction retournant le maximum d'une liste de valeurs



2.4.3 Fonction retournant le minimum d'une liste de valeurs



2.4.4 Fonction qui retourne la valeur numérique que représente la chaîne

function numval ({ *id-var-ch* / val-ch } **)**

Exemple :
Compute lg = fonction numval('123') retourne la valeur numérique 123.

2.4.5 Fonction qui retourne la chaîne avec l'ordre inverse de ses caractères

function reverse ({ *id-var-ch* / val-ch } **)**

2.5 Instructions spécifiques sur chaînes de caractères

2.5.1 Concaténation de chaînes de caractères

string liste [ ] de [{ *id-var-ch-j* / val-ch-j } **....delimited** { *id-délim* / val-délim / **size** }] **into** *id-var-ch*

[**pointer** *id-pointeur*] [**overflow** inst-imper1] [**not overflow** inst-imper2]

[**end-string**]

Les contenus des chaînes *id-var-ch-j* , val-ch-j , pour j=1, 2, . . ., sont concaténés, pour chacun, à hauteur de la sous-chaîne précédant la première occurrence du délimiteur indiqué par id-délim, val-délim ou dans son entier avec l'option 'size'.

Le résultat est placé dans la variable chaîne 'id-var-ch'.
L'option 'pointer' permet de connaître la position, dans 'id-var-ch' du dernier caractère transféré à la fin de l'exécution de l'instruction 'string' ; ne pas oublier de déclarer 'id-pointeur' en temps que variable entière.

L'option '[not] overflow' permet de prendre le contrôle s'il [n'] y a [pas] débordement dans la variable id-chaîne, avec suspension de l'affectation et exécution de l'instruction 'inst-imper1' ['inst-imper2'].

Le pointeur, s'il est utilisé, doit être initialisé avant la première exécution de l'instruction 'string' qui le gère ensuite automatiquement. La partie de *id-var-ch* non modifiée par la concaténation reste inchangée.
Si *id-var-ch* est une chaîne de n caractères, il y a débordement si la valeur du pointeur *id-pointeur* n'appartient pas à l'intervalle [1,n].

Exemple:

Dans la partie déclaration :

```
1  chaine1 pic x(10).
1  chaine2 pic x(10).
1  chaine3 pic x(50).
```

Dans la partie instruction :

```
move 'exemple1' to chaine1
move 'exemple2' to chaine2
string chaine1 chaine2 into chaine3
```

La variable chaine3 contient alors la chaine : 'exemple1 exemple2 '

Si on remplace l'instruction string par précédente par :

```
string chaine1 chaine2 delimited ' ' into chaine3
```

La variable chaine3 contient alors la chaine : 'exemple1exemple2 '

Si on remplace l'instruction string par précédente par :

```
string chaine1 chaine2 delimited 'p' into chaine3
```

La variable chaine3 contient alors la chaine : 'exemexem '

Si on remplace l'instruction string par précédente par :

```
string chaine1 chaine2 delimited 'p' into chaine3 pointer lg
```

et en prenant la précaution d'initialiser lg à 1, on obtient :

La variable chaine3 contient alors la chaine : 'exemexem '
La variable lg contient la valeur : 9

2.5.2 Déconcaténation d'une chaîne de caractères (extraction de sous-chaînes)

unstring *id-var-ch* **delimited** liste [ ] de [[**or**] [**all**] { *id-délim-j* / val-délim-j }]

into liste [ ] de [*id-var-ch-k* [**delimiter** *id-délim-k*] [**count** *id-var-nbcar-k*]]

[**pointer** *id-pointeur*] [**tallying** *id-var-nb-ch*] [**overflow** inst-imper1]

[**not overflow** inst-imper2]

[**end-unstring**]

Il s'agit d'extraire de 'id-var-ch' les sous-chaînes successives délimitées par l'un quelconque des délimiteurs cités et de les ranger, dans l'ordre, dans les variables chaînes que sont '*id-var-ch-k* ' pour k=1, 2, . . .
La variable entière (à déclarer) 'id-pointeur' contient, à la fin de l'exécution de 'unstring', le rang dans 'id-var-ch' du dernier caractère concerné par la déconcaténation.
L'instruction 'unstring' s'arrête, par exemple, quand toutes les chaînes réceptrices sont affectées.
La variable entière '*id-var-nb-ch*' (à déclarer) indique combien de sous-chaînes ont été extraites pendant l'exécution de 'unstring'.
Parmi les cas de débordement on peut citer :

- une valeur de 'id-pointeur' en dehors des rangs possibles de caractères dans *id-var-ch*
- Il reste des caractères non examinés dans '*id-var-ch* alors que toutes les chaînes résultat sont affectées.

Dans la partie instruction :

```
move 'exemple1 exemple2' to chaine3
```

L'exécution de l'instruction :

```
unstring chaine3 delimited '  ' into chaine1 chaine2
```

donnera comme résultat :

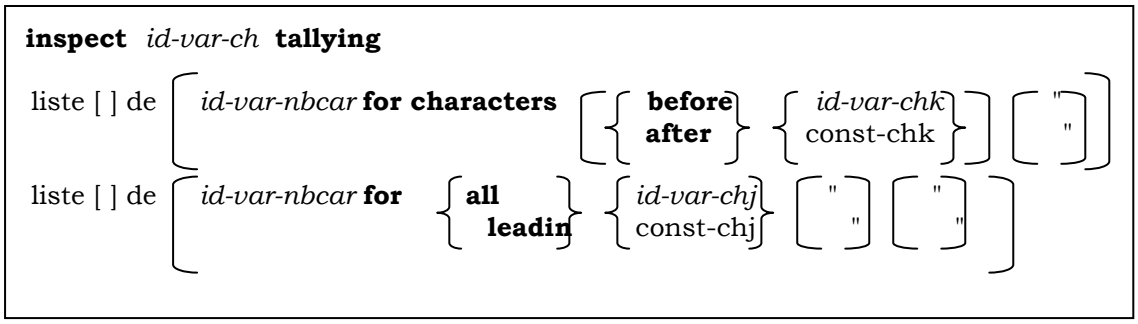
- le contenu de chaîne 1 : 'exemple1 '
- le contenu de chaîne 2 : 'exemple2 '

si on remplace l'instruction unstring précédente par :

```
unstring chaine3 delimited ' ' into chaine1 chaine2 pointer lg
```

chaine1 et chaine2 contiendront les mêmes valeurs que précédemment et lg contiendra la valeur 19.

2.5.3 Instruction de comptage de caractères



Le résultat du comptage est placé dans une variable entière *id-var-nbcar*, par ailleurs déclarée.

La variable chaine est déclarée comme pic x(30)

```
move 'EFABDBCGABEFGG' to chaine
inspect chaine tallying  nb1 for all "AB", all "D";
inspect chaine tallying  nb2  for all "BC"              ;
inspect chaine tallying  nb3 for leading "EF";
inspect chaine tallying  nb4 for leading "B";
inspect chaine tallying  nb5 for characters;
```

Les résultats sont :

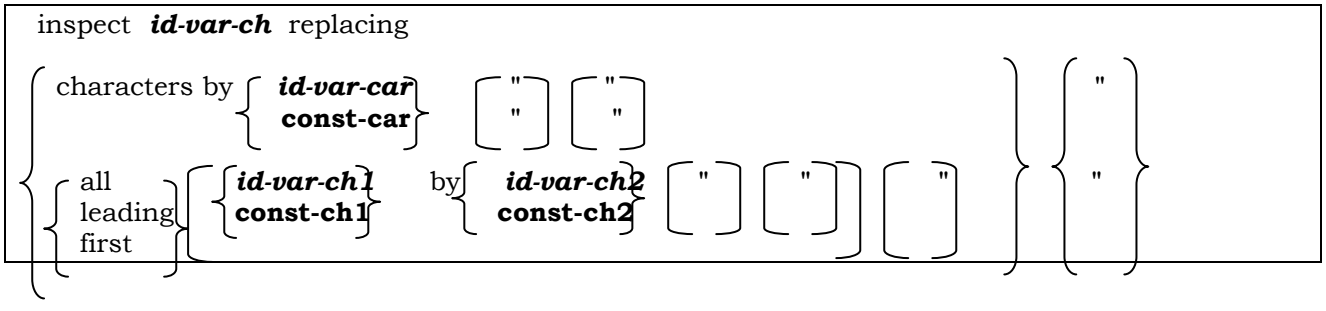
La chaîne initiale	Nb1	Nb2	Nb3	Nb4	Nb5
'EFABDBCGABEFGG'	3	1	1	0	30
'BABABC'	2	1	0	1	30
'BBBC'	0	1	0	3	30

Inspect mot tallying avant for leading ' '
Compte dans la variable **avant** les caractères blancs situés en début de la chaîne contenue dans mot.

Inspect function reverse(mot) tallying apres for leading ' '
Compte dans la variable **apres** les caractères blancs situés en fin de la chaîne contenue dans mot.

Inspect mot tallying avant for characters before space
Compte dans la variable **avant** le nombre de caractères avant le premier caractère blanc.

2.5.4 Instruction de remplacement de caractères



Exemple :
Inspect texte replacing all car1 by car2
Remplace toutes le occurrences du caractère car1 par car2 dans la chaîne texte.

move 'ABBABDC' to chaine
inspect chaine replacing all "AB" by "XY", "D" by "X"
le contenu de chaine est '**XYBXYXC**'.

move 'EFABBEFABDC EFBABC' to chaine
inspect chaine replacing leading "EF" by "TU"
le contenu de chaine est **'TUABBEFABDC EFBABC'**

move 'GARDF GHT' to chaine
inspect chaine replacing first "G" by "R"
le contenu de chaine est **'RARDF GHT'**

2.5.5 Instruction de comptage et remplacement

inspect **tallying . . . . . .** **replacing . . . . . .**

2.5.6 Instruction de conversion de caractères

inspect id-var **converting** { id-var1 / const1 } **to** { id-var2 / const2 } [{ **before** / **after** } { id-var3 / const3 }]

Chaque caractère de id-var qui apparaît dans la chaîne 'id-var1' (ou const1) est remplacé par le caractère de rang correspondant dans 'id-var2' (ou const2), éventuellement avant ou après une première occurrence dans id-var de la chaîne 'id-var3' (ou const3).

move 'AC"AEBDFBCD#AB"D to chaine
inspect chaine converting "ABCD" to "XYZX" after quote before "#".

le résultat est : AC"XEYXFYZX#AB"D

2.5.7 Test du type de l'ensemble des caractères d'une chaîne

id-var-ch [**not**] **alphabetic**
numeric
alphabetic-upper
alphabetic-lower

id-var-ch alphabetic //vrai si id-var ne contient que des lettres et des espaces
id-var-ch numeric //vrai si id-var ne contient que des chiffres et un signe éventuel
id-var-ch alphabetic-upper //vrai s'il n'y a que des lettres majuscules ou des espaces
id-var-ch alphabetic-lower //vrai s'il n'y a que des lettres minuscules ou des espaces

CHAPITRE 5 : FONCTIONS ET SOUS-PROGRAMMES

Au niveau de Cobol les concepts contribuant à la modularité sont essentiellement ceux de **programme**, **paragraphe**, **section**, de **fonction** et de **sous-programme**.

Une Unité d'Exécution (UE) est constituée d'un programme principal et des éventuelles fonctions programmes et sous-programmes qu'il appelle.

1 Les fonctions

En plus des fonctions prédéfinies, l'utilisateur peut définir ses propres fonctions.
Une fonction possède des paramètres formels, tous par principe paramètres d'entrée. Son appel génère une valeur du type de la fonction.

1.1 Définition d'une fonction

function-id. *id-fonc.*
[**file-control.**]
[**data division.**]
[**file section.**]
[**working-storage section.**]
linkage section.
_ _ _ _ _ _ _
liste de dcl-parm-form
_ _ _ _ _ _ _
dcl-parm-res
[**screen section.**]
procedure division [**using** liste [ ] de *id-parm-form*] **giving** *id-parm-res.*
_ _ _ _ _ _ _
corps de la function
_ _ _ _ _ _ _
end function *id-fonc.*

Pour être utilisable une fonction doit être compilée et enregistrée dans un répertoire (**repository**) de (signatures de) fonctions créé implicitement dans le répertoire courant.

1.1.1 Enregistrement d'une fonction

Est déclenché par une commande (repository) qui précède la définition de la fonction.
L'enregistrement peut s'effectue dans le répertoire courant par la commande suivante :

```
$set repository "update on"

        dcl-fonc
```

ou dans un répertoire précisé par son chemin complet (chm-rep)

```
$set rdfpath "chm-rep"
$set repository "update on"

dcl-fonc
```

L'option 'checking' de la commande repository déclenche un contrôle de conformité des profils d'appels de fonctions, dans le programme qui suit, aux signatures dans le répertoire des fonctions.

1.1.2 Déclaration d'une fonction

A condition qu'elle soit répertoriée (repository) une fonction peut être appelée dans un programme si elle est déclarée, au début de ce programme, dans le paragraphe d'étiquette 'repository' (dans la configuration section).

repository.
liste [ ] de **function** *id-fonc* **.**

1.1.3 Appel d'une fonction

id-fonc **(** liste [,] de parm-eff **)**

1.1.4 Sortie d'une fonction

exit function

1.2 Exemple d'utilisation de fonction

Etape 1 : Définition de la fonction dans le fichier '**pg-fonction.cbl'**

```
$set repository "update on"
function-id. fn-fonc.
linkage section.
1 val1  pic 99.
1 val2  pic 99.        PARAMETRES de la fonction
1 val3  pic 99.
screen section.
1 a-plg-res.
   2 line 12 col 3 'Le résultat dans la fonction : '.
   2 a-val3 pic 99 from val3.
procedure division using val1 val2 giving val3.
  compute val3= val1 + val2
  display a-plg-res
end function fn-fonc.
```

Etape 2 : appel de la fonction qui se trouve dans le fichier '**pg-principal.cbl'**

```
program-id. appel-pg-PP.
repository.
function fn-fonc.
data division.
working-storage section.
1  entier1 pic 99.
1  entier2 pic 99.
1  entier3 pic 99.

screen section.
1 a-plg-titre line 2 col 20 'Test sur entier'.
1 a-plg-res1.
   2 line 9 col 3 'le 1er nombre : '.
   2 a-entier1 pic 99 from entier1.
1 a-plg-res2.
   2 line 10 col 3 'le 2eme nombre : '.
   2 a-entier2 pic 99 from entier2.
1 a-plg-res3.
```

```
           2 line 13 col 3 'Le resultat dans le pg principal : '.
           2 a-entier3 pic 99 from entier3.

     procedure division.
       initialize entier1 entier2 entier3
       display  a-plg-titre
       compute entier1=10
       compute entier2=20
       display a-plg-res1
       display a-plg-res2
       compute entier3=fn-fonc(entier1,entier2)        APPEL de la fonction
       display a-plg-res3
       goback.
     end program appel-pg-PP.
```

```
                  Test sur entier




le 1er nombre : 10
le 2eme nombre : 20

Le rÚsultat dans la fonction : 30
Le resultat dans le pg principal : 30
```

2 Les sous-programmes

Un sous-programme possède des paramètres qui, comme pour une procédure, permettent d'échanger des données avec un programme appelant.

2.1 Définition d'un sous-programme

program-id. *id-sous-prog.*
[**file-control.**]
[**data division.**]
[**file section.**]
[**working-storage section.**]

linkage section.
_ _ _ _ _ _ _
liste de parm-form
_ _ _ _ _ _ _
[**screen section.**]
*>point d'entrée principal
procedure division using liste [] de { { **by reference** / **by value** } *id-parm-form* }

[**returning** *id-var*] **.**

_ _ _ _ _ _ _
*>point de sortie
{ **goback** / **exit program** [**giving** { val-ent / *id-var-ent* }] }
_ _ _ _ _ _ _

*>point d'entrée secondaire

entry '*id-entrée*' liste [] de [{ **by reference** / **by value** } *id-parm-form*]

[**returning** *id-var*] **.**

_ _ _ _ _ _ _

end program *id-sous-prog***.**

Le passage des paramètres peut être envisagé par adresse ou par valeur.
Avec l'option **by reference** on peut au moment de l'appel (call) choisir l'un ou l'autre des modes de passage.
Avec l'option **by value** le paramètre ne doit pas avoir une taille de plus de 8 octets (2 mots).

Chaque point d'entrée peut posséder des paramètres spécifiques pris parmi ceux qui sont déclarés dans la **linkage section**. A ce niveau, le passage des paramètres peut être envisagé par adresse ou par valeur.

Au niveau du point d'entrée :

- avec l'option **by reference** pour un paramètre on peut au moment de l'appel (call) choisir lun ou l'autre des modes de passage.
- Avec l'option **by value** le paramètre ne doit pas avoir une taille de plus de 8 octets (2 mots
- Avec by reference optional le paramètre effectif correspondant peut être omited

2.2 Appel d'un sous-programme :

call { *id-sous-prog* / *Id entrée* } [**using** liste [] ({ **by reference** / **by content** } id-parm-eff)]

[**exception** instr-imper1]
[**not exception** instr-imper2]

end-call

Les paramètres effectifs sont associés par rang aux paramètres formels correspondants. Le mode de passage des paramètres peut-être :

- **by reference** : c'est-à-dire par **adresse**. Au moment du déclenchement de l'appel le paramètre formel se voit attribuer une adresse qui est celle du paramètre effectif correspondant. Toute modification du paramètre effectif est répercutée au niveau du paramètre formel dans le programme appelant.

- **by content** : c'est-à-dire **par valeur**. Au moment du déclenchement de l'appel, l'adresse du paramètre formel est distincte de celle du paramètre effectif. Dans ce cas, au déclenchement de l'appel, la valeur du paramètre effectif est affectée au paramètre formel. La modification de la valeur de ce dernier n'affecte en rien le contenu du paramètre effectif.

L'option exception permet de prendre le control dans le cas où, à l'appel du sous-programme, ce dernier ne peut être chargé dynamiquement ou lorsqu'il n'y a pas assez de place mémoire.

2.3 Sortie d'un sous programme

Qu'il s'agisse d'un appel du point d'entrée principal ou d'un point d'entrée secondaire, on peut sortir du corps du sous- programme.

{ **goback** / **exit program [giving** …] }

Le goback est à privilégier.

2.4 Exemples de sous-programmes

2.4.1 Exemple 1 - version 1

Cette première version comporte un programme principal et deux sous-programmes permettant de calculer la somme et le produit de deux nombres. Les 2 nombres et le résultat de leur somme et leur produit apparaissent sous la forme de paramètres.

Le programme principal correspondant au fichier : **prog_princ.cbl**

```
program-id. appel-pg-PP.
working-storage section.
1 entier1 pic 99 value 0.
1 entier2 pic 99 value 0.
1 entier3 pic 999 value 0.

screen section.
1 a-plg-titre line 2 col 20 'Test sur entier'.
1 a-plg-res3.
      2 line 18 col 3 'Le resultat dans le pg principal somme : '.
      2 a-entier3 pic 999 from entier3.
1 a-plg-res4.
      2 line 19 col 3 'Le resultat dans le pg principal produit: '.
      2 a-entier4 pic 999 from entier3.
1 a-plg-res1.
      2 line 14 col 3 'le 1er nombre : '.
      2 a-entier1 pic 99 from entier1.
1 a-plg-res2.
      2 line 15 col 3 'le 2eme nombre : '.
      2 a-entier2 pic 99 from entier2.

procedure division.
display a-plg-titre
compute entier1=10
compute entier2=20
display a-plg-res1
display a-plg-res2
call 'SPPsomme'  using entier1 entier2 entier3 end-call   APPEL du sous-prog SPPsomme
display a-plg-res3
call 'SPPproduit' using entier1 entier2 entier3 end-call   APPEL du sous-prog SPPproduit
display a-plg-res4
goback.
end program appel-pg-PP.
```

Le sous-programme suivant correspondant au fichier : **SPPsomme.cbl**

```
program-id. somme.
linkage section.
1 val1 pic 99.
1 val2 pic 99.
1 val3 pic 999.

screen section.
1 a-plg-res.
2 line 12 col 3 'Le resultat somme dans sous prog : '.
2 a-val3 pic 999 from val3.

procedure division using val1 val2 val3.
compute val3= val1 + val2
display a-plg-res
goback.
end program somme.
```

Le sous-programme suivant correspondant au fichier : **SPPproduit.cbl**

```
program-id. produit.
linkage section.
1 val1 pic 99.
1 val2 pic 99.
1 val3 pic 999.

screen section.
1 a-plg-res.
      2 line 12 col 3 'Le resultat produit dans sous prog : '.
      2 a-val3 pic 999 from val3.

procedure division using val1 val2 val3.
compute val3= val1 * val2
display a-plg-res
goback.
end program produit.
```

2.4.2 Exemple1 version 2

Dans cette version les 2 sous-programmes apparaissent dans le même fichier calcul.cbl sous la forme de 2 points d'entrée.

Soit le programme principal suivant dans le fichier **pg-principal.cbl :**

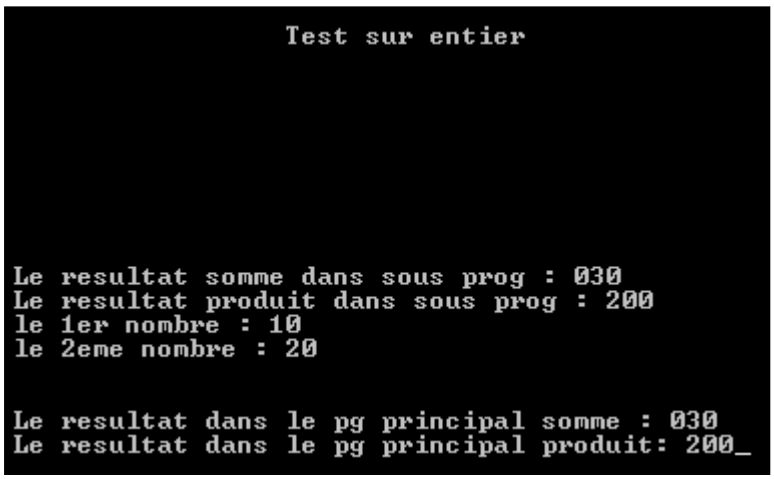
```
program-id. appel-pg-PP.
working-storage section.
1  entier1 pic 99 value 0.
1  entier2 pic 99 value 0.
1  entier3 pic 999 value 0.
screen section.
1 a-plg-titre line 2 col 20 'Test sur entier'.
1 a-plg-res3.
   2 line 18 col 3 'Le resultat dans le pg principal somme : '.
   2 a-entier3 pic 999 from entier3.
1 a-plg-res4.
   2 line 19 col 3 'Le resultat dans le pg principal produit: '.
   2 a-entier4 pic 999 from entier3.
1 a-plg-res1.
   2 line 14 col 3 'le 1er nombre : '.
   2 a-entier1 pic 99 from entier1.
1 a-plg-res2.
   2 line 15 col 3 'le 2eme nombre : '.
   2 a-entier2 pic 99 from entier2.

procedure division.
display  a-plg-titre
compute entier1=10
compute entier2=20
display a-plg-res1
display a-plg-res2
call 'calcul'          APPEL au POINT d'entrée PRINCIPAL
call 'somme' using   entier1 entier2 entier3 end-call      APPEL au POINT d'entrée SECONDAIRE
display a-plg-res3
call 'produit' using entier1 entier2 entier3 end-call      APPEL au POINT d'entrée SECONDAIRE
display a-plg-res4
 goback.
end program appel-pg-PP.
```

et son sous programme dans le fichier **calcul.cbl** :

```
program-id. calcul.
linkage section.
1 val1  pic 99.
1 val2  pic 99.
1 val3  pic 999.
screen section.
1 a-plg-res1.
   2 line 12 col 3 'Le resultat somme dans sous prog : '.
   2 a-val3 pic 999 from val3.
1 a-plg-res2.
   2 line 13 col 3 'Le resultat produit dans sous prog : '.
   2 a-val3 pic 999 from val3.
procedure division.
goback.

entry 'somme' using   val1  val2  val3.
  compute val3= val1 + val2
  display a-plg-res1
  goback.

entry 'produit' using  val1  val2 val3.
  compute val3= val1 * val2
  display a-plg-res2
  goback.

end program calcul.
```

POINT d'entrée SECONDAIRE 'somme'

POINT d'entrée SECONDAIRE 'produit'



2.4.3 Exemple 2

Soit le programme suivant correspondant au fichier : **prog_princ.cbl**

```
identification division.
program-id. appel.
data division.
working-storage section.
copy 'VarPer.txt'.
01 nom-ssprog pic x(06) value 'ssprog'.

*Pour savoir si l'utilisateur veut continuer la saisie
01 continuer   pic x value 'O'.
```

```
screen section.
01 a-plg-titre.
   02 blank screen.
   02 line 6 col 10 value 'Calcul p' & x'82' & 'rimetre'.

01 s-plg-largeur.
   02 line 9 col 1 value 'Entrez la largeur :'.
   02 s-largeur pic zz9 to largeur.

01 s-plg-longueur.
   02 line 10 col 1 value 'Entrez la longueur :'.
   02 s-longueur pic zz9 to longueur required.

01 a-plg-erreur.
   02 line 19 col 1.
   02 a-messErreur pic X(40) from mes-erreur.

01 a-plg-efface.
   02 line 19 col 1 blank line.

01 a-plg-cadres.
    02 line 17 col 8 value 'perimetre du rectangle :  '.
    02 col 40 value 'cm '.

01 a-plg-valres.
    02 line 17 col 30.
    02 a-perimetre pic zzzzz9 from perimetre.

01 s-plg-continuer.
   02 line 25 col 5 value "Voulez vous continuer ? (O/N) ".
   02 s-continuer pic x to continuer.

procedure division.

*   Boucle de saisie
*   ----------------------------

*   On boucle jusqu'a ce que continuer ne soit plus egal à O
    perform until ( function upper-case(continuer) not = 'O' )

*     Affichage du titre
*      ----------------------------
      display a-plg-titre


*      Saisie de la largeur
*      ----------------------------
      display s-plg-largeur
      accept s-plg-largeur

*      Saisie de la longueur
*      ----------------------------
      display s-plg-longueur
      accept s-plg-longueur

*     Calcul --> dans un sous programme
*     ---------------------------------
      call nom-ssprog using  varperim

     if  code-er = '00' then
*       Affichage des résultats
*       ----------------------
        display a-plg-cadres
```

```
          display a-plg-valres
          display a-plg-efface
        else
          display a-plg-erreur
        end-if


*       Test si l'utilisateur veut continuer la saisie
        display s-plg-continuer
        accept s-plg-continuer
      end-perform

      goback.
    End program appel.
```

Complété par le fichier : **ssprog.cbl**

```
    identification division.
    program-id. ssprog.
    working-storage section.
    linkage section.
    copy 'VarPer.txt'.

      procedure division using varperim.
        move '00'       to code-er
        move spaces     to mes-erreur
        move 0          to perimetre.
*     Contrôles
        if largeur > longueur
           MOVE    '01'    to code-er
           move 'la largeur est supérieure à la longueur'
                        to mes-erreur
        end-if
        if largeur = 0
           MOVE    '02'    to code-er
           move 'la largeur ne peut pas être nulle'
                        to mes-erreur
        end-if
        if longueur = 0
           MOVE    '02'    to code-er
           move 'la longueur ne peut pas être nulle'
                        to mes-erreur
        end-if

        if code-er = '00'
          compute perimetre = 2 * ( longueur + largeur)
        end-if
        goback.
```

Complété par le fichier : **VarPer.txt**

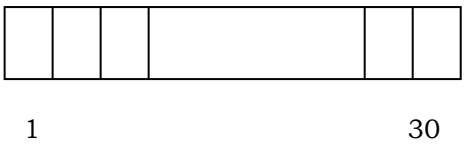
```
    01 varperim.
        02 var-ent.
                04 largeur       PIC 999 value 0.
                04 longueur      PIC 999 value 0.
        02 var-sort.
                04 code-er       PIC XX  value '00'.
                04 mes-erreur    PIC X(40) value spaces.
                04 perimetre     PIC 9(6) value 0.
```

CHAPITRE 6 : LES TABLEAUX

1 Exemples de déclaration de tableaux

A - Tableau à une dimension

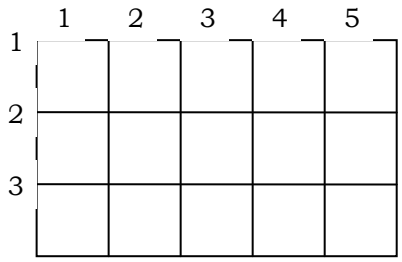
1 tab.
2 entier pic 9 occurs 30.



entier(i) accès au i^{eme} élement

B - Tableau à 2 dimensions

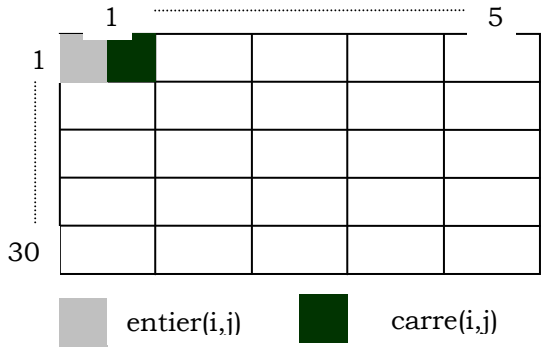
1 tab.
2 ligne occurs 3.
3 element pic 9 occurs 5.



Ligne(i) : accès à 1 ligne complète
Element(i,j) : accès à une cellule

C - Tableau avec structure

1 tab.
2 ligne-carre occurs 30.
3 element occurs 5.
4 entier pic 99.
4 carre pic 9(5).



2 Exemple détaillé n°4

La première version de ce programme permet d'afficher les nombres de 1 à n avec leurs carrés.

```
program-id. pg-tab-carre.
    data division.
    working-storage section.
    1 i pic 99.
    1 n pic 99.
    1 tab-carre.
       2 lign-carre  occurs 50.
          3 entier pic 99.
          3 carre pic 9(5).
    screen section.
    1 effacer-ecran blank screen.
    1 plg-titre.
       2 line 2 col 20 value 'Tableau des carrés de 1 à'.
       2 s-chp-n pic zz to n.
       2 line 4 col 2 value 'Entiers'.
       2 col 20 value 'Carres'.
    1 a-plg-tab-carre.
       2 line 6.
       2 a-plg-lign-carre occurs 50.
          3 a-chp-entier line + 2 col 5 pic zz from entier.
          3 a-chp-carre col 20 pic z(5) from carre.
    procedure division.
    debut.
       display effacer-ecran
       display plg-titre
       accept s-chp-n
       initialize tab-carre
       perform test after varying i from 1 by 1 until i=n
          compute entier(i)=i
          compute carre(i)=i*i
       end-perform
       display  a-plg-tab-carre
       goback.
       end program pg-tab-carre.
```

***Tab** est un tableau de 50 lignes chaque ligne est une structure composée de 2 éléments entier et carre.*

Tab est aussi une chaine de 50x7 caractères soit 350 caractères

Il faut reprendre les noms de la définition du tableau

Affichage du bloc complet

Entiers	Carres
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64

Cette 2ème version permet une meilleure gestion des écrans d'affichage (25 lignes et 80 caractères maxi).

```
program-id. pg-tab-carre.
    data division.
    working-storage section.
    1 i pic 99.
    1 n pic 99.
    1 lg pic 99 value 5.
    1 tab-carre.
       2 lign-carre  occurs 50.
          3 entier pic 99.
          3 carre pic 9(5).
    screen section.
    1 effacer-ecran blank screen.
    1 plg-titre.
       2 line 2 col 20 value 'Tableau des carres de 1 a'.
       2 s-chp-n pic zz to n.
       2 line 4 col 2 value 'Entiers'.
       2 col 20 value 'Carres'.
    1 a-plg-ligne.
       2 line lg.
       2 a-entier col 2 pic zz from entier(i).
       2 a-carre col 20 pic z(5) from carre(i).

    procedure division.
    debut.
       display effacer-ecran
       display plg-titre
       accept s-chp-n
       initialize tab-carre
       perform  varying i from 1 by 1 until i>n
          compute entier(i)=i
          compute carre(i)=i*i
          compute lg = lg + 1
          display  a-plg-ligne
       end-perform
       goback.
       end program pg-tab-carre.
```

Affichage d'une ligne à la fois

3 Déclaration d'un tableau

3.1 Déclaration d'un tableau *t* de taille fixe (decl-tab)

nb-niv *t* { id-type | descr-type } **occurs** { val-ent | const-ent } **.**

Il s'agit d'un tableau de val-ent (ou const-ent) éléments dont le type est défini en amont (id-type) ou décrit par descr-type.

3.2 Déclaration d'un tableau *t* de taille variable (decl-tab)

1 *st-t* **.**
2 *taille-t* **pic** 9(val-ent) **.**
2 *t* { id-type | descr-type } **occurs** [taille-min **to**] taille-max **depending** *taille-t* **.**

Le nombre effectif d'éléments (*taille-t*) est donc compris entre les valeurs taille-min (ou 1 par défaut) et taille-max.
Il est recommandé d'accompagner la déclaration du tableau de la déclaration de la variable (*taille-t*) qui doit indiquer la taille courante du tableau.

3.3 Déclaration d'un type tableau

Il suffit de préciser la clause typedef au niveau d'une structure (*st*-t) contenant uniquement la déclaration tableau (decl-tab).

```
1 st-t typedef.
      decl-tab
```

4 Les accès

4.1 Accès à un élément d'un tableau : variable indicée

t **(** var-ent [+ val-ent] **)**

4.2 Accès à un champ (id-chp) d'un élément de type structure d'un tableau

id-chp **(** var-ent [+ val-ent] **)**

où s'il est nécessaire de qualifier, pour lever toute ambiguïté,

id-chp [qualification] **of** *t* **(** var-ent [+ val-ent] **)**

4.3 Accès par référence modifiée

4.3.1 A partir du tableau dans son entier

t **(** rang-deb **:** long **)**

4.3.2 A partir d'une variable indicée

id-var **(** *j* **)** **(**rang-deb **:** long **)**

4.4 Accès au début d'un tableau

t ou *t* **(** 1 **)**

pour accéder au premier élément du tableau.

5 Affectation

Ce sont les instructions **move** et **compute** vues précédemment. Elles s'appliquent à tout ou partie d'un tableau, sachant que compute ne concerne dans sa partie droite que des éléments simples numériques, en particulier pas de référence modifiée.
Ne pas oublier qu'un tableau, une partie de tableau ou toute référence modifiée est considéré comme de type chaîne de caractères.

6 Initialisation

6.1 Statiques (lors de la déclaration du tableau)

6.1.1 Au niveau du tableau

```
1   st-t  value val-ch .
      decl-tab
```

Il y a affectation de val-ch à la chaîne de caractères que représente le tableau en mémoire

6.1.2 Au niveau de chaque (partie d') élément

```
1   st-t .

    decl-tab      } *>avec 'value val' sur une ou plusieurs lignes décrivant tout ou
                  *> partie de l'élément courant
```

La valeur val est affectée à toutes les occurrences de l'élément ou partie d'élément dans la description duquel elle apparaît.

6.1.3 A l'aide d'une redéfinition

```
1  st-val.
   _ _ _ _ _ _                    }
   _ _ _ _ _ _                    }  *>structure contenant une valeur de type chaîne
1  st-t redefines st-val.

     decl-tab
```

6.2 Dynamiques

Utilisation de l'instruction **move** [**all**] ou de l'instruction **initialize** sur le tableau dans son ensemble (initialisation de chaque partie d'élément à 0 ou espace selon son type) ou utilisation de ces mêmes instructions au niveau (d'une partie) de l'élément dans une itération.

7 Saisie et affichage d'un tableau

7.1 Elément par élément

Il suffit d'associer, dans sa déclaration, à un champ d'écran l'élément courant (ou la partie d'élément courant) à saisir ou afficher et d'itérer sur l'instruction de saisie ou d'affichage.

7.2 En bloc

Il suffit d'associer au tableau à saisir ou à afficher une plage contenant un tableau de champs d'écran de même dimension et d'associer, dans sa déclaration, à chaque champ d'écran le nom de l'élément (ou partie d'élément) de tableau à saisir ou à afficher.
Dés lors il suffit d'une seule instruction impérative de saisie ou d'affichage au niveau de la plage d'écran.

8 Tri des éléments d'un tableau

Le tri peut être effectué sur des arguments explicites cités dans l'instruction de tri ou sur des arguments implicites déclarés au moment de la déclaration du tableau.

8.1 Tri sur des arguments explicites

Avec id-var-ind élément courant *t* ou partie de *t*

sort *t* liste [ ] de [{ **ascending** / **descending** } id-var-ind [**duplicates**]]

Exemple :

```
1 tab.
  2 entier pic 9 occurs 30.
```

L'instruction sort entier ascending permet de trier le tableau par ordre croissant.

8.2 Tri sur des arguments implicites

Dans la clause **occurs** apparaissent les arguments de tri par défaut du tableau

occurs liste [ ] de [{ **ascending** / **descending** } id-var-ind]

et dans ce cas l'instruction de tri à l'appui des arguments implicites s'écrit :

sort *t* [**duplicates**]

9 Tableaux à plusieurs dimensions

En Cobol il s'agit de tableaux de tableaux et pour cela il suffit d'introduire la clause **occurs** dans la déclaration d'une partie d'élément de tableau (imbrication d' occurs). Introduire alors dans la variable indicée, accès à un élément, autant d'indices qu'il y a d' occurs imbriqués pour accéder à la déclaration de cet élément.

CHAPITRE 7 : LES FICHIERS

1 Introduction

Un fichier est un ensemble de données qui peut être vu :

- du point de vue informationnel, il s'agit du **fichier logique**
- du point de vue de sa mémorisation et de sa gestion par le SGF (Système de Gestion des Fichiers), il s'agit du **fichier physique**.

Un **fichier logique** est un ensemble d'enregistrement le plus souvent de même type (en général une structure).

Le SGF gère les **fichiers physiques** répertoriés dans un catalogue des fichiers.
Chaque élément du catalogue contient au moins le nom du fichier physique (nom-fichier.dat), l'identifiant du propriétaire, l'adresse de début du fichier. Un **fichier physique** est composé d'une suite d'enregistrements le plus souvent du même type.
Chaque **enregistrement**, unité d'accès aux données du fichier physique, est composé des données d'un article et éventuellement d'un entête d'enregistrement voire d'un marqueur de fin d'enregistrement.

2 Les fichiers séquentiels

Les enregistrements, tous de même taille, sont consécutifs sur le support et ne sont constitués que des données des articles (pas d'information système). Chaque enregistrement possède un unique enregistrement prédécesseur et un unique enregistrement successeur. Et cela dans l'ordre où les enregistrements ont été ajoutés dans le fichier.

L'**accès séquentiel** consiste, pour accéder à un enregistrement, à accéder à tous ses prédécesseurs dans l'ordre où ils sont dans le fichier.

En Cobol on dispose de trois organisations différentes de fichiers séquentiels intitulées

- **record sequential** : organisation séquentielle par **enregistrements** (données structurées), c'est, en général, l'organisation par défaut
- **line sequential** : organisation séquentielle par **lignes** qui est celle des fichiers texte des éditeurs sur PC
- **printer sequential** : organisation séquentielle directement adaptée à l'impression

2.1 Partie des déclarations

La déclaration d'un fichier est constituée :

- d'une **phrase select** (caractéristiques du fichier physique)
- d'une **déclaration de niveau fd** (caractéristiques du fichier logique)

2.1.1 Phrase select

S'effectue dans la file-control de l' input-output section.

Déclaration des caractéristiques du fichier physique ('nom-fichier.dat' ou *id-var-fs*) assigné au fichier logique (*id-fs*) et du mot d'état associé (*id-mot-etat*).

select [**optional**] *id-fs* **assign** { 'nom-fichier.dat' | *id-var-fs* }

organization [**record**] **sequential**
[**access sequential**]
[**status** *id-mot-etat*] **.**

2.1.2 Déclaration du fichier logique

S'effectue dans la **file section**.

```
fd      id-fs .
1       id-var-art .
        _ _ _ _ _ _
        *>déclaration de la variable du type
        *> de l'article du fichier (buffer du fichier)
        _ _ _ _ _ _
```

Cas de la déclaration d'un fichier logique à longueur variable d'article

```
fd   id-fs  record varying  [long-min]  to  [long-max]  [depending  id-var-long] .
```

la variable entière *id-var-long*, à déclarer, reçoit en lecture ou permet d'indiquer en écriture, la longueur de l'article en nombre de caractères.

2.2 Partie des instructions

2.2.1 Instructions globales sur un fichier

2.2.1.1 Instruction d'ouverture d'un fichier (création si nécessaire)

```
open   { input   }   id-fs   [reversed]
       { output  }   id-fs
       { i-o     }
       { extend  }
```

- l'ouverture en 'output' permet de remplacer l'ancien contenu du fichier.
- la clause 'optional' de la phrase select permet d'ouvrir directement en 'extend' un fichier non encore existant.
- la clause 'reversed' permet la lecture arrière du fichier

2.2.1.2 Instruction de fermeture du fichier

```
close  id-fs
```

2.2.2 Instructions élémentaires sur un fichier

*2.2.2.1 Instruction de lecture (séquentielle) d'un article (ouverture en **input** ou **i-o**)*

```
read   id-fs   [into  id-var]
               [end  inst-imper-1]
               [not end  inst-imper-2]
[end-read]
```

- la clause 'end' détecte l'adresse de fin de fichier lors d'une tentative de lecture.
- la clause 'into' affecte un exemplaire de l'article lu à *id-var*.

*2.2.2.2 Instruction d'écriture (séquentielle) d'un article (en **output** ou **extend**)*

write *id-var-art* [**from** { *id-var* / val-ch / const-ch }] [**end-write**]

*2.2.2.3 Instruction de réécriture (séquentielle) d'un article (en **i-o**)*

rewrite *id-var-art* [**from** { *id-var* / val-ch / const-ch }] [**end-rewrite**]

La réécriture doit être précédée d'une lecture réussie de l'article.

2.3 Exemple détaillé n°5

Le programme suivant concerne la consultation d'un fichier séquentiel. Il s'agit d'effectuer un traitement par le module *mod-trait-produit* de tous les produits figurant dans le fichier.

```
program-id. pg-consult-f-produit.

file-control.
   select f-produit assign 'f-prod.dat' organization record
   sequential.
```
← *Assignation du fichier logique au fichier physique et indication de l'organisation*

```
data division.

file section.
fd f-produit.
1 produit.
   2 code-produit pic x(7).
   2 designation pic x(40).
   2 prix-unitaire pic 999v99.
   2 stock pic 9(4).
```
← *Description d'un enregistrement*

```
working-storage section.
1 v-fin-f-produit pic x value 'N'.
88 fin-f-produit value 'O' false 'N'.
```
← *Déclaration d'un booléen permettant de gérer la fin de fichier*

```
procedure division.
debut.
   open input f-produit
   read f-produit end set fin-f-produit to true end-read
```
← *Ouverture du fichier*
← *Une lecture en avance*

```
perform until fin-f-produit
   perform mod-trait-produit
   read f-produit end set fin-f-produit to true end-read
end-perform
```
Boucle permettant de traiter tous les enregistrements jusquà la fin de fichier

```
close f-produit
```
← *fermeture du fichier*

```
goback.

mod-trait-produit.
.....
end program pg-consult-f-produit.
```
← *Description du module de traitement d'un enregistrement*

2.4 Complément sur le traitement des fichiers séquentiels

Les opérations que l'on peut avoir à mettre en œuvre, dans le cadre de sa mise à jour sont :

- l'adjonction d'articles, au milieu ou en fin de fichier
- la modification du contenu de certains articles
- la suppression d'articles

La réalisation de la mise à jour peut être faite selon deux modes, le mode **temps différé** ou le mode **interactif**.

Dans le mode temps différé, les modifications à apporter au fichier à mettre à jour sont cumulées pendant une certaine période, généralement dans un fichier séquentiel des mises à jour. Le programme temps différé qui prend en compte ces modifications s'exécute en l'absence de l'utilisateur.
Dans le mode interactif, le programme s'exécute en présence de l'utilisateur qui saisit, au fur et à mesure, les modifications à apporter au fichier à mettre à jour, modifications non préalablement enregistrées.

En terme d'efficacité, les fichiers séquentiels ne s'accommodent guère que du mode de mise à jour temps différé. On procède, toutefois en interactif, lors du chargement initial d'un fichier (version initiale du fichier de base qui sera à maintenir ultérieurement ou création d'un fichier des modifications) ou lors de l'extension, par la fin d'un fichier.

La mise à jour d'un fichier séquentiel peut être organisée de deux manières :

- **mise à jour sur place**

Les modifications sont faites sur la version courante du fichier à mettre à jour. Ce qui n'empêche pas de faire une copie préalable du fichier pour des raisons de sécurité : on procède à une sauvegarde de ce fichier. Dans ce type de mise à jour, les seules opérations possibles sur le fichier sont :

- la modification des contenus d'enregistrements existants : nécessite une ouverture en modification. Il faut accéder à l'enregistrement à modifier en passant par les prédécesseurs, puis réécrire dessus.
- l'adjonction d'enregistrements en fin de fichier : nécessite une ouverture en extension, d'où un positionnement en fin de fichier. Il suffit, alors, d'écrire, un à un, les enregistrements à ajouter.

- **mise à jour par recopie**

Dans ce cas on crée, par le biais du programme de mise à jour, une nouvelle version du fichier à mettre à jour : le fichier mis à jour.
Dès lors toutes les opérations élémentaires sont possibles. Outre la modification de contenu d'un enregistrement et l'adjonction d'un enregistrement en fin de fichier on peut :

- supprimer un enregistrement : il suffit de ne pas le recopier dans la nouvelle version du fichier
- insérer un enregistrement entre deux autres : il suffit de l'écrire au bon moment au cours de la recopie du fichier.

Remarque :
Il en ressort que les fichiers séquentiels ne sont pas adaptés au mode 'interactif' de mise à jour et que le mode 'temps différé' nécessite de trier le fichier des modifications sur les mêmes critères que le fichier à mettre à jour et de procéder, alors, à un parcours interclassé des deux fichiers.

2.5 Le tri d'un fichier séquentiel

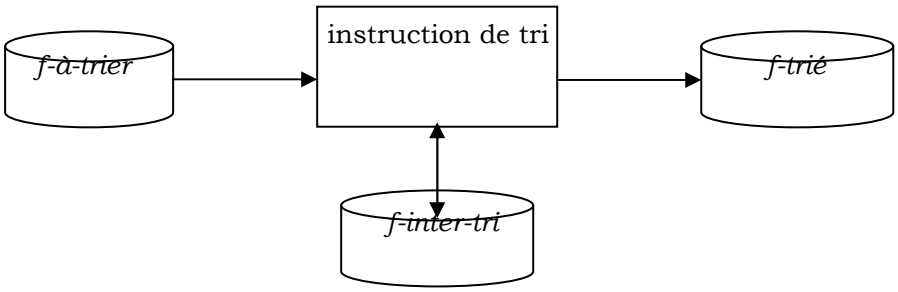
Il s'agit de classer entre eux les articles d'un fichier en fonction des valeurs d'un ou de plusieurs champs des articles désignés comme arguments du tri.
En Cobol une instruction (**sort**) permet d'effectuer le tri d'un fichier. Cette instruction met en œuvre trois fichiers:

-le fichier à trier soit *f-à-trier*

-un fichier **temporaire** intermédiaire soit *f-inter-tri*
-le fichier trié soit *f-trié*

Quant au tri il se déroule automatiquement en trois phases:

- chargement du fichier intermédiaire *f-inter-tri* avec les enregistrements du fichier à trier *f-à-trier*
- tri du fichier intermédiaire *f-inter-tri* sur les arguments indiqués avec le résultat du tri dans *f-inter-tri*
- chargement du fichier résultat *f-trié* avec les enregistrements ordonnés du fichier *f-inter-tri*



Dans le programme Cobol, les fichiers séquentiels *f-à-trier* et *f-trié* se déclarent comme on l'a vu précédemment.
Le **fichier temporaire** fait l'objet d'une phrase '**select**' minimum,

```
select  f-inter-tri  assign  'f-inter-tri'.
```

et sa déclaration doit contenir au niveau de l'article les champs (id-ch-argj) qui sont arguments du tri.

La **déclaration du fichier intermédiaire** s'écrit donc,

```
 sd  f-inter-tri .
enr-inter-tri.
        2- - - - - -
        2 id-chp-arg1- - - - -
        2- - - - - -
        2 id-chp-arg j- - - - -
2 - - - - - -
```

L'**instruction de tri d'un fichier** s'écrit:

```
sort  f-inter-tri    liste [ ] de [ { ascending } id-chp-argj - --[duplicates] ]
                                    { descending }
      using  f-à-trier
      giving  f-trié
```

La clause '**using**' assure le chargement de f-inter-tri avec les enregistrements de f-à-trier, puis le tri de f-inter-tri s'effectue et enfin la clause '**giving**' transfère les enregistrements classés de f-inter-tri dans f-trié.

Une variante de l'instruction de tri consiste à mettre en œuvre à la place des clauses '**using**' et/ou '**giving**' des procédures pour assurer respectivement le chargement et/ou le 'déchargement' du fichier intermédiaire du tri.

L'**instruction de tri d'un fichier** s'écrit alors:

sort f-inter-tri liste [ ] de [{ **ascending** / **descending** } id-chp-argj - --[**duplicates**]]

input procedure id-mod-entrée [**thru** id-mod-fin-entrée]
output procedure id-mod-sortie [**thru** id-mod-fin-sortie]

Dans la procédure d'entrée du tri, constituée d'un ou de plusieurs paragraphes ou sections, le fichier intermédiaire est automatiquement, à l'entrée de la procédure, ouvert en écriture et fermé à la sortie.

L'**instruction d'écriture dans le fichier intermédiaire** s'écrit,

release enr-inter-tri [**from** id-var-ch] *>instruction équivalente à un 'write'

Dans la procédure de sortie du tri, constituée d'un ou plusieurs paragraphes ou sections et exécutée après la deuxième phase du tri, le fichier intermédiaire est automatiquement ouvert en lecture au début et fermé à la fin.
Il s'agit de lire un à un les articles du fichier intermédiaire pour pouvoir les traiter.

L'**instruction de lecture spécifique d'un fichier intermédiaire** de tri s'écrit,

return f-inter-tri [**into** id-var-ch]
[**end** inst-imper1] [**not end** inst-imper2] *>instruction équivalente à un 'read'
end-return

A l'instruction '**sort**' est associé automatiquement, sans déclaration, un registre d'état du tri '**sort-return**' positionné,
-à 0 si le tri s'est bien déroulé
-à 16 si le tri a échoué
Il est possible de positionner impérativement (set sort-return to 16) ce registre à la valeur 16 dans les procédures d'entrée et/ou de sortie du tri pour interrompre immédiatement le tri à la rencontre de la prochaine exécution de l'instruction 'release' ou 'return'.

Exemple : Tri d'un fichier séquentiel

```
program-id. pg-tri-f-produit.
file-control.
 select f-produit assign 'f-prod.dat' organization
 record sequential
 status mot-etat-f-produit.
 select n-f-produit assign 'n-f-prod.dat' organization
 record sequential.
 select f-inter-produit assign 'f-inter-produit'.
data division.
file section.
fd f-produit.
1 produit typedef.
  2 code-produit pic x(7).
  2 designation pic x(40).
  2 prix-unitaire pic 999v99.
  2 stock pic 9(4).
1 enr-produit produit.
fd n-f-produit.
1 n-enr-produit produit.
```

```
sd f-inter-produit.
1 enr-inter-produit.
  2  code-produit pic x(7).
  2            pic x(49).

working-storage section.
1 v-fin-f-produit pic x value 'n'.
88 fin-f-produit value 'o' false 'n'.
1 mot-etat-f-produit pic xx.
88 f-produit-inexistant value '35'.
88 f-produit-vide value '10'.
1  v-fin-f-inter-produit pic x value 'n'.
88 fin-f-inter-produit value 'o' false 'n'.

procedure division.
sort f-inter-produit ascending code-produit of enr-inter-produit
input procedure mod-avant-tri
output procedure mod-apres-tri
goback.

mod-avant-tri.
open input f-produit.
if f-produit-inexistant set sort-return to 16 end-if
read f-produit end set fin-f-produit to true end-read
if f-produit-vide set sort-return to 16 end-if

perform until fin-f-produit
release enr-inter-produit from enr-produit
read f-produit end set fin-f-produit to true end-read
end-perform
close f-produit.

mod-apres-tri.
open output n-f-produit
return f-inter-produit into n-enr-produit end
set fin-f-inter-produit to true end-return
perform until fin-f-inter-produit
   write n-enr-produit
   return f-inter-produit into n-enr-produit end
   set fin-f-inter-produit to true end-return
end-perform
close n-f-produit.
end program pg-tri-f-produit.
```

2.6 La fusion de plusieurs fichiers séquentiels

La fusion de plusieurs fichiers (id-fichierk) tous supposés **triés dans le même ordre sur le même ensemble d'arguments** (id-chp-argj) consiste en l'interclassement de leurs articles dans un seul fichier résultat lui-même finalement classé dans le même ordre.

L'**instruction de fusion de plusieurs fichiers** s'écrit:

merge f-inter-fusion liste [ ] de [{ **ascending** / **descending** } id-chp-argj]

using liste [ ] de id-fichierk

{ **giving** liste [ ] de id-fichier-resn / **output procedure** id-mod-sortie [**thru** id-mod-fin-sortie] }

Le résultat de la fusion peut être copié dans plusieurs fichiers (id-fichier-resn).
Comme dans l'instruction '**sort**' la procédure de sortie permet d'accéder un à un aux articles fusionnés dans le fichier intermédiaire f-inter-fusion grâce à l'instruction '**return**'.
Le résultat de la fusion est chargé dans chacun des fichiers de la liste '**giving**' ou pris en compte dans la procédure de sortie par des '**return**' .

2.7 Création d'un fichier de données sous NetExpress

La création d'un fichier de données comporte deux étapes :

A - Allocation du fichier peut se faire soit directement sous NetExpress soit indirectement par un programme cobol.
B- Chargement des données dans le fichier peut s'effectuer directement sous NetExpress, à l'aide d'un programme cobol ou plus simplement par conversion d'un fichier déjà existant.

La démarche suivante permet de créer un fichier sous NetExpress :

1) Allocation du fichier :
   Fichier →New→Data File → Définir la longueur de l'enregistrement et la position de la clé primaire
   Donner un nom au fichier : **fi-test.dat**

2) Créer un programme cobol dans un projet contenant la description d'un enregistrement en accès séquentiel ou direct :

Exemple de programme :

```
program-id. pgtest.
Select fi-test assign ' fi-test.dat ' organization indexed record key cle access dynamic.
fd fi-test.
1 enr_test.
        2 cle pic x(5).
        2 lib pic x(3).
end program pgtest.
```

3) Compiler le projet.
4) Sélectionner le projet contenant la structure et faire clic droit, choisir ***New record layout***. L'objectif est de créer un fichier '.str' à partir de la structure définie dans la programme pour avoir un modèle d'enregistrement.
5) Enregistrer le fichier .str et fermer.
6) Ajouter le fichier .dat au projet.
7) Ouvrir le fichier .dat.
8) Faire ***Fichier →DataTools→New record layout*** et choisir le bon fichier .str .
9) Sur la partie droite faire clic droit et choisir ***insert indexed record*** (si le fichier est direct).
10) Insérer la clé et la suite de l'enregistrement de manière assistée.

Concernant la conversion, il suffit de choisir ***Fichier →DataTools→Convert*** et de saisir les informations du fichier à convertir et du fichier à créer.

3 Les fichiers directs

On distingue les fichiers relatifs et les fichiers indexés.

3.1 Le fichier relatif

Un fichier relatif est constitué d'une suite d'emplacements 'consécutifs' en mémoire externe.
Chaque emplacement a une adresse relative qui correspond à son rang (de 1 à n) dans le fichier.
La clé d'accès du fichier est le rang des emplacements dans le fichier.
Chaque emplacement est susceptible de contenir un enregistrement du fichier. Il existe un bit de chargement par emplacement. Un emplacement peut être **'chargé',** un enregistrement y a été

écrit ou '**non chargé**', aucun enregistrement n'y a été écrit ou son contenu a fait l'objet d'un effacement.

L'accès dans un fichier relatif, géré par le **SGF** peut être séquentiel ou direct :

- l***'accès séquentiel*** est assuré dans l'ordre croissant des valeurs de la clé d'accès du fichier (rangs des emplacements) aux seuls emplacements chargés, s'est à dire automatiquement aux seuls enregistrements existants dans le fichier.
- ***l'accès direct*** à un emplacement pour une valeur donnée de la clé d'accès du fichier (un rang donné) et donc accès à l'enregistrement contenu s'il en existe un.

3.2 Le fichier indexé

Dans un fichier indexé, la clé d'accès du fichier est constituée d'un ou de plusieurs champs, simples ou structurés, de l'article du fichier. Un fichier indexé est, en fait, constitué de deux fichiers physiques :

- le fichier des données, les enregistrements effectifs du fichier.
- le fichier 'index' dont chaque élément est constitué :
  - d'une valeur de la clé d'accès (entrée de la table représentée par le fichier indexé)
  - de l'adresse, dans le fichier de données, de l'article associé à cette valeur de clé d'accès

Remarques :
La clé d'accès d'un fichier indexé est de type quelconque.
Le fichier 'index' est ordonné par valeurs croissantes de la clé et généralement partitionné.
Contrairement au fichier relatif, il n'y a pas d'emplacements prédéfinis, seuls les articles correspondant à des valeurs effectives de la clé d'accès (entrées de la table) sont mémorisés.

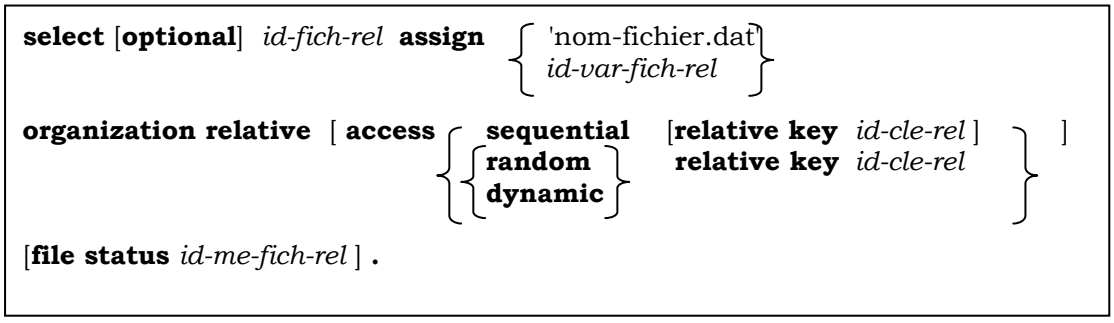
Le **SGF** gère deux types d'accès dans un fichier indexé :

- ***l'accès séquentiel*** : C'est l'accès aux enregistrements dans l'ordre croissant des valeurs de la clé d'accès du fichier. Cet accès implique la consultation séquentielle automatique du fichier index par le 'SGF'.
- ***l'accès direct*** : C'est l'accès **direct** à un enregistrement, étant donnée la valeur de clé d'accès qui le caractérise. Cela implique la consultation 'directe' automatique du fichier 'index' par le 'SGF' pour obtenir l'adresse de l'article ayant cette valeur de clé.

Les opérations sont identiques pour les deux types de fichiers directs, à ceci près que pour un **fichier relatif** la clé d'accès est représentée par une **variable entière** associée au fichier, au moment de sa déclaration dans un programme, alors que pour un **fichier indexé**, c'est un ou plusieurs **champs de l'article** qui jouent le rôle de clé d'accès.

3.3 Déclarations des fichiers directs

3.3.1 Déclaration d'un fichier relatif



La clé du fichier, souvent appelée clé relative (*id-cle-rel*) doit être déclarée comme variable entière dans le programme.

Si, dans le cas de l'accès séquentiel, une clé (*id-cle-rel*) est associée au fichier le SGF la valorise avec le rang de l'article courant pendant le parcours séquentiel du fichier.
En accès direct (**random**) ou mixte (**dynamic**) il est nécessaire d'associer une clé (*id-cle-rel*) au fichier pour réaliser les accès directs.

La déclaration de niveau '**fd**' en file section : est semblable à celle d'un fichier séquentiel. A noter toutefois que la clé relative (*id-cle-rel*) ne peut pas être un champ de l'article du fichier.

3.3.2 Déclaration d'un fichier indexé

select [**optional**] *id-fich-ind* **assign** { 'nom-fichier.dat' | *id-var-fich-ind* }

organization indexed **access** { **sequential** | **random** | **dynamic** }

record key { *id-cle-ind* | *id-cle-comp* = liste [] de *id-chp-art* }

[**file status** *id-me-fich-ind*] **.**

La clé du fichier (souvent appelée clé indexée) doit être un **champ de son type d'article** (*id-cle-ind*) ou une clé composée de la concaténation d'un certain nombre de **champs (non consécutifs) de ce type d'article (*id-cle-comp*).**

La déclaration de niveau '**fd**' en file section : est semblable à celle d'un fichier séquentiel. A noter toutefois que la clé du fichier doit faire partie du type d'article du fichier.

3.3.3 Instructions sur les fichiers directs

Les opérations élémentaires sont de type séquentiel ou direct, leur mise en œuvre dépend de l'accès choisi pour le fichier et du type de période d'accessibilité.
Les instructions sont les mêmes pour un fichier **relatif** ou un fichier **indexé**, à ceci près qu'elles mettent en œuvre dans un cas une clé **relative** dans l'autre cas une clé **indexée**.

3.3.3.1 Définition d'une période d'accessibilité

Entre l'ouverture et la fermeture du fichier

open { **input** | **output** | **i-o** } *id-fich-dir*

close *id-fich-dir*

3.3.3.2 Instructions élémentaires

A - Lecture séquentielle d'un article

read *id-fich-dir* [{ **next** | **previous** }] [**into** *id-var*]

[**end** inst-imper1] [**not end** inst-imper2]
end-read

Contextes de mise en oeuvre:

'access sequential' ou 'access dynamic'
'open input' ou 'open i-o'

Les options 'next' (lecture séquentielle en avant) et 'previous' (lecture séquentielle en arrière) sont restreintes à 'access dynamic'.

B - Lecture directe d'un article

read *id-fich-dir* [**into** *id-var*]
[**invalid** inst-imper1] [**not invalid** inst-imper2]
end-read

Avant d'exécuter une lecture directe la clé du fichier (*id-cl-rel, id-cle-ind* ou *id-cle-comp*), déclarée dans la phrase 'select', doit être valorisée.
Contextes de mise en œuvre:

'access random' ou 'access dynamic'
'open input' ou 'open i-o'

C- Ecriture séquentielle ou directe d'un article

write *id-article* [**from** *id-var*]
[**invalid** inst-imper1] [**not invalid** inst-imper2]
end-write

Contextes de mise en oeuvre:

- *écriture séquentielle*
  - 'open output'
  - 'access sequential'
  - Pour un fichier relatif le SGF charge automatiquement l'emplacement suivant dans le fichier.
  - Pour un fichier indexé le programme doit préciser avant chaque exécution du 'write' la valeur de la clé pour l'article à écrire
  - L'option 'invalid' sert pour un fichier indexé à prendre le contrôle s'il n'y a pas respect de l'ordre croissant des valeurs de la clé à chaque écriture.

- *écriture directe*
  - 'open output' ou 'open i-o'
  - 'access random' ou 'access dynamic'
  - L'option 'invalid' permet de prendre le contrôle si
    -pour un fichier relatif l'emplacement concerné est déjà chargé
    -pour un fichier indexé il existe déjà un article ayant même valeur de clé

D- Recherche (de l'adresse donc de l'existence) d'un article

start *id-fich-dir* [**key** { = | < | <= | > | >= } *id-clé*[**size** { *id-var-ent* | val-ent }]
[**invalid** inst-imper1] [**not invalid** inst-imper2]
end-start

La recherche est effectuée par rapport à la valeur de clé indiquée dans *id-clé*.

Contextes de mise en oeuvre:

- 'open input' ou 'open i-o'
- 'access sequential' ou 'access dynamic'

- Pour un fichier relatif, la clé *id-clé* est la clé relative (*id-cle-rel*).

- Pour un fichier indexé, la clé *id-clé* est la clé indexée (*id-clé-ind* ou *id-clé-comp*) ou toute partie de la clé indexée commençant par son premier caractère (cas où on ne connaît que les premiers caractères de la valeur de clé de l'article recherché).
- Cette première partie de clé peut être indiquée à l'aide du nom d'un champ début de la clé, à l'aide d'une référence modifiée ou grâce à l'option '**size**' en indiquant le nombre de caractères à prendre en compte en partant du début de la clé.

Remarque:
L'instruction 'start' ne délivre aucune donnée mais, si elle s'exécute correctement, place un pointeur à une adresse précise d'un article dans le fichier.

E-réécriture séquentielle ou directe d'un article

rewrite *id-article* [**from** *id-var*]
[**invalid** inst-imper1] [**not invalid** inst-imper2]
end-rewrite

Contextes de mise en oeuvre:

- 'open i-o'
- Dans le cas de 'access sequential' l'instruction 'rewrite' doit être précédée d'une lecture réussie et la valeur de la clé doit être la même que pour la lecture, de plus l'option '(not) invalid' n'est pas autorisée.
- Dans le cas de ''access random' ou 'access dynamic' la réécriture est directe (pas de lecture préalable indispensable) et l'option '(not) invalid' permet de prendre le contrôle en cas d'échec de la réécriture.

F- Séquentiel ou direct d'un article

delete *id-fich-dir*
[**invalid** inst-imper1] [**not invalid** inst-imper2]
end-delete

Contextes de mise en œuvre:

- 'open i-o'
- Dans le cas de 'access sequential' l'instruction 'delete' doit être précédée d'une lecture réussie et la valeur de la clé doit être la même que pour la lecture, de plus l'option '(not) invalid' n'est pas autorisée.
- Dans le cas de ''access random' ou 'access dynamic' l'effacement est direct (pas de lecture préalable indispensable) et l'option '(not) invalid' permet de prendre le contrôle en cas d'échec de l'effacement.

G-Cas particulier d'opération globale sur un fichier : effacement du fichier
Il est possible d'effacer 'physiquement' un (ou plusieurs) fichier(s) du catalogue des fichiers à condition qu'au moment de son effacement le soit fermé et non verrouillé.

delete file liste [ ] de *id-fich*

3.4 Les fichiers indexés multi-tables (ou multi-clés)

3.4.1 Déclaration d'un fichier indexé multi-clés

Un **fichier indexé** possède nécessairement une clé primaire (**record key**) simple (*id-clé-ind*) ou composée (*id-clé-comp*) mais peut aussi posséder des clés secondaires (**alternate** key) simples ou composées pour chacune desquelles est géré également automatiquement un fichier index spécifique.
La déclaration du fichier, inchangée au niveau '**fd**', déclare toutes les clés au niveau de la 'phrase **select**'.

select [**optional**] *id-fich-ind* **assign . . . .**
organization indexed **access . . . .**

record { *id-cle-ind* | *id-cle-comp* = liste [] de *id-chp-article* }

liste [] de [**alternate** { *id-cle-sec* | *id-cle-cp-sec* = liste [] de *id-chp-article* } [**duplicates**]]

file status *id-me-fich-ind*.

Les clés secondaires simples (*id-cle-sec*) ou composées (*id-cle-cp-sec*) peuvent être utilisées comme la clé primaire et de plus peuvent être ambiguës à condition d'ajouter l'option '**duplicates**'.
Pendant une période d'accessibilité au fichier multi-clés **la clé courante**, par défaut la clé primaire (*id-clé-ind* ou *id-clé-comp*) peut être changée grâce aux instructions '**read**' ou '**start**'.

Exemple :

```
select fpret assign ' Pret.dat' organization indexed
record key is clef = dateP, RefS, CodeE
alternate key RefS duplicates
alternate key CodeE duplicates
access dynamic.

Fd fpret.
1 enrEmprunt.
   2 DateP pic 9(8).
   2 RefS pic x(5).
   2 CodeE pic x(30).
   2 DateRetour pic 9(8).
```

3.4.2 Instructions et clés secondaires

A- lecture directe d'un article

read *id-fich-ind* [**into** *id-var*] **key** { *id-clé-ind* | *id-clé-comp* | *id-clé-sec* | *id-clé-cp-sec* }
[**invalid** inst-imper1] [**not invalid** inst-imper2]
end-read

Contextes de mise en oeuvre:
- 'access random' ou 'access dynamic'
- 'open input' ou 'open i-o'

Cette instruction de lecture directe indique la clé primaire ou secondaire à mettre en œuvre comme **nouvelle clé courante**, à partir de cet instant, pour toutes les opérations élémentaires sur le fichier et assure automatiquement l'accès au fichier index correspondant.

B-Recherche (de l'adresse ou de l'existence) d'un article

start *id-fich-ind* **key** [**not**] { **=** | **>** | **<** | **>=** | **<=** } { *id-clé-ind* | *id-clé-comp* | *id-clé-sec* | *id-clé-cp-sec* } [**size** { *id-var-ent* | val-ent }]

[**invalid** inst-imper1] [**not invalid** inst-imper2]
end-start

Contextes de mise en oeuvre:

- 'access sequential' ou 'access dynamic'
- 'open input' ou 'open i-o'

L'instruction de recherche de l'adresse ou de l'existence d'un article satisfaisant à une condition sur la valeur de sa clé offre donc aussi, comme l'instruction de lecture directe, l'occasion de préciser en même temps, quelle est la **nouvelle clé courante** (primaire ou secondaire) pour le fichier.

C-Lecture séquentielle d'un article

read *id-fich-ind* { **next** | **previous** } [**into** *id-var*]

[**end** inst-imper1] [**not end** inst-imper2]
end-read

Contextes de mise en oeuvre:

- 'access sequential' ou 'access dynamic'
- 'open input' ou 'open i-o'

L'option '**end**' permet de prendre le contrôle en cas de dépassement de l'adresse de fin de fichier en lecture avant (next) ou de dépassement de l'adresse de début de fichier en cas de lecture arrière (previous).

Remarque :
Les lectures séquentielles peuvent apparaître à la suite d'une instruction de recherche (start) ou d'une lecture directe (read).

4 Exemple détaillé n°6

L'exemple considéré concerne la mise à jour d'un fichier direct de produits **f-prod** à partir d'un fichier séquentiel **foper** des opérations de la journée.
Les opérations à mettre en œuvre sont : la création d'un produit nouveau, la suppression d'un produit et la modification d'un produit. Les messages d'erreurs éventuels sont stockés dans un fichier séquentiel **fmessage.**

```
program-id. maj-dir-dif-f-prod.
select fprod assign 'fprod.dat'
organization indexed record key refprod
access random.
select foper assign 'foper.dat'
organization record sequential.
select fmessage assign 'fmessage.dat'
organization record sequential.
```

Déclaration et assignation des fichiers

```
fd fprod.
1 prod.
        2 refprod      pic x(5).
        2 designprod   pic x(20).
        2 prixprod     pic 9(4)v99.
        2 qteprod      pic 9(4).
        2 fourprod     pic x(20).
```

Déclaration de la structure d'un enregistrement

```
fd foper.
 1 oper.
        2 codoper          pic x.
        2 refprodoper      pic x(5).
        2 prodacreer.
                3 designprodoper   pic x(20).
                3 qteprodoper      pic 9(4).
                3 prixprodoper     pic 9(4)v99.
                3 fourprodoper     pic x(20).
        2 prodamodif   redefines  prodacreer.
                3 qteprodoper   pic 9(4).
                3 texte         pic x(46).
```

Exemple de redéfinition de type

```
fd fmessage.
 1 lemessage.
        2 libmess      pic x(50).
        2 refprodmess  pic x(5).
working-storage section.
1   pic x  value 'n'.
88 finfoper value 'o' false 'n'.
1   pic x.
88 prodexiste value'o' false 'n'.
1 pic x.
88 ercreaprod value 'o' false 'n'.
1 pic x.
 88 ersupprod value 'o' false 'n'.
 1 pic x.
88 ermodifprod value 'o' false 'n'.
```

```
procedure division.
 debut.

 open i-o fprod input foper output fmessage

 read foper end set finfoper to true end-read

 perform until finfoper
    perform mod-oper
   read foper end set finfoper to true end-read
end-perform

 close fprod foper fmessage

 goback.
```

Ouverture des 3 fichiers
Fprod en lecture/écriture
Foper en lecture
Fmessage en écriture

Lecture du 1er enregistrement de Foper
et positionnement du booléen à vrai
en cas de fin de fichier

Boucle de traitement de tous les enregistrements
de foper jusqu'à la fin du fichier foper :
exécuter le module mod-oper
lecture de l'enregistrement suivant

Fermeture des 3 fichiers

```
mod-oper.

   evaluate true
   when codoper = 'c' perform mod-creaprod
   when codoper = 's' perform mod-supprod
   when codoper = 'm' perform mod-modifprod
   when other         perform mod-erreurcodoper

  end-evaluate.
```

Module principal

```
mod-creaprod.                *>si codoper = 'c'

move refprodoper to refprod
read fprod invalid set prodexiste to false
             not invalid set prodexiste to true
end-read

if prodexiste
      then   move "creation a tort" to libmess
             move refprodoper to refprodmess
             write lemessage
      else
          move refprodoper to refprod
          move designprodoper to designprod
          move prixprodoper to prixprod
          move qteprodoper of prodacreer to qteprod
          move fourprodoper to fourprod
          write prod invalid set ercreaprod to true
                     not invalid set ercreaprod to false
          end-write
    if ercreaprod
       then    move "erreur d'ecriture dans fprod" to libmess
               move refprodoper to refprodmess
               write lemessage
    end-if
end-if.
```

Lecture de frop pour verifier l'existence ou non du produit à créer

Module de création d'un nouvel enregistrement de fprod à partir de foper

Ecriture dans fprod

Ecriture du message d'erreur

```
mod-supprod.                    *>si codoper = 's'
move refprodoper to refprod
read fprod  invalid set prodexiste to false
not invalid set prodexiste to true  end-read
if prodexiste   then
    delete fprod invalid set ersupprod to true
                    not invalid set ersupprod to false
    end-delete
    if ersupprod
        then move "erreur de suppression dans fprod"   to libmess
             move refprodoper to refprodmess
             write lemessage
    end-if
 else   move "suppression à tort" to libmess
        move refprodoper to refprod
        write lemessage
end-if.
```

Module de supression

```
mod-modifprod.                  *>si codoper = 'm'
move refprodoper to refprod
read fprod  invalid set prodexiste to false
            not invalid set prodexiste to true
end-read
if prodexiste  then
        compute qteprod = qteprod + qteprodoper of prodamodif
        rewrite prod invalid set ermodifprod to true
                    not invalid set ermodifprod to false
        end-rewrite
       if ermodifprod then
          move "erreur de reecriture dans fprod" to libmess
          move refprodoper to refprodmess
          write lemessage
        end-if
 else
          move "modification a tort" to libmess
          move refprodoper to refprodmess
          write lemessage
end-if.     mod-erreurcodoper.
     *>si codoper <>'c'et 's' et 'm'
     move "codoper errone : operation non traitee" to libmess
     move refprodoper to refprodmess
     write lemessage.
end program maj-dir-dif-f-prod.
```

Module de modification

5 Exemple détaillé n°7

Programme permettant d'afficher tous les véhicules de marque Peugeot 207 dans un fichier avec une clé primaire et une clé secondaire dupliquée.

```
Program-id. pgvoit
select F-VOITURES
   assign to "voitures.dat"
   organization indexed  access random
   record key PLAQUE
   alternate record key MARQUE duplicates.

 file section.
      fd F-VOITURES.
      1 ENR-VOITURE.
        2 PLAQUE pic X(10).
        2 MARQUE pic X(10).
        2 MODELE pic X(10).
        2 COULEUR pic X(10).

Working storage section.
1 pic x value 'o'.
88 FIN-FICHIER value 'o' false 'n'.

Procedure division.
Open input F-VOITURES
move 'PEUGEOT' to MARQUE
start F-VOITURES key = MARQUE
```

```
    invalid key  display 'Pas de Peugeot'
    not invalid key  set  FIN-FICHIER  to FALSE
        perform until FIN-FICHIER
        read F-VOITURES next
               end   set  FIN-FICHIER to TRUE
               not  end
                 if MARQUE ='PEUGEOT'
                       and MODELE = '207'
                    then   display PLAQUE, MODELE
                  end-if
               end-read
           end-perform
end-start
close F-VOITURES
end program pgvoit.
```

ANNEXES

LISTE DES FONCTIONS PREDEFINIES

The "arguments" column defines argument type and the "type" column defines the type of the function, as follows:

Alph means alphabetic
Anum means alphanumeric
Int means integer
MFNat means national
Num means numeric

Function-Name	Arguments	Type	Value Returned
MFABS	Int1 or Num1	Depends upon arguments	The absolute value of argument
ACOS	Num1	Num	Arcosine of Num1
ANNUITY	Num1, Int2	Num	Ratio of annuity paid for Int2 periods at interest of Num1 to initial investment of one
ASIN	Num1	Num	Arcsine of Num1
ATAN	Num1	Num	Arctangent of Num1
CHAR	Int1	Anum	Character in position Int1 of the alphanumeric program collating sequence
MFCHAR-NATIONAL	Int1	Nat	Character in position Int1 of the national program collating sequence
COS	Num1	Num	Cosine of Num1
CURRENT-DATE	None	Anum	Current date and time and difference from Greenwich Mean Time
DATE-OF-INTEGER	Int1	Int	Standard date equivalent (YYYYMMDD) of integer date
DATE-TO-YYMMDD	Int1	Int	Argument-1 converted Int2 from YYMMDD to YYYYMMDD based on the value of argument-2
DAY-OF-INTEGER	Int1	Int	Julian date equivalent (YYYYDDD) of integer date
DAY-TO-YYYYDDD	Int1	Int	Argument-1 converted Int2 from YYDDD to YYYYDDD based on the value of argument-2
MFDISPLAY-OF	Nat1, Anum1	Anum	Usage display representation of argument Nat1
MFE	None	Num	The value of *e*, the natural base
MFEXP	Num1	Num	*e* raised to the power Num1
MFEXP10	Num1	Num	10 raised to the power Num1
FACTORIAL	Int1	Int	Factorial of Int1
MFFRACTION-PART	Num1	Num	Fraction part of Num1
INTEGER	Num1	Int	The greatest integer not greater than Num1
INTEGER-OF-DATE	Int1	Int	The integer date equivalent of standard date (YYYYMMDD)
INTEGER-OF-DAY	Int1	Int	The integer date equivalent of Julian date (YYYYDDD)
INTEGER-PART	Num1	Int	Integer part of Num1

LENGTH	Alph1 or Anum1 or (MF)Nat1 or Num1	Int	Length of argument in number of character positions
(MF)LENGTH-AN	Alph1 or Anum1 or Int1 or Nat1 or Num1	Int	Length of argument in number of alphanumeric character positions
LOG	Num1	Num	Natural logarithm of Num1
LOG10	Num1	Num	Logarithm to base 10 of Num1
LOWER-CASE	Alph1 or Anum1 or (MF)Nat1	Depends upon argument	All letters in the argument are set to lowercase
MAX	Alph1 ... or Anum1 ... or Int1 ... or (MF)Nat1 ... or Num1 ...	Depends upon arguments*	Value of maximum argument
MEAN	Num1 ...	Num	Arithmetic mean of arguments
MEDIAN	Num1 ...	Num	Median of arguments
MIDRANGE	Num1 ...	Num	Mean of minimum and maximum arguments
MIN	Alph1 ... or Anum1 ... or Int1 ... or (MF)Nat1 ... or Num1 ...	Depends upon arguments*	Value of minimum argument
MOD	Int1, Int2	Int	Int1 modulo Int2
(MF)NATIONAL-OF	Anum1, (MF)Nat1	Nat	Usage national representation of argument Anum1
NUMVAL	Anum1 or (MF)Nat1	Num	Numeric value of simple numeric string
NUMVAL-C	Anum1, Anum2, or (MF)Nat1, Nat2	Num	Numeric value of numeric string with optional commas and currency sign
ORD	Alph1 or Anum1 or (MF)Nat1	Int	Ordinal position of the argument in collating sequence
ORD-MAX	Alph1 ... or Anum1 ... or (MF)Nat1 ... or Num1	Int	Ordinal position of maximum argument
ORD-MIN	Alph1 ... or Anum1 ... or (MF)Nat1 ... or Num1	Int	Ordinal position of minimum argument
(MF)PI	None	Num	Value of
PRESENT-VALUE	Num1 Num2 ...	Num	Present value of a series of future period-end amounts, Num2, at a discount rate of Num1
RANDOM	Int1	Num	Random number
RANGE	Int1 ... or Num1	Depends upon arguments*	Value of maximum argument minus value of minimum arguments
REM	Num1, Num2	Num	Remainder of Num1/Num2
REVERSE	Alph1 or Anum1 or (MF)Nat1	Depends upon argument	Reverse order of the characters of the argument
(MF)SIGN	Num1	Int	The sign of Num1
SIN	Num1	Num	Sine of Num1
SQRT	Num1	Num	Square root of Num1

STANDARD-DEVIATION	Num1 ...	Num	Standard deviation of arguments
SUM	Int1 ... or Num1 ...	Depends upon arguments*	Sum of arguments
TAN	Num1	Num	Tangent of Num1
UPPER-CASE	Alph1 or Anum1 or (MF) Nat1	Depends upon argument	All letters in the argument are set to uppercase
VARIANCE	Num1 ...	Num	Variance of argument
WHEN-COMPILED	None	Anum	Date and time program was compiled
YEAR-TO-YYYY	Int1 Int2	Int	Argument-1 converted from Int2 YY to YYYY based on the value of argument-2

LISTE DES OPTIONS POUR LES ENTREES-SORTIES EN PLEIN ECRAN

Screen Clauses/ Screen Options/ Data Description Clauses	**SCREEN SECTION**				**MF WITH PHRASE**	
	Input Field	**Output Field**	**Update Field**	**Literal Field**	**ACCEPT**	**DISPLAY**
AUTO	X		X		X	
BACKGROUND-COLOR	X	X	X	X	X	X
BELL	X	X	X	X	X	X
BLANK	X	X	X	X		X
BLANK WHEN ZERO	X	X	X			
BLINK	X	X	X	X	X	X
COLUMN	X	X	X	X		
ERASE	X	X	X	X	X	
FOREGROUND-COLOR	X	X	X	X	X	X
FULL	X		X		X	
GRID	X	X	X	X	X	X
HIGHLIGHT	X	X	X	X	X	X
JUSTIFIED	X	X	X			
LEFT-JUSTIFY					X	
LEFTLINE	X	X	X	X	X	X
LINE	X	X	X	X		
LOWLIGHT	X	X	X	X	X	X
OCCURS	X	X	X			
OVERLINE	X	X	X	X	X	X
PROMPT	X		X		X	
REQUIRED	X		X		X	
REVERSE-VIDEO	X	X	X	X	X	X
RIGHT-JUSTIFY					X	
SECURE	X		X		X	
SIGN	X	X	X			
SIZE	X	X	X	X	X	X
SPACE-FILL					X	
TRAILING-SIGN					X	
UNDERLINE	X	X	X	X	X	X
UPDATE					X	

TABLE DES CARACTÈRES ASCII ÉTENDU

La norme **ASCII** (*American Standard Code for Information Interchange*), a longtemps été utilisée pour le codage de caractères en informatique. Elle a été inventée par l'américain Bob Bemer en 1961. Encore aujourd'hui, la table ASCII est grandement utilisée, même si parfois complétée par une table étendue.

caractère	code ASCII	code hexadécimal
NUL (*Null*)	0	00
SOH (*Start of heading*)	1	01
STX (*Start of text*)	2	02
ETX (*End of text*)	3	03
EOT (*End of transmission*)	4	04
ENQ (*Enquiry*)	5	05
ACK (*Acknowledge*)	6	06
BEL (*Bell*)	7	07
BS (*Backspace*)	8	08
TAB (Tabulation horizontale)	9	09
LF (*Line Feed*, saut de ligne)	10	0A
VT (*Vertical tabulation*, tabulation verticale)	11	0B
FF (*Form feed*)	12	0C
CR (*Carriage return*, retour à la ligne)	13	0D
SO (*Shift out*)	14	0E
SI (*Shift in*)	15	0F
DLE (*Data link escape*)	16	10
DC1 (*Device control 1*)	17	11
DC2 (*Device control 2*)	18	12
DC3 (*Device control 3*)	19	13
DC4 (*Device control 4*)	20	14
NAK (*Negative acknowledgement*)	21	15
SYN (*Synchronous idle*)	22	16
ETB (*End of transmission block*, fin de bloc de transmission)	23	17
CAN (*Cancel*, annulation)	24	18
EM (*End of medium*, fin du médium)	25	19
SUB (*Substitute*, substitut)	26	1A
ESC (*Escape*, caractère d'échappement)	27	1B
FS (*File separator*, séparateur de fichier)	28	1C
GS (*Group separator*, séparateur de groupe)	29	1D
RS (*Record separator*, séparateur d'enregistrement)	30	1E
US (*Unit separator*, séparateur d'enregistrement)	31	1F
SP (*Space*, espace)	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34

5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
[	91	5B
\	92	5C
]	93	5D
^	94	5E
_	95	5F
`	96	60
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72

s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
|	124	7C
}	125	7D
~	126	7E
Touche de suppression	127	7F

TABLE DES MATIERES