INSTITUT de la gestion publique et du développement économique

Support stagiaire

COBOL
MODULE 2
Les tables

MINISTÈRE DE L'ÉCONOMIE
DES FINANCES ET DE L'INDUSTRIE

Sommaire

Présentation des tables

1. PRÉSENTATION.

1.1. DÉFINITION.

Une table est un groupe de données composé d'éléments simples ayant tous des attributs identiques (même nature et même longueur)

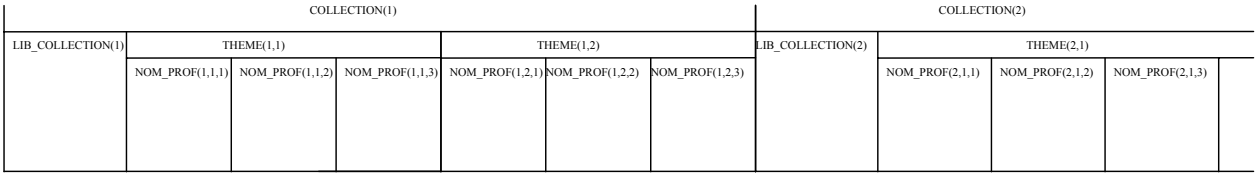
1.2. ILLUSTRATION.

1.2.1. REPRÉSENTATION VISUELLE D'UNE TABLE.

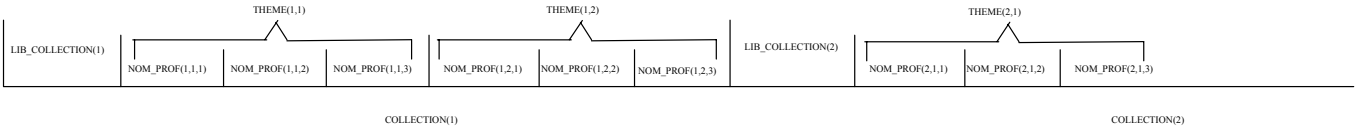
Une maison d'édition commercialise trois collections. Chacune de ces collections aborde deux thèmes. Enfin, chacun de ces thèmes est traité par trois spécialistes du sujet exposé.

Pour représenter cette situation on peut adopter le schéma suivant :

Transparent 02.

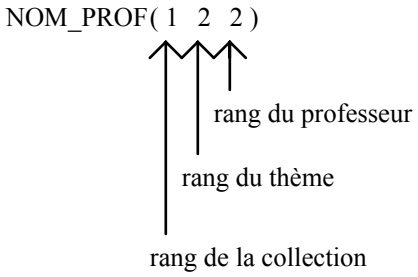


Une autre représentation possible pourrait être :



NOTES

Mais il existe encore d'autres façons de schématiser cette situation. Ces figures font apparaître la répétition de zones identiques, ainsi que les niveaux hiérarchiques. Ainsi, la donnée LIB_COLLECTION est répétée 2 fois sur la figure ci-dessus ; son contenu sera différent selon la collection envisagée, mais sa description sera identique, ici : zone alphanumérique de 20 caractères. D'autre part, pour une collection donnée, nous aurons le choix entre 2 thèmes. Ainsi chacune des zones groupes COLLECTION peut être vue comme une grosse boîte, à l'intérieur de laquelle nous aurons deux boîtes plus petites, les boîtes THEME, elles-mêmes contenant 3 petites boîtes avec le nom d'un professeur. Ceci implique que si l'on souhaite connaître le nom d'un des professeurs, il faudra indiquer la collection dont il s'agit, le thème, et le rang du professeur. D'où la nécessité des indices, qui vont du niveau général au niveau le plus fin. Si l'on veut le nom du professeur de rang 2, traitant le thème 2 de la collection 1, on regardera le contenu de la boîte NOM_PROF(1,2,2) où :

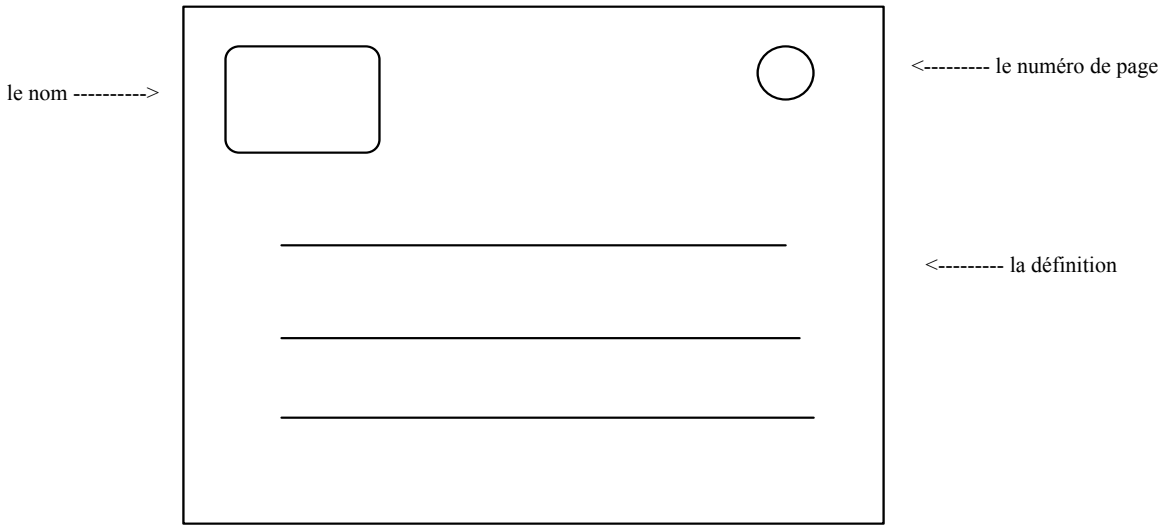


NOTES

1.2.2. RECHERCHE DANS UNE TABLE

Une table permet la recherche d'informations. Différentes méthodes sont utilisées.

Une table peut être regardée comme un dictionnaire dans lequel il n'y aurait qu'un mot par page. On aurait alors le schéma suivant :



NOTES

Recherche d'un mot dans un dictionnaire :

Première méthode (principe de la recherche séquentielle) :

- Supposons que les mots ne soient pas classés par ordre alphabétique. Pour retrouver un mot, une méthode consiste à consulter une à une toutes les pages du dictionnaire. C'est ainsi que commençant par le début ou par la fin du dictionnaire, nous comparerons le mot cherché avec le mot lu sur les pages successives et ce jusqu'à ce que l'on trouve le mot voulu (à moins qu'il n'existe pas, auquel cas on ne s'en apercevra que lorsque tout le dictionnaire aura été consulté). Cette première méthode peut toujours être utilisée.

Deuxième méthode (principe de la recherche directe) :

- On cherche le mot se trouvant à la nième page. On va directement à la page recherchée, et on récupère l'information voulue. Cette recherche nécessite un classement, c'est-à-dire une relation entre le numéro de page et le mot.

Troisième méthode (principe de la recherche dichotomique) :

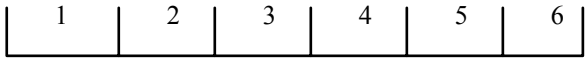
- La méthode pour trouver un mot dans le dictionnaire consiste, ici, à le feuilleter et, en comparant le mot cherché avec le mot de la page à laquelle le dictionnaire est ouvert, à déterminer si le mot cherché se trouve avant ou après le mot lu et ainsi de suite jusqu'à ce que le mot cherché soit identique au mot lu. Cette méthode n'est valable que si les mots dans le dictionnaire sont triés par ordre alphabétique. Sans cet ordre il serait impossible de situer le mot cherché par rapport au mot lu.

NOTES

1.3. TERMINOLOGIE APPLIQUÉE AUX TABLES.

- le mot recherché s'appelle argument de recherche
- le mot dans le dictionnaire se nomme argument ou argument de référence
- la définition s'appelle la fonction recherchée
- la page s'appelle un poste ou un élément
- le numéro de la page est le rang du poste ou de l'élément

La table a plusieurs éléments tout comme le dictionnaire a plusieurs pages. On a ainsi en mémoire centrale de l'ordinateur une table dont les éléments se suivent de la manière suivante :



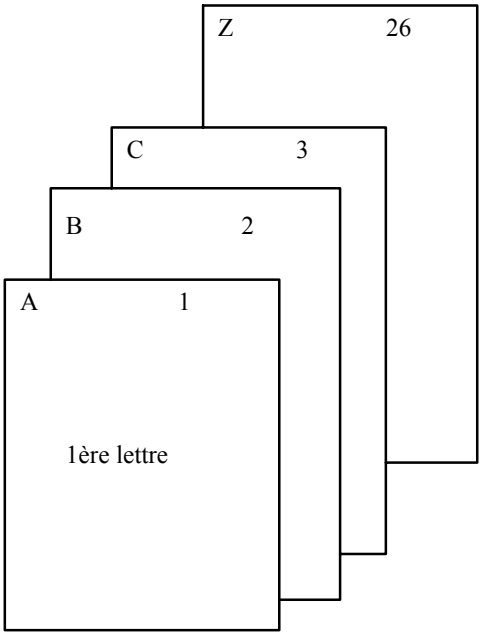
TABLE

Cette table est composée de 6 éléments ou postes, chacun étant lui-même composé d'un argument et d'une fonction.

NOTES

Une table en mémoire qui se compose comme suit :

Correspond au dictionnaire suivant :



La lettre constitue l'argument, le nombre constitue la fonction

NOTES

1.4. INTÉRÊTS DES TABLES.

les tables sont fréquemment utilisées en gestion :

- pour vérifier que les indicatifs sont valides (par exemple lorsqu'un client demande une pièce P4810 pour une machine M513, on vérifie dans la table que machine et pièce existent et sont compatibles)
- pour rechercher en table le prix unitaire d'un article connaissant son code (on créerait une table des codes avec pour chacun d'eux le prix unitaire applicable)
- sélection d'articles selon critère(s) (par exemple dans une entreprise, sélection des salariés ayant un nombre n d'années d'ancienneté)

NOTES

FICHE 02

Description d'une table

1. DESCRIPTION.

La table est décrite dans la partie "WORKING- STORAGE SECTION" de la DATA DIVISION d'un programme ; lors de l'exécution de ce dernier, cette description permettra la réservation en mémoire centrale d'emplacements contigus de mémoire.

1.1. CLAUSE OCCURS.

Elle permet de définir une seule fois plusieurs données semblables : elle indique le nombre de fois qu'un élément de donnée est répété et permettra l'utilisation des indices et des index.

Format 1 de la clause OCCURS :

nom_donnée OCCURS nb_entier [TIMES]
[ASCENDING KEY (ou DESCENDING KEY) IS nom_donnée1[,nom_donnée2]]
[INDEXED BY nom_index1[,nom_index2……]]

Format 2 de la clause OCCURS

nom_donnée OCCURS nb_entier1 TO nb_entier2 [TIMES] DEPENDING ON nom_donnée3
[ASCENDING KEY (ou DESCENDING KEY) IS nom_donnée4[,nom_donnée5]]
[INDEXED BY nom_index1[,nom_index2……]]

NOTES

commentaires :

- Les parties entre crochets sont facultatives
- dans le format 1, la valeur de nb_entier représente le nombre exact de répétitions, et est un entier positif
- dans le format 2, la valeur courante de la zone spécifiée par nom_donnée3 représente le nombre de répétitions. La valeur de nb_entier2 représente le nombre maximum de répétitions et nb_entier1 le nombre minimum. Ceci n'implique pas que la longueur de nom_donnée soit variable, mais simplement que le nombre de répétitions l'est. La valeur de la zone spécifiée par nom_donnée3 doit être comprise entre nb_entier1 et nb_entier2
- KEY IS indique que la donnée répétitive (ici, nom_donnée) est ordonnée en séquence croissante ou décroissante, selon les valeurs contenues dans nom_donnée1, nom_donnée2….
- la clause ASCENDING ou DESCENDING KEY IS s'emploie pour signaler un critère de classement, elle indique que la donnée répétitive (ici nom_donnée) est ordonnée en séquence croissante ou décroissante, selon les clés. Ce critère de classement sera utilisé notamment par l'instruction SEARCH ALL (recherche dichotomique en table). Les arguments de classement doivent être situés à une adresse fixe par rapport au début de l'élément dont ils sont les clefs ; ils ne doivent pas contenir de clause OCCURS.
- INDEXED BY nom_index1 définit un index. Cette notion sera étudiée plus tard dans le cours.

NOTES

1.2. EXEMPLES DE DESCRIPTION DE TABLE

Attention : LE MOT "TABLE" EST UN MOT RESERVE COBOL.

Exemple 1 :

```
01 TAB1.
     05 ELEMENT PIC X(5) OCCURS 10.
```

Dans cet exemple, on a réservé 50 caractères en mémoire centrale sous la forme de 10 postes de 5 caractères

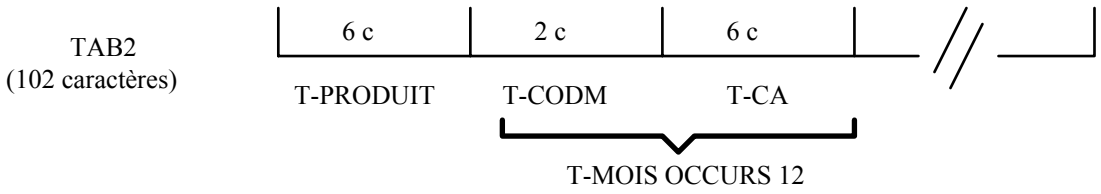
La clause OCCURS ne peut apparaître aux niveaux 01, 66, 77, 88.

Exemple 2 :

Soit une table composée ainsi :

- position 1 à 6 : Code produit
- position 7 à 8 : Code mois } Ces deux zones sont
- position 9 à 14 : Chiffre d'affaires } répétées 12 fois

Le zonage de cette description est :



Longueur de la table : 6 + (12 * 8) = 102 c

NOTES

La description Cobol est la suivante :

```
01 TAB2.
       05 T-PRODUIT PIC X(6).
       05 T-MOIS OCCURS 12.
             10 T-CODM PIC 99.
             10 T-CA PIC 9(6).
```

Si T-CA avait été la seule zone répétée 12 fois, le Cobol devenait :

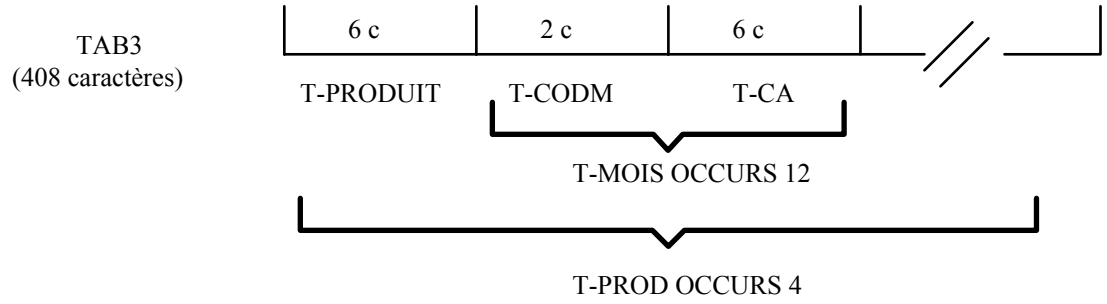
```
01 TAB2.
       05 T-PRODUIT PIC X(6).
       05 T-CA PIC 9(6) OCCURS 12.
```

La longueur de la table est alors : 6 + (12 * 6) = 78 caractères.

Exemple 3 :

Supposons maintenant que cette table soit composée de 4 types de produits.

Le zonage est :



Longueur de la table : [6 + (12 * 8)] * 4 = 408 c

La description Cobol est la suivante :

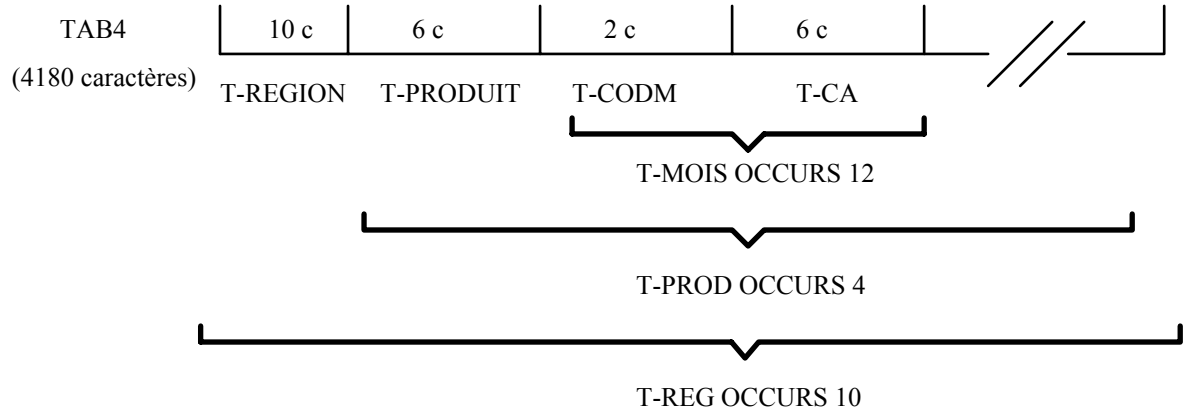
```
01 TAB3.
       02 T-PROD OCCURS 4.
             05 T-PRODUIT PIC X(6).
             05 T-MOIS OCCURS 12.
                   15 T-CODM PIC 99.
                   15 T-CA PIC 9(6).
```

NOTES

Exemple 4 :

Reprenons l'exemple précédent en répartissant les produits par région (maximum : 10 régions)

Le zonage est :



Longueur de la table : [10 + [6 + (12 *8)] * 4] * 10 = 4180 c

La description Cobol est la suivante :

```
01 TAB4.
        05 T-REG OCCURS 10.
              10 T-REGION PIC X(10).
              10 T-PROD OCCURS 4.
                    15 T-PRODUIT PIC X(6).
                    15 T-MOIS OCCURS 12.
                          20 T-CODM PIC 99.
                          20 T-CA PIC 9(6).
```

NOTES

Exemple 5 :

Dans cet exemple on veut traduire par une table la situation suivante : un client peut passer une ou plusieurs commandes, huit au maximum (le nombre de commandes, variable, est indiqué par la zone appelée "T-COMMANDE"). Seul le montant de la (ou des) commande(s) doit apparaître (on ne s'occupe pas des références des commandes).

La description Cobol est la suivante :

```
01 TABFACTURE.
     05 T-CLIENT PIC X(10).
     05 T-COMMANDE PIC 9.
     05 T-MONTANT PIC 9(7) OCCURS 1 TO 8 DEPENDING ON T-COMMANDE.
```

Si, par exemple, un client a passé 3 commandes, le zonage de la table TABFACTURE sera le suivant :

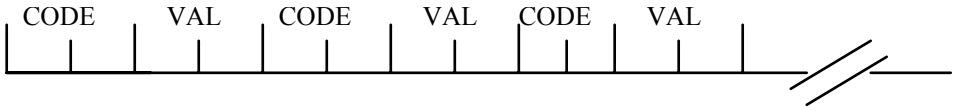
TABFACTURE	CLIENT	3	MONTANT(1)	MONTANT(2)	MONTANT(3)

REMARQUE : T-COMMANDE est placé devant T-MONTANT

NOTES

Exemple 6 :

Soit une table composée de codes et valeurs correspondantes, les codes étant classés suivant les valeurs croissantes :



Cette table sera décrite de la manière suivante :

```
01 TABCODE.
       05 ELEMENT OCCURS 100 TIMES ASCENDING KEY COD.
            10 COD PIC XX.
            10 VALEUR PIC S9(4).
```

Attention : les arguments de classements (12 maximum) doivent être situés à une adresse fixe par rapport au début de l'élément dont ils sont les clefs et ne peuvent contenir eux-mêmes une clause OCCURS.

par exemple :

EX1

```
01 ZONE.
       05 ELEMENT OCCURS 12 TIMES ASCENDING KEY CUMUL.
            10 COD PIC XX.
            10 VALEUR PIC S9(4).
       05 CUMUL PIC S9(4).
```

Cette description est erronée car CUMUL, zone extérieure à ELEMENT, ne peut être un argument de classement ; par contre, COD est un argument possible.

NOTES

EX2

```
01 ZONE.
        05 DEPARTEMENT OCCURS 100 TIMES ASCENDING KEY CODEP.
            10 NBRE PIC 99.
            10 VILLE OCCURS 100 TIMES DEPENDING NBRE ASCENDING KEY
            CODVIL.
                15 CODEP PIC 99.
                15 CODVIL PIC 999.
                15 HABIT PIC 9(9).
```

CODEP ne peut être un argument KEY pour DEPARTEMENT car CODEP est inclus dans une clause OCCURS donc situé à une adresse variable par rapport à chaque groupe DEPARTEMENT.

Il aurait été préférable d'écrire :

```
01 ZONE.

        05 DEPARTEMENT OCCURS 100 TIMES ASCENDING KEY CODEP.
            10 CODEP PIC 99.
            10 NBRE PIC 99.
            10 VILLE OCCURS 100 TIMES DEPENDING NBRE ASCENDING KEY
            CODVIL.
                15 CODVIL PIC 999.
                15 HABIT PIC 9(9).
```

NOTES

1.3. TP1A

Objectif du TP : décrire une table

Enoncé : dans une entreprise, le service de la comptabilité utilise un fichier contenant pour chaque agent, les montants mensuels du salaire brut, de la retenue de Sécurité Sociale, de la cotisation retraite, et du salaire net. Ce fichier, nommé PAIE, est stocké sur bande, et sa structure est la suivante :

Code enregistrement	3 caractères alphanumériques	: X(3)
Matricule	5 caractères alphanumériques	: X(3))
Nom	30 caractères alphanumériques	: X(30)
Code service	3 caractères alphanumériques	: X(3)
Salaire brut	5 chiffres et 2 décimales	: 9(5)V99
Retenue Sécurité Sociale	5 chiffres et 2 décimales	: 9(5)V99
Cotisation retraite	5 chiffres et 2 décimales	: 9(5)V99
Salaire net	5 chiffres et 2 décimales	: 9(5)V99

zones répétées 12 fois

L'organigramme de l'entreprise fait apparaître 5 services, comprenant chacun 3 salariés.

L'entreprise souhaite éditer une liste indiquant pour chaque service :

- le matricule de chaque salarié y travaillant
- le montant des salaires nets versés par mois et par agent et leur cumul
- le montant cumulé de ces salaires pour le service considéré

Question : donner la description de la table qui pourrait servir dans le traitement demandé

FICHE 03

Utilisation des tables

1. PRÉSENTATION INDEX/INDICES.

1.1. INTRODUCTION

Pour gérer les tables, l'utilisation des index diffère assez peu de celle des indices, bien que le concept de base ne soit pas le même. On préférera en général les index aux indices pour des questions de performance et surtout pour les instructions puissantes de recherche en table qu'ils offrent.

1.2. INDICE.

Un indice n'est pas attaché à une table, il peut être utilisé par ailleurs dans le programme. Il doit être décrit en WORKING-STORAGE SECTION (**WSS**) ou en FILE DESCRIPTION (**FD**).

Il peut être :

- soit un littéral numérique entier
- soit un nom de donnée qui doit désigner une valeur numérique entière positive

La valeur maximale autorisée pour un indice est celle définie par la clause OCCURS.

Dans la Procédure Division (**PD**), lorsqu'un indice est utilisé pour désigner un élément particulier d'une table, son contenu est un nombre entier désignant le numéro d'ordre de l'élément compté à partir de 1 pour le 1er.

Attention : sa valeur doit alors être un entier positif non nul, dont le maximum autorisé est défini par la clause OCCURS.

NOTES

exemple 1 :

Reprenons l'exemple 2 de la fiche 02. Pour désigner le cinquième chiffre d'affaires (T-CA), on écrira :

MOVE T-CA(5) TO CUMUL-CA

exemple 2 :

Reprenons l'exemple 3 de la fiche 02. Pour désigner le cinquième chiffre d'affaires du 2ème produit, on écrira :

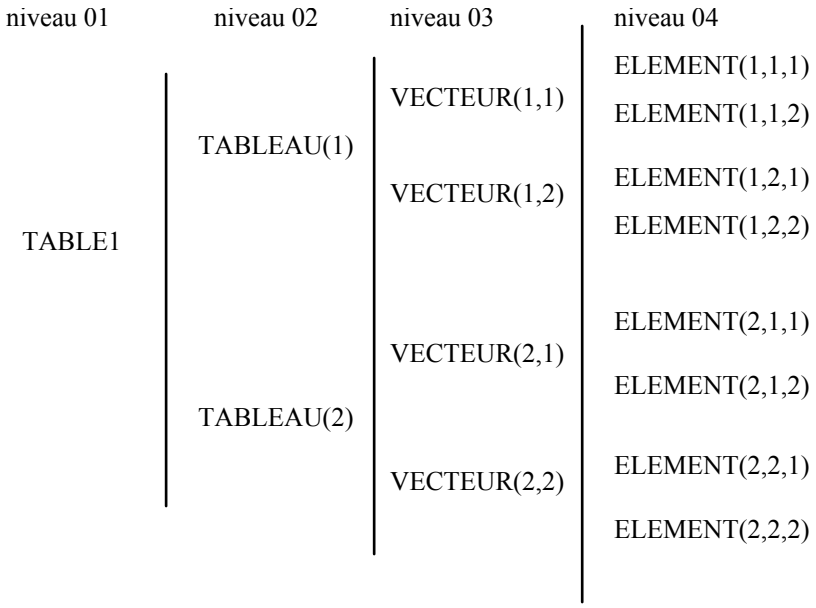
MOVE T-CA(2,5) TO CUMUL-CA

On peut générer un indice de manière indirecte :

EL (IND + n)

Exemple de table à trois niveaux d'indice :

```
01 TABLE1.
       02 TABLEAU OCCURS 2.
            03 VECTEUR OCCURS 2.
                 04 ELEMENT OCCURS 2 PIC 9(4).
```



NOTES

1.3. INDEX.

Un index doit être attaché à une table lors de sa création (12 noms d'index maximum par niveau)

```
01 TAB2.
        05 EL1 OCCURS 2 INDEXED BY IDX1 IDX2 IDX3.
              10 EL2 OCCURS 3 INDEXED BY IDX4 IDX5.
                    15 RUB1 PIC X(4).
                    15 RUB2 PIC X(2).
```

Commentaires :

- On a attaché 3 index au premier niveau de la table (IDX1, IDX2, IDX3) et 2 index au deuxième niveau
- Un index n'a pas besoin d'être décrit en WORKING-STORAGE SECTION car Cobol réserve automatiquement un emplacement en mémoire
- Alors que le contenu d'un indice est un nombre entier désignant le n° d'ordre de l'élément, le contenu de l'index est un nombre d'octets désignant le déplacement de l'élément par rapport au début de la table, d'où une liaison étroite entre l'index et sa table

NOTES

Exemple :

```
TABLE   1      EL PIC X(6) OCCURS 5 INDEXED BY IDX1

       0          6          12          18          24          30
       I----------I----------I----------I----------I----------I
```

```
TABLE   2      EL PIC X(4) OCCURS 3 INDEXED BY IDX2

       0       4       8       12
       I-------I-------I-------I
```

- si on est positionné sur le 3ème poste de la 1ère table, le contenu de IDX1 = 12 (adresse relative du poste)
- si on est positionné sur le 3ème poste de la 2ème table, le contenu de IDX2 = 8 (adresse relative du poste)

Ceci confirme le fait qu'un index doit bien être attaché à sa table. En effet charger la valeur de IDX2 dans IDX1, nous positionnerait, pour la 1ère table, au milieu du 2ème poste.

On remarque aussi que le contenu de l'index correspondant au positionnement sur le premier élément d'une table quelconque est toujours = 0.

Le contenu de l'index étant une adresse relative formalisée en binaire, on ne pourra pas utiliser le verbe MOVE pour mettre une valeur dans l'index.

De même, on ne pourra pas faire un DISPLAY d'index pour accéder à son contenu.

Cependant cela reste transparent pour le programmeur ; ce dernier continue à travailler en numéro de poste (comme avec un indice), Cobol se chargeant du reste. La différence réside dans l'emploi de verbes spécifiques (SET et SEARCH)

On pourra indexer de manière indirecte :

- EL(IDX1 + 1) EL(IDX1 - 4)…….

On peut mélanger indice et index : EL(IND + 3,IDX - 4)

NOTES

1.4. INSTRUCTION SET DANS LE CAS D'UNE UTILISATION D'INDEX

On utilisera le verbe SET pour :

♦ *POSITIONNER* un index ou plusieurs index

exemple : SET IDX1 TO 1 , signifie positionner l'index IDX1 sur le 1er poste de la table
SET IDX1 IDX2 TO 1 , positionne IDX1 et IDX2 sur la même occurrence dans leur table respective.

♦ *MOUVEMENTER* un index dans un autre index

exemple : SET IDX2 TO IDX1, cette instruction positionne IDX2 sur l'occurrence pointée par IDX1 (travail en parallèle sur 2 tables).

♦ *INCREMENTER* ou *DECREMENTER* 1 index

exemple : SET IDX UP BY 1, incrémente d'une occurrence
SET IDX DOWN BY 3, décrémente de 3 occurrences

♦ pour *SAUVEGARDER* la valeur d'un index ; dans ce cas il faut préalablement définir une zone en WORKING-STORAGE SECTION

Par exemple :

01 IDXSAUV USAGE IS INDEX.

Dans ce cas il n'est pas précisé de clause PICTURE : c'est le compilateur Cobol qui se charge de réserver la zone (généralement 4 octets binaires).

Dans la PROCEDURE DIVISION, on pourra si l'on a besoin de sauvegarder le contenu d'un index IDX par exemple, écrire l'instruction : SET IDXSAUV TO IDX.

Il est toutefois préférable d'attacher à une table plusieurs index, et de se servir de l'un d'eux comme index de sauvegarde.

NOTES

1.5. TABLEAU COMPARATIF INDICE / INDEX

INDICES	INDEX
indépendants de la table	associés à une table
doivent être décrits (littéral ou nom de donnée)	Cobol se charge de les réserver
on peut avoir 7 niveaux d'indice en ANS85 (3 en ANS74)	on peut avoir 7 niveaux d'index en ANS85 (3 en ANS74)
l'indice indirect est autorisé depuis ANS85 ex : el(ind + 4)	on peut utiliser l'indexage indirect ex : el (idx + 4)
l'indice est un nombre entier désignant le numéro d'ordre de l'élément	l'index contient un déplacement en nombre d'octets par rapport au début de la table
on peut gérer les indices grâce aux verbes Cobol habituels (ADD,MOVE)	on emploie des verbes particuliers (SET, SEARCH)
on peut sauvegarder un indice dans une zone numérique	on peut sauvegarder un index dans une zone particulière : 01 zone usage index ou dans un autre index d'une même table
un indice peut servir à plusieurs tables	un index est associé à une table. Il est déconseillé de l'utiliser pour une autre table

NOTES

FICHE 04

Chargement d'une table

Une table peut être chargée de 3 façons différentes :

- lors de sa description, par assignation de valeurs prédéfinies
- par initialisation
- par chargement à partir d'un fichier.

1. CHARGEMENT D'UNE TABLE PAR ASSIGNATION DE VALEURS PRÉDÉFINIES.

La démarche est la suivante : il faut réserver en mémoire centrale une zone de travail qui ait la taille de la table entière. Cette zone est chargée avec les valeurs souhaitées grâce à la clause ***VALUE***. Ensuite, cette même zone est redécoupée (= redéfinie) pour constituer la table, de manière à mettre en évidence les éléments qui se répètent plusieurs fois (= les postes de la table). **L'opération se fait impérativement dans cet ordre**, dans la mesure où il est interdit de redéfinir une zone occursée, ou de lui assigner des valeurs, de même qu'il est interdit de donner des valeurs à une zone redéfinie.

Exemple :

```
01 TAB-JOUR.
      05 PIC X(8) VALUE "LUNDI".
      05 PIC X(16) VALUE "MARDI...MERCREDI".
      05 PIC X(16) VALUE "JEUDI...VENDREDI".
      05 PIC X(16) VALUE "SAMEDI..DIMANCHE".
01 TAB-J REDEFINES TAB-JOUR.
      05 W-JOUR PIC X(8) OCCURS 7.
```

Remarque : chaque point à l'intérieur d'une clause VALUE matérialise un caractère blanc

On accédera à cette table par le numéro du jour :
W-JOUR(5) = "VENDREDI".

NOTES

2. CHARGEMENT PAR INITIALISATION.

L'initialisation des tables se fait en PROCEDURE DIVISION par le verbe MOVE.

Exemple 1 :

En WORKING-STORAGE SECTION :

```
01 TABLE1.
      05 W-POSTE1 OCCURS 5.
            10 W-EL11 PIC 9(3).
            10 W-EL12 PIC 9(5).
```

En PROCEDURE DIVISION :

```
      MOVE ZERO TO TABLE1
```

Exemple 2 :

En WORKING-STORAGE SECTION :

```
01 TABLE2.
      05 W-POSTE2 OCCURS 5.
            10 W-EL21 PIC X(5).
            10 W-EL22 PIC X(3).
```

En PROCEDURE DIVISION :

```
      MOVE SPACES TO TABLE2
```

NOTES

Exemple 3 :

En WORKING-STORAGE SECTION :

```
01 TABLE3.
      05 W-POSTE3 OCCURS 8 INDEXED BY IND3.
            10 W-EL31 PIC 9(2).
            10 W-EL32 PIC X(10).
```

En PROCEDURE DIVISION :

```
      PERFORM VARYING I3 FROM 1 BY 1 UNTIL IND3 > 8
            MOVE ZERO TO W-EL31(IND3)
            MOVE SPACES TO W-EL32(IND3)
      END-PERFORM
```

Cette méthode permet la mise à jour de la table poste par poste. Il existe une autre instruction permettant de traduire cette initialisation automatiquement, mais il faut savoir qu'elle est plus gourmande en temps CPU. Il s'agit de l'instruction INITIALIZE.

On écrira ainsi en PROCEDURE DIVISION :

- INITIALIZE TABLE3.

Pour compléter ce paragraphe, étudions l'instruction INITIALIZE.

Format de cette instruction :

- INITIALIZE zone [REPLACING (ALPHABETIC / NUMERIC / ALPHANUMERIC) DATA BY valeur].

Si REPLACING n'est pas précisé, les zones alphanumériques sont initialisées à espaces et les zones numériques à zéros. Avec l'option REPLACING il est possible d'initialiser les zones alphanumériques avec une chaîne de caractères autres que le caractère blanc, par exemple "AAAAAAAAAA", et les zones numériques avec une valeur autre que zéro, par exemple 12.

NOTES

3. CHARGEMENT DYNAMIQUE À PARTIR D'UN FICHIER.

La table, décrite en WORKING-STORAGE SECTION, est chargée à partir d'un fichier lors de l'exécution du programme. Dans ce cas de figure, la table ne comporte pas de clause VALUE, c'est-à-dire que les postes de la table ne sont pas préalablement servis à l'exécution du programme.

A partir d'un fichier, il est possible de charger une table soit séquentiellement, soit directement. Le chargement séquentiel est toujours possible dans la limite de la taille autorisée. Chaque poste de la table doit contenir un argument de recherche. Le chargement direct n'est possible que si l'on peut faire correspondre un argument de recherche avec un numéro de poste. Dans ce cas, l'argument de recherche est égal au numéro de poste et il est inutile de faire figurer l'argument de recherche dans le poste de la table.

NOTES

3.1. CHARGEMENT SÉQUENTIEL.

3.1.1. CHARGEMENT SÉQUENTIEL À UN NIVEAU.

Les opérations doivent se dérouler dans l'ordre suivant :

1 - évaluation du nombre de postes de la table. S'il n'est pas indiqué, il faudra l'évaluer en fonction du connu. Par exemple, si la clé est formée de 3 caractères numériques, on pourra se limiter à 999 postes, ou pour tester la fin de table, à 1000 postes.

2 - description de la table (rappel : TABLE est un mot réservé COBOL ; et l'on ne peut avoir de clause OCCURS au niveau 01).

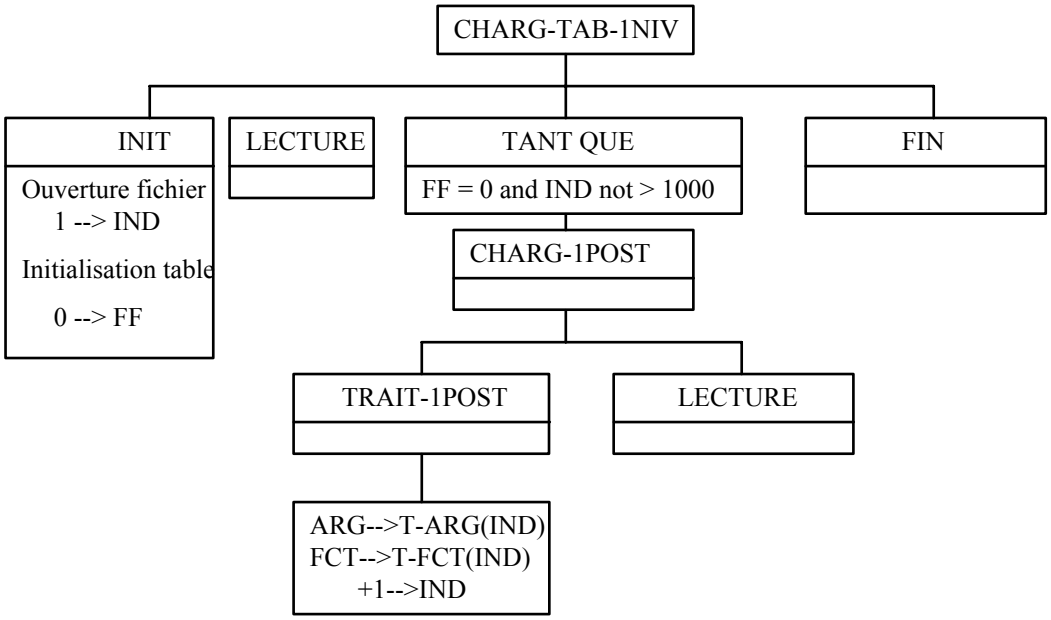
3 - initialisation de la table par précaution.

4 - chargement de la table séquentiellement en se servant d'un indice qu'il faudra initialiser, ou d'un index.

(rappel : les indices, nom de donnée contenant un numéro de poste devront être définis en numérique. On peut initialiser un indice à zéro (ce n'est qu'une zone numérique ordinaire) mais on ne peut pas l'utiliser pour indicer une zone quand il contient 0).

NOTES

Exemple d'arbre programmatique d'un chargement séquentiel (table à 1 niveau) :



Attention : même en ayant prévu le nombre de postes de la table, il faut tester le cas de dépassement de capacité, par exemple en cas de doublon sur l'argument.

NOTES

Traduction COBOL :

```
DATA DIVISION.
FD FICHIER.
1 ART.
      5       PIC X(20).
      5 ARG PIC 9(3).
      5       PIC X(10).
      5 FCT  PIC X(15).
      5       PIC X(50).

WORKING-STORAGE SECTION.
1 FF PIC X VALUE "0".
1 IND PIC 9(4).
1 TAB.
      5 T-POST OCCURS 1000.
            10 T-ARG PIC 9(3).
            10 T-FCT PIC X(15).

PROCEDURE DIVISION.
DEB.
      OPEN INPUT FICHIER
      INITIALIZE TAB
      READ FICHIER AT END MOVE "1" TO FF
      END-READ
      PERFORM VARYING IND FROM 1 BY 1 UNTIL IND > 1000 OR FF = "1"
            MOVE ARG TO T-ARG(IND)
            MOVE FCT TO T-FCT(IND)
            READ FICHIER AT END MOVE "1" TO FF
            END-READ
      END-PERFORM
      IF FF NOT = "1" DISPLAY "DEPASSEMENT TABLE "
      END-IF
      CLOSE FICHIER
      STOP RUN.(ou suite du traitement).
```

NOTES

NB : le chargement d'une table indexée se fera de la même manière. Il ne faudra pas déclarer d'index en WORKING-STORAGE SECTION, l'index étant défini lors de la déclaration de la table.

EXEMPLE :

```
1 TAB.
      5 T-POST OCCURS 1000 INDEXED BY IND.
            10 T-ARG PIC 9(3).
            10 T-FCT PIC X(15).
```

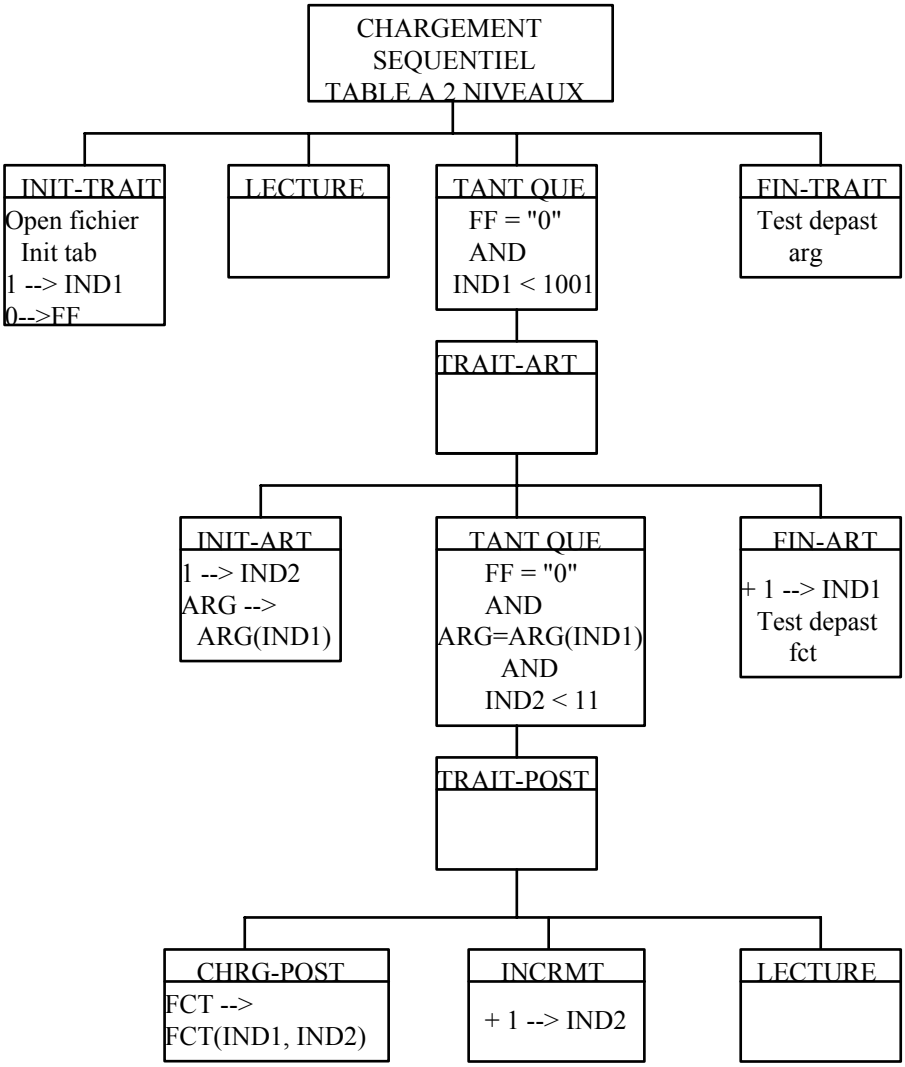
Dans ce cas, aucun changement n'intervient dans la PROCEDURE DIVISION.

NOTES

3.1.2. CHARGEMENT SÉQUENTIEL À 2 NIVEAUX.

Les étapes préconisées dans le chargement des tables à un niveau restent valables dans le chargement des tables à 2 niveaux. En fait, il s'agit d'une table à l'intérieur d'une autre table.

Exemple d'arbre programmatique d'un chargement séquentiel (table à 2 niveau) :



NOTES

Traduction COBOL :

```
DATA DIVISION.
FD FICHIER.
1 ART.
      5       PIC X(20).
      5 ARG PIC 9(3).
      5       PIC X(10).
      5 FCT  PIC X(15).
      5       PIC X(50).

WORKING-STORAGE SECTION.
1 FF PIC X VALUE "0".
1 IND1 PIC 9(4).
1 IND2 PIC 9(4).
1 TAB.
      5 T-POST OCCURS 1000.
            10 T-ARG PIC 9(3).
            10 T-FCT PIC X(15) OCCURS 10.

PROCEDURE DIVISION.
DEB.
      OPEN INPUT FICHIER
      INITIALIZE TAB
      READ FICHIER AT END MOVE "1" TO FF
      END-READ
      PERFORM VARYING IND1 FROM 1 BY 1 UNTIL IND1 > 1000 OR FF = "1"
            MOVE ARG TO T-ARG(IND1)
            PERFORM VARYING IND2 FROM 1 BY 1 UNTIL ARG NOT = ARG(IND1) OR
                  IND2 > 10 OR FF = "1"
                  MOVE FCT TO T-FCT(IND1, IND2)
                  READ FICHIER AT END MOVE "1" TO FF
                  END-READ
            END-PERFORM
            PERFORM TEST-NB-FCT
      END-PERFORM
      IF FF NOT = "1" DISPLAY "DEPASSEMENT : TROP D'ARG"
      END-IF
      CLOSE FICHIER
      STOP RUN.(ou suite du traitement).
```

NOTES

3.2. CHARGEMENT DIRECT.

La façon de procéder reste applicable en cas de chargement direct, c'est-à-dire :

- ♦ évaluation du nombre de postes de la table
- ♦ initialisation de la table et des données de WORKING-STORAGE SECTION.

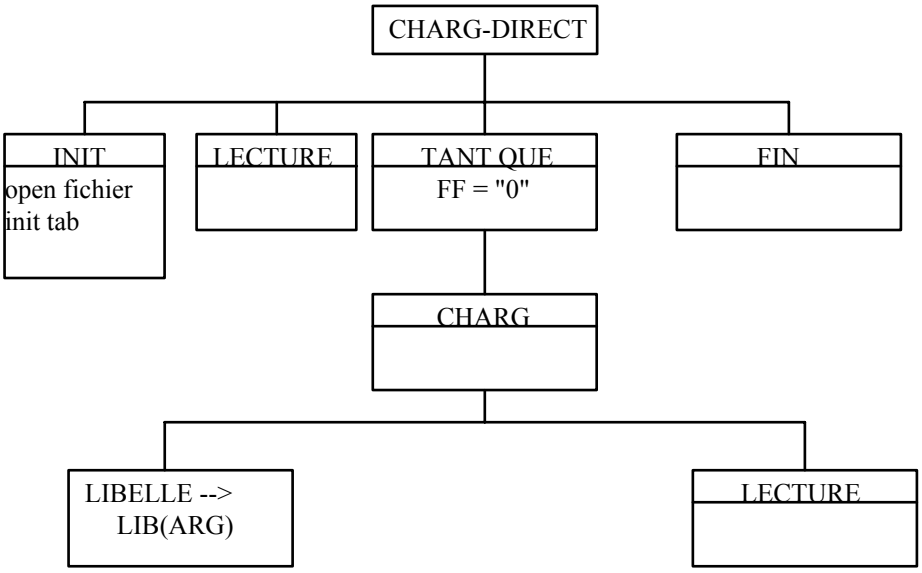
Exemple : Soit un fichier non trié contenant un numéro d'article et un libellé, plus d'autres informations qui ne nous servirons pas ici. Il faut constituer une table en mémoire permettant une recherche directe d'un libellé à partir de son numéro d'article.

A partir du même fichier que celui étudié précédemment, la description de la table sera différente : l'argument n'aura plus à figurer dans la table puisqu'il correspondra au rang dans la table (attention : l'argument doit alors impérativement être numérique). Dans le cas où des numéros d'articles seraient absents dans le fichier d'entrée, leur place dans la table resterait vide : d'où l'importance de l'initialisation, avant le chargement, pour les détecter lors d'une consultation ultérieure. Par contre, le test sur le dépassement de capacité sera inutile puisqu'en cas de doublon, ceux-ci s'écraseront successivement.

Même si le fichier en entrée n'est pas trié, la table résultant du chargement direct sera triée sur l'argument.

NOTES

Exemple d'arbre programmatique d'un chargement direct :



NOTES

Traduction COBOL :

```
DATA DIVISION.
FD FICHIER.
1 ART.
      5       PIC X(20).
      5 ARG PIC 9(3).
      5       PIC X(10).
      5 LIBELLE PIC X(15).
      5       PIC X(50).
WORKING-STORAGE SECTION.
1 FF PIC X VALUE "0".
1 TAB.
      5 LIB OCCURS 1000 PIC X(15).
PROCEDURE DIVISION.
DEB.
      OPEN INPUT FICHIER
      INITIALIZE TAB
      READ FICHIER AT END MOVE "1" TO FF
      END-READ
      PERFORM UNTIL FF = "1"
            MOVE LIBELLE TO LIB(ARG)
            READ FICHIER AT END MOVE "1" TO FF
            END-READ
      END-PERFORM
      CLOSE FICHIER
      STOP RUN.(ou suite du traitement).
```

NOTES

3.3. TP2A.

On veut charger en mémoire un fichier PRODUIT contenant 10 articles

Structure du fichier PRODUIT :

N° de produit ----------2 caractères, trié en ordre croissant et pouvant prendre les valeurs de 01 à 10
Libellé du produit ----------10 caractères

QUESTION :

Quel mode de chargement utiliseriez-vous pour charger une table des produits ?

3.4. TP3A

On veut charger en mémoire un fichier STOCK non trié contenant 10 articles

Structure du fichier STOCK :

N° de stock ----------2 caractères, pouvant prendre les valeurs 02,10,03,11,15,05,21,28,06,01
Libellé du stock ----------10 caractères

QUESTION :

Quel mode de chargement utiliseriez-vous pour charger une table des stocks ?

3.5. TP4A(DÉBUT).

Objectif du TP :

- chargement d'une table
- recherche en table

Les services sociaux de la région parisienne disposent d'un fichier CENTRE recensant tous les lieux de colonie.

La structure du fichier CENTRE est la suivante :

N° du centre ---------- 3 caractères numériques
Libellé du centre ----------10 caractères alphanumériques

Le fichier CENTRE n'est pas trié, et il existe 10 centres de vacances.

Question : écrire le Cobol du chargement de la table à partir du fichier CENTRE

FICHE 05

Recherche en table

Une fois chargée ou constituée en mémoire, on peut accéder à la table selon 3 méthodes :

- la recherche séquentielle, qui consiste à balayer la table jusqu'à obtention du poste désiré,
- la recherche directe, selon certaines conditions,
- la recherche dichotomique, utilisée surtout pour les grosses tables.

1. RECHERCHE SÉQUENTIELLE.

La recherche séquentielle ne nécessite aucune condition particulière, la table peut être triée ou non.

Le principe :

- on se positionne sur le premier poste de la table en initialisant l'index (ou l'indice) à 1.
- on compare l'argument de recherche avec l'argument de la table. S'il y a égalité, il s'agit du poste cherché. Dans le cas contraire, l'index ou l'indice est incrémenté de 1 et on continue l'opération jusqu'à ce qu'on trouve le poste ou que l'on arrive en fin de table. Dans ce dernier cas, l'argument de référence n'existe pas.

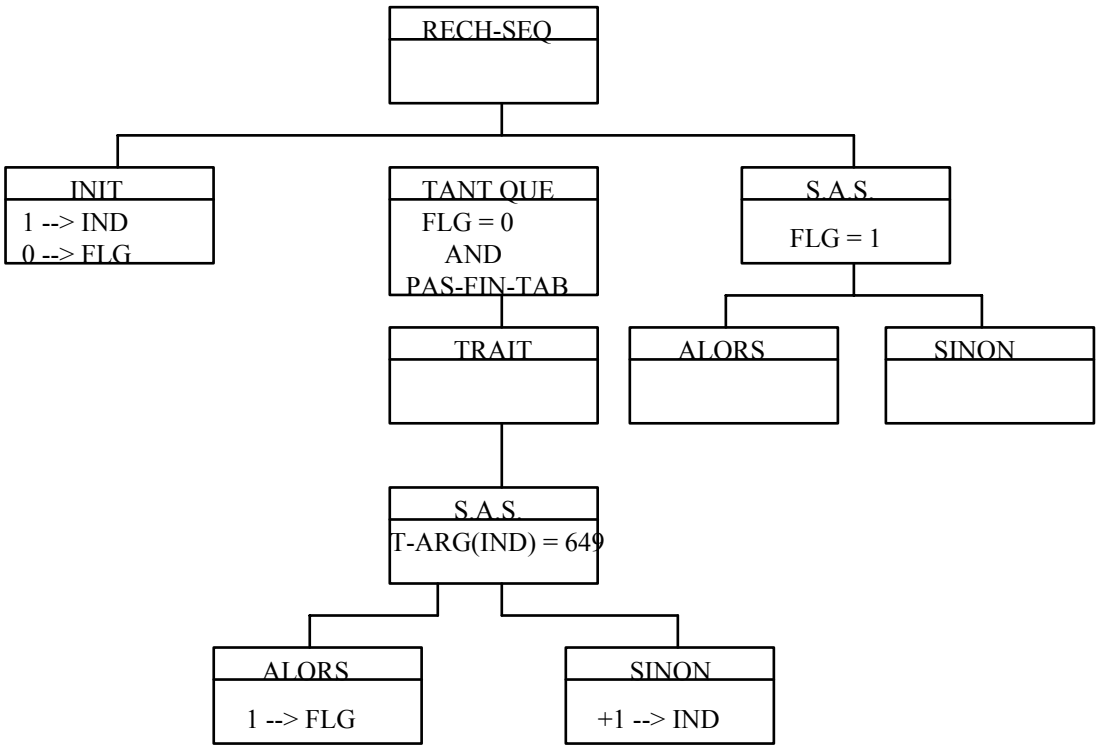
Pour optimiser la recherche séquentielle, il est possible de :

- tester le dernier poste chargé (en cas de chargement séquentiel et selon l'initialisation),
- tester un indicateur de fin de table positionné à la fin du chargement,
- tester l'égalité par rapport à l'index (ou indice) obtenu en fin de chargement et sauvegardé,
- en cas de table triée, on peut aussi tester le numéro du poste par rapport à l'argument de recherche.

NOTES

Exemple :

On souhaite obtenir le libellé de l'élément dont l'argument est 649. Si on a pris la précaution d'initialiser la table à zéro, on balaiera la table jusqu'à obtention du résultat ou jusqu'à ce que l'argument soit égal à zéro.



NOTES

Traduction COBOL :

```
DATA DIVISION.
WORKING-STORAGE SECTION.
1 TAB.
      5 T-POST OCCURS 100 INDEXED BY IND.
                        10 T-ARG PIC 999.
                        10 T-FCT PIC X(30).
1 FLG PIC 9 VALUE 0.
PROCEDURE DIVISION.
RECH-TAB.
      PERFORM VARYING IND FROM 1 BY 1 UNTIL IND > 100 OR FLG = 1
                        IF T-ARG(IND) = 649 MOVE 1 TO FLG
                        END-IF
      END-PERFORM
      IF FLG = 1 DISPLAY "ELEMENT TROUVE : " T-FCT(IND)
                        ELSE DISPLAY "ERREUR : " T-ARG
      END-IF.
```

NOTES

2. L'ORDRE SEARCH.

Pour les tables indexées, il est possible d'utiliser un verbe unique SEARCH pour la recherche séquentielle.

FORMAT COBOL.

SEARCH *POST* [VARYING *IDX*]

AT END ordre impératif1
WHEN condition1 faire…..
WHEN condition2 faire…..
END-SEARCH.

Le verbe SEARCH permet de rechercher en table un élément qui satisfasse à une ou plusieurs conditions données.

Dans la clause VARYING, *IDX* peut être :

- soit un index attaché à la table : dans la liste des index attachés à la table et associés à la zone occursée *POST*, il ne pourra pas s'agir de l'index principal, c'est-à-dire du premier index déclaré (en effet, l'index principal varie automatiquement grâce au verbe SEARCH). *IDX* variera à chaque boucle à partir d'une valeur initiale **chargée au préalable par un ordre SET.**
- soit un index externe, également initialisé. Dans ce cas, il y a progression parallèle de *IDX* et du premier index déclaré pour *POST*. A la fin de l'exploration de la 1ère table, si on se trouve sur le 5ème poste on pourra se trouver également sur le 5ème poste de la 2ème table (l'initialisation ayant été parallèle).
- soit une zone de donnée numérique entière ou une donnée d'"USAGE" index dont le contenu sera incrémenté conjointement et de la même valeur que l'index associé à *POST* à chaque boucle.

Les conditions mettront généralement en oeuvre des comparaisons de valeurs d'index, aussi le programmeur devra-t-il constamment garder à l'esprit que ce ne sont pas les contenus des index qui seront comparés mais leur équivalent en nombre d'occurrences. Le verbe SEARCH ressemble beaucoup au PERFORM VARYING mais on ne peut explorer qu'un seul niveau d'index, et non l'ensemble de la table comme il est possible de le faire avec le PERFORM. D'autre part, il est impératif de mettre une valeur initiale à chaque index par l'ordre SET avant tout verbe SEARCH. L'incrémentation des index n'a lieu qu'en fin de boucle.

NOTES

Si on reprend l'exemple précédent, la DATA DIVISION restant identique :

PROCEDURE DIVISION.

```
RECH-TAB.
      SET IND TO 1
      SEARCH T-POST
                              AT END DISPLAY "ERREUR : " T-ARG
                              WHEN T-ARG(IND) = 649 DISPLAY "ELEMENT TROUVE : " T-FCT
      END-SEARCH
```

Remarques :

- L'index a bien été positionné à 1 afin de balayer totalement la table.
- T-POST, **zone occursée**, est déclarée lors de son utilisation avec le verbe SEARCH **sans son index**.
- A chaque boucle, COBOL évalue tout d'abord la condition AT END de fin de table ; si elle n'est pas satisfaite, les conditions sont examinées dans l'ordre de leur classement. Aussi, l'ordre des conditions n'est pas indifférent : quand aucune condition n'est satisfaite, l'index associé à POST est alors incrémenté (celui dans le VARYING aussi éventuellement) et enfin il y a retour en début de boucle ; quand la condition est remplie au niveau d'un WHEN, on exécute l'instruction correspondante et on se débranche immédiatement après l'instruction qui suit le END-SEARCH, en ignorant les WHEN qui suivent. A noter que quand on sort du SEARCH l'index reste positionné sur le poste correspondant.
- La ou les conditions suivant les WHEN sont quelconques : elles ne font pas intervenir de critères de classement. WHEN dans le SEARCH simple, accepte tous les symboles de comparaison et les 2 opérateurs logiques AND et OR.

Le VARYING est utilisé pour faire évoluer l'index d'une autre table parallèlement ou un autre index de la même table (jusqu'à 12 index par niveau : le premier index associé à la table varie automatiquement. Les autres doivent évoluer avec la clause VARYING). S'il s'agit d'une autre table, le compilateur se chargera des conversions et quand l'élément sera trouvé, si on est positionné sur le 2ème poste de la première table on sera également positionné sur le 2ème poste de la deuxième table grâce à la clause VARYING.

NOTES

3. RECHERCHE DIRECTE.

C'est la plus simple des recherches à mettre en place mais elle nécessite une table triée, sans interruption, ni doublon. Néanmoins, les postes pourront ne pas être tous servis si leur place est réservée. Dans ce cas-là, les postes contiendront la valeur d'initialisation.

Le numéro de poste doit correspondre à l'argument de recherche. Lors du chargement direct on mettait la fonction du poste dans l'emplacement correspondant à la place de l'argument. En recherche directe, le principe est le même : si on veut le libellé du poste 649, on mettra 649 dans l'indice ou l'index et on ira chercher directement le libellé.

Exemple :

Le chargement se faisait par : MOVE LIBELLE TO T-FCT(ARG)

La recherche se fera par : MOVE T-FCT(649) TO ZON

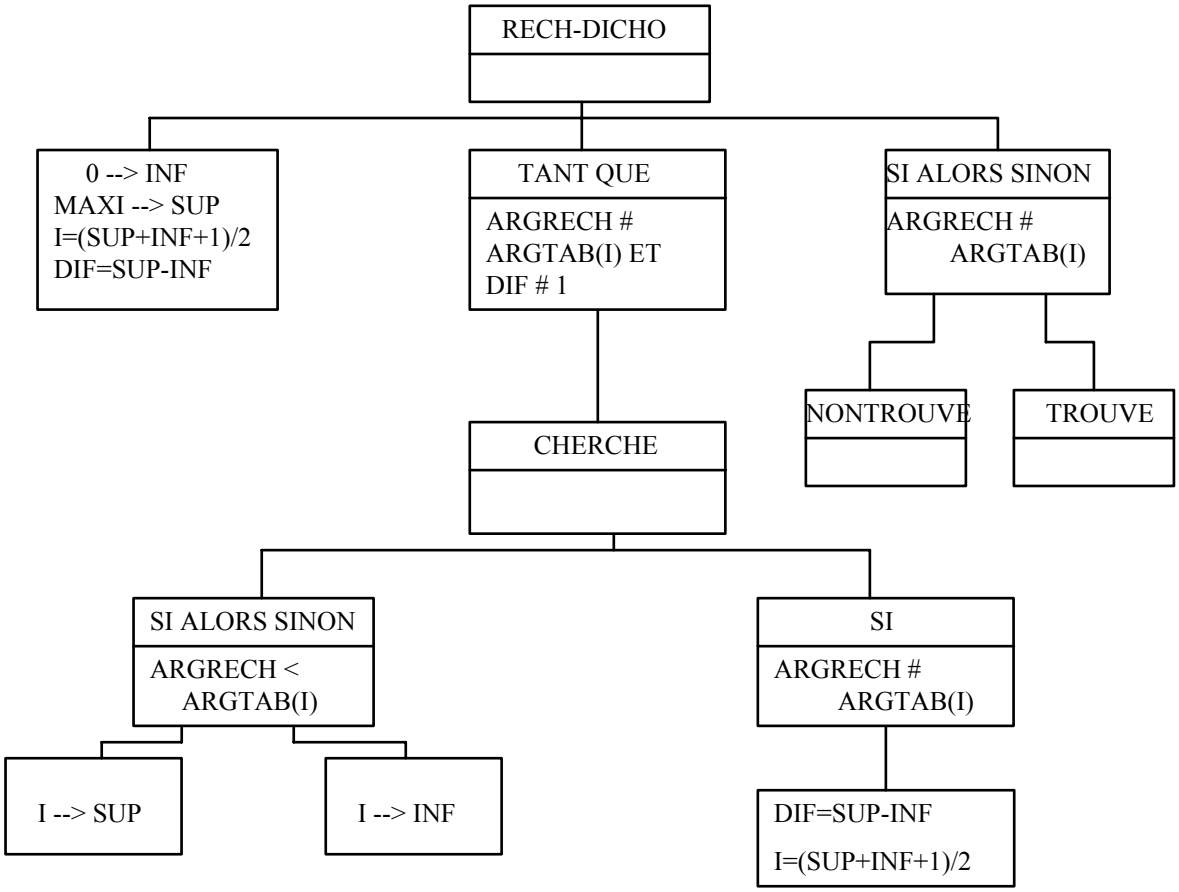
NOTES

4. RECHERCHE DICHOTOMIQUE.

Cette méthode est utilisée pour rechercher un élément dans une table triée mais non continue (à la différence de la recherche directe qui nécessite une table triée et continue).

PRINCIPE :

On divise chaque fois le champ de recherche par 2, et on compare le code de l'élément recherché avec le code de l'élément central : selon que l'élément recherché se situe dans la 1ère ou la 2ème moitié de la table, on élimine l'autre partie. On recommence l'opération avec la table réduite de moitié.



Comme il a été dit plus haut, une recherche dichotomique n'est rentable qu'à partir d'une trentaine de postes. Son principe a été expliqué, ainsi que son application parce qu'une routine COBOL l'utilise : c'est le SEARCH ALL qui va être étudié plus longuement au paragraphe suivant.

NOTES

5. L'ORDRE SEARCH ALL.

Comme nous l'avons déjà précisé lors de l'étude de la recherche dichotomique, cette recherche nécessite des tables triées, et les critères de tri doivent eux-mêmes répondre à certaines exigences.

Format Cobol :

```
SEARCH ALL post
      AT END instruction impérative I1
      WHEN condition C1 instruction impérative I2
      WHEN……
END-SEARCH
```

Cette instruction peut se traduire comme suit :

- explorer tous les postes
- en fin de table exécuter I1
- quand C1 est réalisée faire I2, etc…

Le poste doit appartenir à la table, c'est-à-dire être lié par une clause OCCURS, mais comme pour la syntaxe du verbe SEARCH simple, on ne doit pas associer d'index à cet élément ; en outre, le poste doit être associé à une clause ASCENDING ou DESCENDING KEY car les conditions doivent obligatoirement faire intervenir un des critères de classement indiqués et ce critère doit être le 1er terme de la condition.

NOTES

La condition, lors de l'utilisation du verbe SEARCH ALL ne fera intervenir que le symbole " = " et exclusivement l'opérateur logique " AND " s'il s'agit d'une condition composée.

Avec cette instruction, destinée à l'exploration totale d'une table, le premier index est toujours employé pour progresser dans la table et il ne sera pas nécessaire de l'initialiser par l'ordre SET, COBOL s'en chargeant.

EXEMPLE :

```
1 TAB.
      5 POST OCCURS 100 ASCENDING KEY EL1 EL2 INDEXED BY I1 I2.
            10 EL1 PIC 999.
            10 EL2 PIC 999.
            10 LIBELLE PIC X(50).

      SEARCH ALL POST
            AT END PERFORM PAS-TROUVE
            WHEN EL1(I1) = 234 AND EL2(I1) = 456 NEXT SENTENCE
      END-SEARCH.
```

Remarques :

- ici encore l'INDEX n'apparaît pas dans la syntaxe du verbe SEARCH ALL.
- EL2 étant cité comme élément de tri, EL1 doit aussi être cité.
- si l'élément est trouvé, l'index reste positionné sur lui.
- on ne peut utiliser le SEARCH ALL (comme le SEARCH simple) que pour explorer un niveau d'index. Quand on veut explorer une table à plusieurs niveaux, il faut créer une boucle avec un PERFORM pour les niveaux supérieurs.

NOTES

EXEMPLE :

```
1 TAB.
      5 POST OCCURS 50 ASCENDING KEY EL1 INDEXED BY I1.
            10 EL1 PIC 99.
            10 EL2 OCCURS 10 INDEXED BY I2.
                  15 LIB1 PIC X(12).

      PERFORM VARYING I1 FROM 1 BY 1 UNTIL I1 > 50
            SEARCH ALL EL2
                  AT END DISPLAY " LIBELLE NON TROUVE "
                  WHEN LIB1(I1,I2) = "DECEMBRE    "
                        DISPLAY "LIBELLE TROUVE "
            END-SEARCH
      END-PERFORM
```

Remarque :

Le PERFORM VARYING gère les premiers niveaux d'index, le SEARCH ALL uniquement la recherche au niveau le plus fin.

NOTES

Exemple :

```
1 TAB1.
      5 EL1 OCCURS 100 INDEXED BY I1 ASCENDING KEY COD1.
            10 COD1 PIC 999.
            10 LIB1 PIC X(50).
1 TAB2.
      5 EL2 OCCURS 20 INDEXED BY I2.
            10 LIB2 PIC X(30).
1 W-SAUV.
      5 COD PIC 999.
      5 LIB PIC X(30).
PROCEDURE DIVISION.
RECHERCHE-SEQUENTIELLE.
      SET I1 TO 1
      SET I2 TO 1
(ou   SET I1 I2 TO 1)
      SEARCH EL1 VARYING I2
            AT END DISPLAY "ERREUR"
            WHEN COD1(I1) = 35 MOVE COD1(I1) TO COD
                              MOVE LIB2(I2) TO LIB
      END-SEARCH.
RECHERCHE-DICHOTOMIQUE.
      SEARCH ALL EL1
            AT END DISPLAY "ERREUR"
            WHEN COD1(I1) = 35 SET I2 TO I1
                              MOVE COD1(I1) TO COD
                              MOVE LIB2(I2) TO LIB
      END-SEARCH.
```

NOTES

REMARQUES :

- dans la recherche simple, il faut initialiser les deux index avant leur utilisation. On a utilisé le VARYING pour faire varier l'index attaché à la 2ème table, I1 variant automatiquement. Quand l'élément est trouvé, les 2 index sont pointés sur la même occurrence.
- dans la recherche dichotomique, la 1ère condition était que la table soit triée, l'argument de tri étant précisé dans la clause ASCENDING KEY. L'initialisation de l'index I1 n'est plus nécessaire, mais lorsque l'élément est trouvé, il faut positionner le 2ème index par un SET.

En conclusion, si la programmation peut résoudre tout problème de recherche en table, les instructions SEARCH et SEARCH ALL permettent une recherche en table très simple, et justifient à elles seules la préférence des index par rapport aux indices.

NOTES

5.1. TP4A(FIN)

Les services sociaux ont aussi à leur disposition une table interne des départements franciliens et de leur libellé :

75 Paris;77 Seine et Marne;78 Yvelines;91 Essonne;92 Hauts de Seine;93 Seine Saint Denis;94 Val de Marne;95 Val d'Oise.

Question : décrire la table interne des départements.

D'autre part, chaque famille désirant inscrire un ou plusieurs enfants en colonie, dépose un dossier qui constitue un article du fichier COLONIE.

Structure du fichier COLONIE :

N° identification ---------- 6 caractères triés en ordre croissan
Nom ---------- 10 caractères
Adresse---------- 10 caractères
Prénom---------- ---------- --7 caractères
N° de centre d'affectation ---------3 caractères
} zones répétées 4 fois

Règles de gestion :

- si l'enfant n'est pas affecté dans un centre, la zone "N° de centre" est à zéro.
- pour un article donné, il y a au moins un enfant et au plus 4 enfants
- les deux premiers caractères du numéro d'identification de la famille correspondent au département.

Question : afficher une liste des enfants affectés en colonie, par département, comprenant le nom, le prénom, le numéro de centre et le libellé du centre.

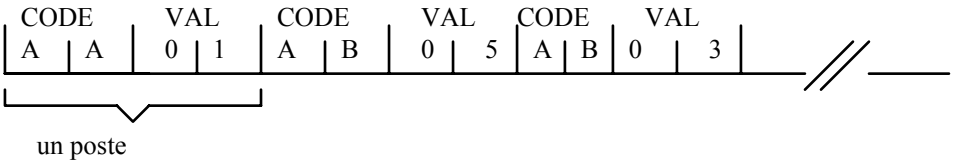
A chaque rupture sur le département, la liste sera précédée d'une ligne mentionnant le libellé du département traité.

FICHE 06

Tri de table

Pour optimiser certains accès (accès séquentiel), ou pour certains autres (accès directs ou dichotomiques), une table doit être triée.

Prenons la table suivante :



Cette table est triée en majeur et en ascendant sur CODE, en mineur et en décroissant sur VAL.

Les algorithmes de tris sont nombreux ; en effet, un algorithme doit par exemple prendre en compte le volume des données à trier.

Certains tris sont complexes à programmer, par exemple le tri dichotomique.

1. EXEMPLE D'ALGORITHME DE TRI : LE BUBBLE-SORT

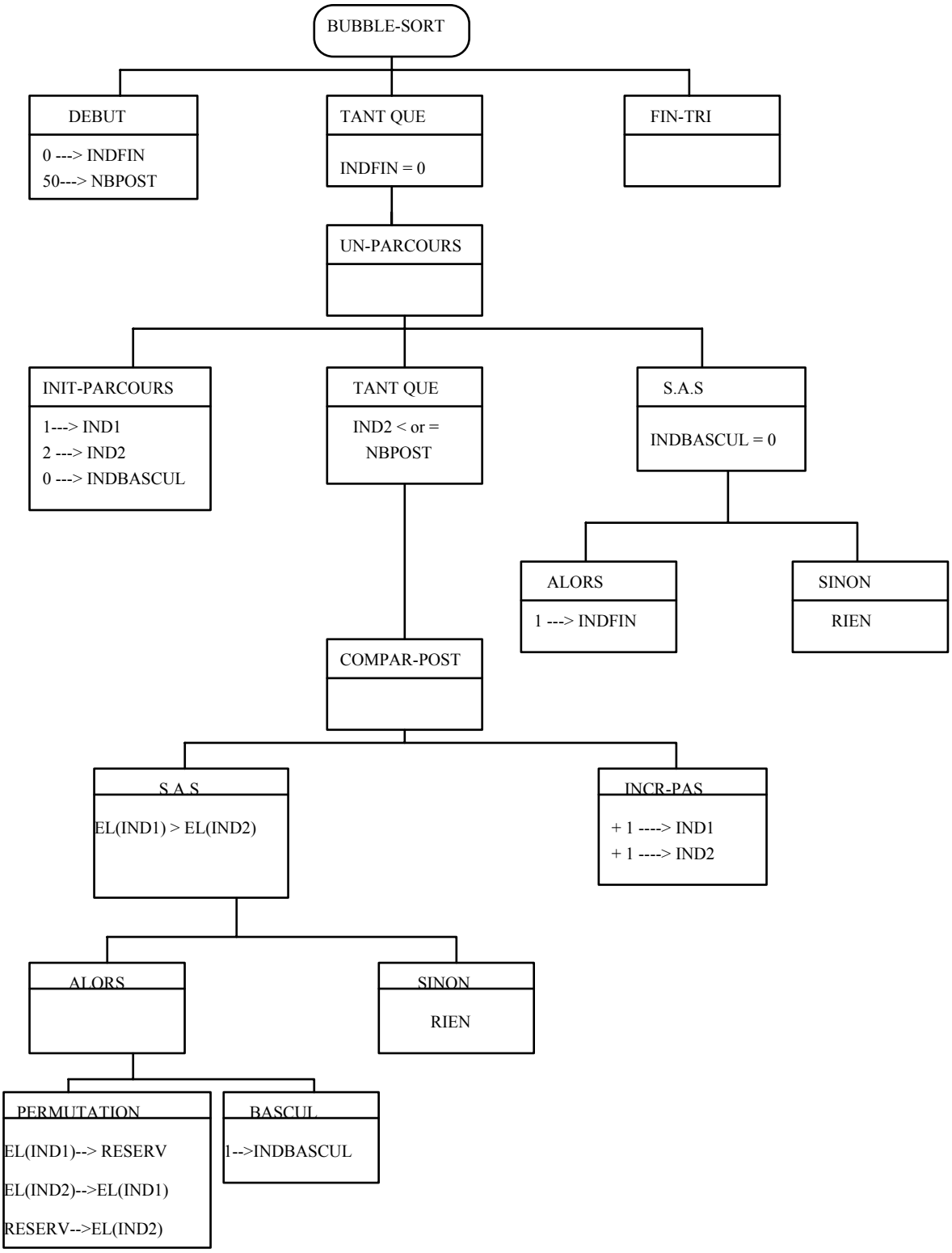
Principe du BUBBLE-SORT :

- on parcourt la table en comparant chaque poste à son suivant immédiat : si l'ordre du tri n'est pas respecté, on intervertit les deux postes et on continue jusqu'à ce que toute la table ait été parcourue.
- on parcourt la table du début à la fin n fois : la table est triée quand il n'y a plus aucune inversion de postes lors d'un parcours. Il faut donc prévoir un indicateur de basculement qu'on initialisera à 0 avant chaque nouveau parcours de la table. Quand on procédera à un basculement, on mettra l'indicateur à 1. Si l'indicateur reste à 0 à la fin d'un parcours, la table est triée.

NOTES

Transparent 03.

Exemple d'arbre programmatique d'un BUBBLE-SORT :



Description COBOL du tri à bulle :

1ère solution :

```
WORKING- STORAGE SECTION
77 INDFIN PIC 9 VALUE ZERO.
77 INDBASCUL PIC 9 VALUE 0.
77 RESERV PIC 9(3).
77 NBPOST PIC 99 VALUE 50.
77 IND1 PIC 9(3).
77 IND2 PIC 9(3).
01 TAB.
   05 EL PIC 9(3) OCCURS 50.
PROCEDURE DIVISION.
BUBBLE-SORT.
      PERFORM UN-PARCOURS UNTIL INDFIN = 1
      PERFORM FIN-TRI
      STOP RUN.
UN-PARCOURS.
      PERFORM INIT-PARCOURS
      PERFORM COMPAR-POST UNTIL IND2 > NBPOST
      IF INDBASCUL = 0
         MOVE 1 TO INDFIN
      END-IF.
INIT-PARCOURS.
      MOVE 1 TO IND1
      MOVE 2 TO IND2
      MOVE 0 TO INDBASCUL.
COMPAR-POST.
      IF EL(IND1) > EL(IND2)
         PERFORM PERMUTATION THRU GEST-INDIC
      END-IF
      ADD 1 TO IND1 IND2.
PERMUTATION.
      MOVE EL(IND1) TO RESERV
      MOVE EL(IND2) TO EL(IND1)
      MOVE RESERV TO EL(IND2).
GEST-INDIC.
      MOVE 1 TO INDBASCUL.
FIN-TRI.
```

NOTES

2ème solution :

```
WORKING- STORAGE SECTION
77 INDBASCUL PIC 9 VALUE 0.
88 W-VRAI VALUE 1.
88 W-FAUX VALUE 0.
77 RESERV PIC 9(3).
77 NBPOST PIC 99 VALUE 50.
77 IND1 PIC 9(3).
01 TAB.
   05 EL PIC 9(3) OCCURS 50.
PROCEDURE DIVISION.
BUBBLE-SORT.
      PERFORM UN-PARCOURS TEST AFTER UNTIL W-FAUX
      PERFORM FIN-TRI
      STOP RUN.
UN-PARCOURS.
      SUBTRACT 1 FROM NBPOST
      SET W-FAUX TO TRUE
      PERFORM VARYING IND1 FROM 1 BY 1 UNTIL IND1 = NBPOST
         IF EL(IND1) > EL(IND1 + 1)
            PERFORM PERMUTATION
         END-IF
      END-PERFORM.
PERMUTATION.
      MOVE EL(IND1) TO RESERV
      MOVE EL(IND1 + 1) TO EL(IND1)
      MOVE RESERV TO EL(IND1 + 1)
      SET W-VRAI TO TRUE.
FIN-TRI.
```

NOTES

Commentaire :

Cette fois on utilise qu'un seul indicateur (INDBASCUL).

Afin de pouvoir entrer la première fois dans UN-PARCOURS alors que INDBASCUL est à 0, on fait un PERFORM TEST AFTER.

On n'utilise qu'un indice IND1, puisque que Cobol ANS85 autorise l'indice relatif (IND + 1).

Une propriété du tri à bulle : à chaque parcours, le plus grand élément se trouve "remonté" en fin de table et se trouve de ce fait trié.

Connaissant cette propriété, on diminue d'un poste à chaque fois le parcours de la table (-1 dans NBPOST).

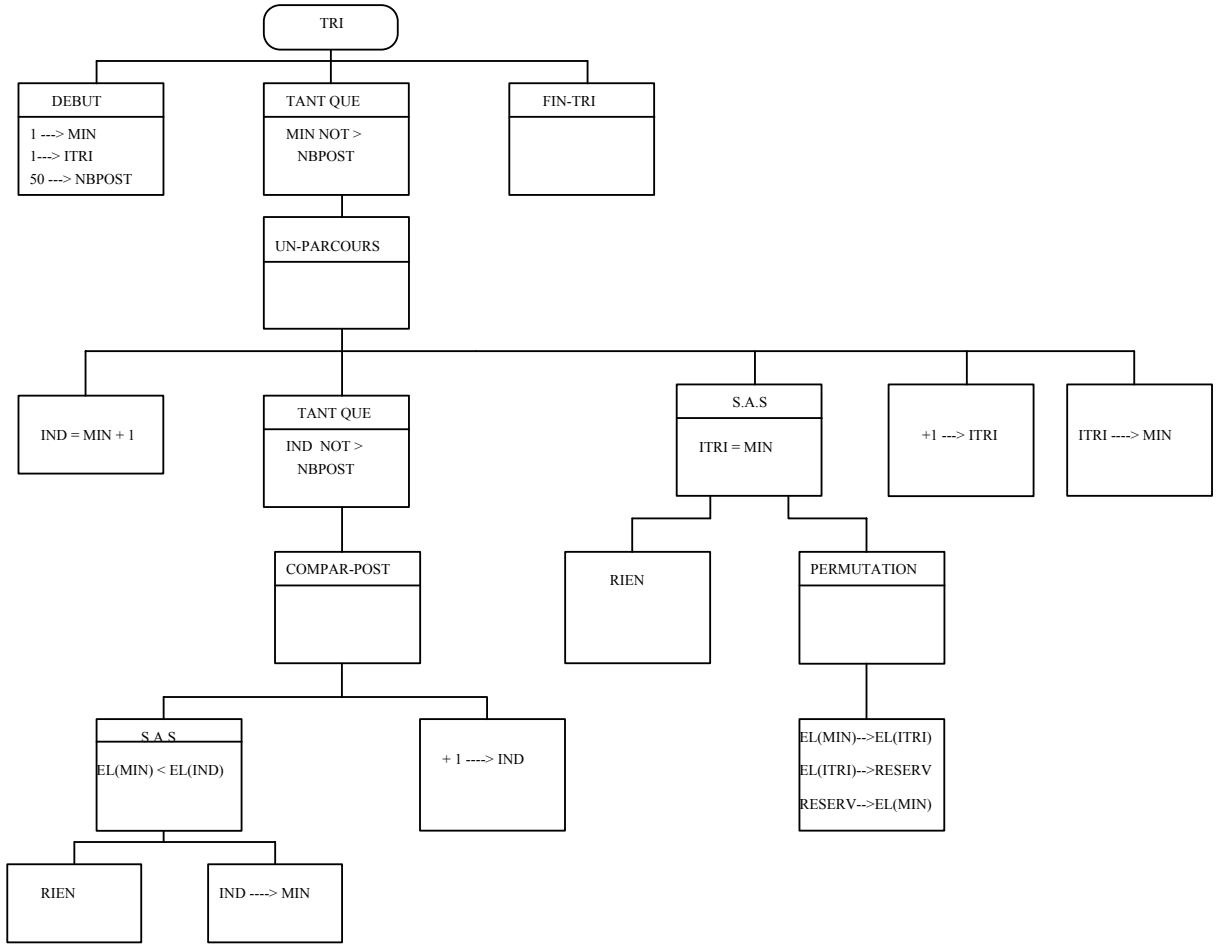
2. AUTRE EXEMPLE D'ALGORITHME DE TRI.

Conventions :

- MIN = pointe l'élément le plus petit
- IND = indice de progression dans la table
- ITRI = donne le rang de l'élément de rangement du plus petit
- NBPOST = taille de la table

NOTES

Transparent 04.



NOTES

3. UTILISATION DU TRI COBOL (SORT).

On ne donnera ici qu'un exemple d'utilisation du SORT Cobol ; cet utilitaire sera abordé dans le module SYNTHESE.

Intérêts de l'exemple :

- la table décrite dans le TP est ici triée en utilisant le SORT Cobol
- chargement séquentiel d'une table à trois niveaux (T-TAB1) à partir d'un fichier ; cette table est décrite avec l'option DEPENDING ON, et l'on voit que pour l'initialiser on utilise le PERFORM VARYING, de manière à initialiser chaque poste de la table ; en effet, lorsqu'il y a une telle option dans une table, on ne peut utiliser l'instruction INITIALIZE. A noter que l'objet d'une clause OCCURS avec l'option DEPENDING ON doit être la dernière zone groupe ou la dernière zone élémentaire d'un enregistrement, il ne peut être suivi d'une donnée d'un niveau égal ou inférieur ; d'autre part quand une zone groupe contient la clause OCCURS, la donnée qui en dépend ne peut être de longueur variable. On n'aurait pas pu avoir :

```
T-TAB1.
   5 T-CODSERV OCCURS 5.
      10 T-SERV PIC X(3).
      10 T-CODMAT OCCURS 1 TO 3 DEPENDING ON paramètre.
         15 T-MATRICULE PIC X(5).
         15 T-MONTSALNET OCCURS 3.
            20 T-SALNET PIC 9(6)V99.
         15 T-CUMSALNET PIC 9(6)V99.
      10 CUMSALSERV PIC 9(10)V99.
```

NOTES

Transparent 05 **UTILISATION DU TRI COBOL (SORT).**

```
B=MAIN m=LIB:   IGPDE    CBX       >  ln=50                                    ctl  ____
                                                          TOP
>>>>100             IDENTIFICATION DIVISION.
       200           PROGRAM-ID. IGPDE.
       300           ENVIRONMENT DIVISION.
       700           INPUT-OUTPUT SECTION.
       800           FILE-CONTROL.
       900                 SELECT FITRAV          ASSIGN TO FITRAV.

     1 000           DATA DIVISION.
     1 100           FILE SECTION.
     1 200           SD FITRAV.
     1 300           01 ARTRAV.
     1 400                5 CRITERE PIC X (12).
     1 500           WORKING-STORAGE SECTION.
     1 600           1 P-TABSPORT.
     1 700                5  F  PIC   X (12)   VALUE    "05BASKET".
     1 800                5  F  PIC   X (12)   VALUE    "08FOOTBALL".
     1 900                5  F  PIC   X (12)   VALUE    "01RUGBY".
     2 000                5  F  PIC   X (12)   VALUE    "10CYCLISME".
     2 100                5  F  PIC   X (12)   VALUE    "29AIKIDO".
     2 200                5  F  PIC   X (12)   VALUE    "12VOLLEY".
     2 300                5  F  PIC   X (12)   VALUE    "38TENNIS".
     2 400                5  F  PIC   X (12)   VALUE    "18EQUITATION".
     2 500                5  F  PIC   X (12)   VALUE    "15LUTTE".
     3 200                5  F  PIC   X (12)   VALUE    "36DANSE".
     3 300           1 P-INDSPORT REDEFINES P-TABSPORT.
     3 400                5 P-ACTIV OCCURS 10 ASCENDING KEY P-SPORT INDEXED BY SP1.
     3 600                     10 P-SPORT PIC 99.
     3 700                     10 P-LIBELLE PIC X (10).
     3 800           PROCEDURE DIVISION.
>>>3 900             DEBUT.
     4 000                SORT FITRAV ON ASCENDING KEY CRITERE
     4 100                INPUT PROCEDURE AVANT-TRI
     4 200                OUTPUT PROCEDURE APRES-TRI
     4 300                PERFORM VARYING SP1 FROM 1 BY 1 UNTIL SP1 > 10
     4 400                     DISPLAY P-SPORT (SP1) " " P-LIBELLE (SP1)
     4 500                END-PERFORM
     4 600                STOP RUN.
     4 700           AVANT-TRI
     4 800                PERFORM VARYING SP1 FROM 1 BY 1 UNTIL SP1 > 10
     4 900                                RELEASE ARTRAV FROM P-ACTIV (SP1)
     5 000                END-PERFORM.
     5 100           APRES-TRI.
>>>5 200                 SET SP1 TO 1
     5 300                PERFORM VARYING SP1 FROM 1 BY 1 UNTIL SP1 > 10
     5 400                   RETURN FITRAV
     5 500                          AT END DISPLAY "FIN DU RETURN FITRAV "
     5 600                          NOT AT END
     5 700                                MOVE ARTRAV TO P-ACTIV (SP1)
     5 800                   END-RETURN
     5 900                END-PERFORM
```

3.1. TP5A.

I - Une association sportive désire établir des statistiques sur le nombre de ses adhérents par activité et par période.

La liste des activités est déterminée de manière stable, et est stockée en bibliothèque. Pour chaque application, il suffit de l'insérer par l'instruction COPY.

LISTE DES ACTIVITES SPORTIVES : SPORTS

```
01 P-TABSPORT.
      5     PIC X(12) VALUE "05BASKET".
      5     PIC X(12) VALUE "08FOOTBALL".
      5     PIC X(12) VALUE "01RUGBY".
      5     PIC X(12) VALUE "10CYCLISME".
      5     PIC X(12) VALUE "29AIKIDO".
      5     PIC X(12) VALUE "12VOLLEY".
      5     PIC X(12) VALUE "38TENNIS".
      5     PIC X(12) VALUE "18EQUITATION".
      5     PIC X(12) VALUE "15LUTTE".
      5     PIC X(12) VALUE "49SKI".
      5     PIC X(12) VALUE "13GOLF".
      5     PIC X(12) VALUE "26NATATION".
      5     PIC X(12) VALUE "02ATHLETISME".
      5     PIC X(12) VALUE "25HOCKEY".
      5     PIC X(12) VALUE "09PATINAGE".
      5     PIC X(12) VALUE "36DANSE".
01 P-INDSPORT REDEFINES P-TABSPORT.
      5 P-ACTIV OCCURS 16 ASCENDING KEY P-SPORT INDEXED BY SP1 SP2 SP3.
            10 P-SPORT PIC 99.
            10 P-LIBELLE PIC X(10).
```

1.1 - Ecrire un programme réalisant le tri de la table des activités sportives en fonction croissante sur P-SPORT

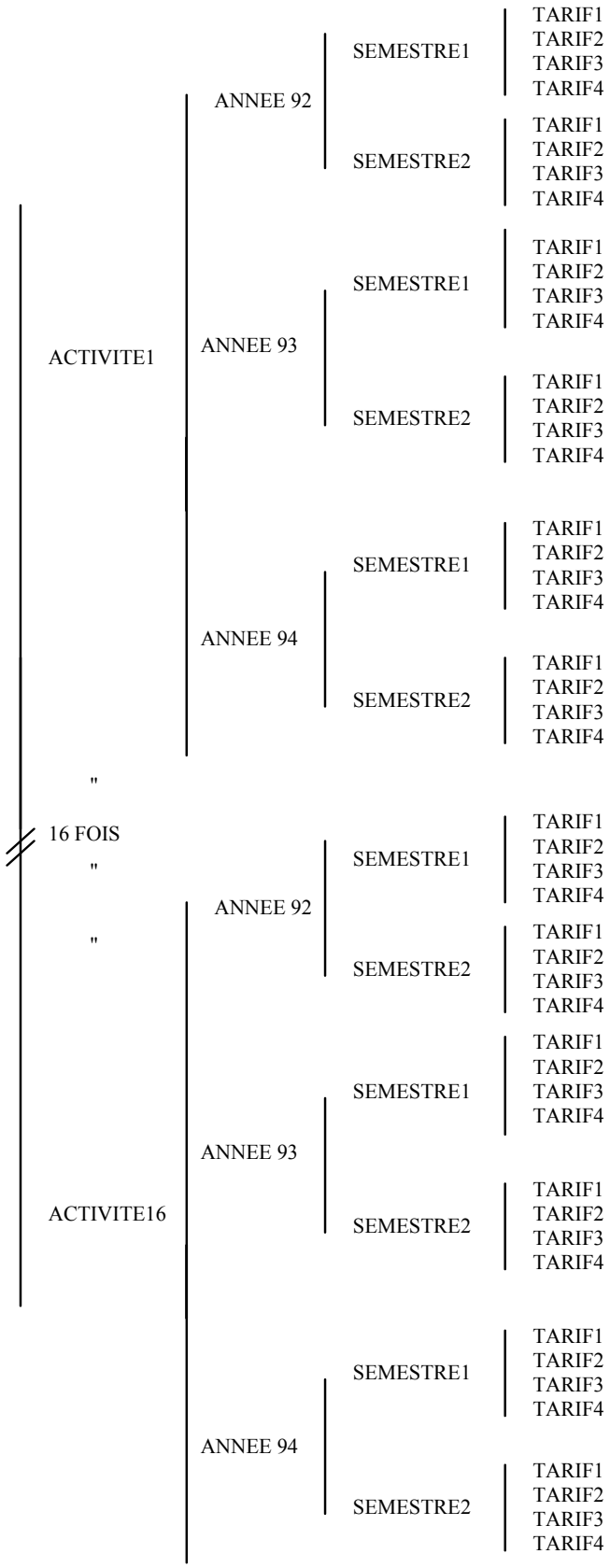
1.2 - Editer la table ainsi triée

II - L'association sportive possède un fichier du nombre de ses adhérents par activité et par période, appelé ADHERENT. Ce fichier est stocké sur bande et il est trié en séquence descendante sur l'année, le semestre, et le code activité. Sa structure est la suivante :

- code activité 2 caractères numériques
- année 2 caractères numériques
- semestre 1 caractère numérique
- filler 1 caractère alphanumérique
- nb adhérents tarif1 3 caractères numériques
- filler 1 caractère alphanumérique
- nb adhérents tarif2 3 caractères numériques
- filler 1 caractère alphanumérique
- nb adhérents tarif3 3 caractères numériques
- filler 1 caractère alphanumérique
- nb adhérents tarif4 3 caractères numériques.

Les différents tarifs correspondent à la situation de chaque adhérent. Les tarifs 1 et 2 sont applicables aux habitants de la commune, les tarifs 3 et 4 aux personnes extérieures. Les tarifs 1 et 3 s'appliquent aux personnes qui n'exercent qu'une activité sportive, les tarifs 2 et 4 à celles qui pratiquent plus d'une activité au cours d'un même semestre.

2.1 - A partir de ce fichier, et en reprenant les résultats du I, constituer, pour les années 92, 93, 94 une table indexée se présentant comme suit :



Contrôles préalables à effectuer :

- vérifier la validité du code activité par rapport à P-TABSPORT
- vérifier la validité du code semestre (valeurs possibles : 1 ou 2)

2.2 - Editer la nouvelle table ainsi créée

III - Sélectionner les activités dont la variation du 2ème semestre par rapport au 1er semestre en 1993, est supérieure ou égale à 50%, et cela pour les adhérents qui ne pratiquent qu'une activité, qu'ils habitent la commune ou non.