INITIATION A L'ALGORITHMIQUE : STRUCTURES DE DONNEES ELEMENTAIRES

CHAPITRE 4 : TABLEAUX ET CHAINES DE CARACTERES

INTRODUCTION

Un tableau est une variable qui permet de rassembler sous un même nom un nombre fini d'éléments ayant tous le même type.

Le type qui caractérise une telle variable est également appelé tableau. Lorsqu'un tableau est défini, le nombre d'éléments qu'il peut contenir doit être précisé et il ne pourra pas être changé par la suite. Ce nombre d'éléments est appelé la taille ou la capacité du tableau.

Les éléments du tableau sont accessibles à partir de leur(s) indice(s) : c'est pourquoi on les appelle aussi de variables indicées. Lorsqu'un élément particulier d'un tableau est désigné en précisant un seul indice on parle de tableau à une dimension. S'il est nécessaire de préciser plusieurs indices pour accéder à une donnée, on parlera de tableau à plusieurs dimensions.

Intérêt des tableaux

L'utilisation des tableaux permet :

- d'effectuer facilement des traitements collectifs sur des ensembles de données (recherche, tri, calcul de moyennes, opérations statistiques, …)
- d'avoir des programmes plus lisibles et plus compact

Limites

- la taille du tableau est limitée dès la compilation.
- les données du tableau sont non persistantes

BIENVENU KENFACK

I- TABLEAU A UNE DIMENSION (VECTEUR)

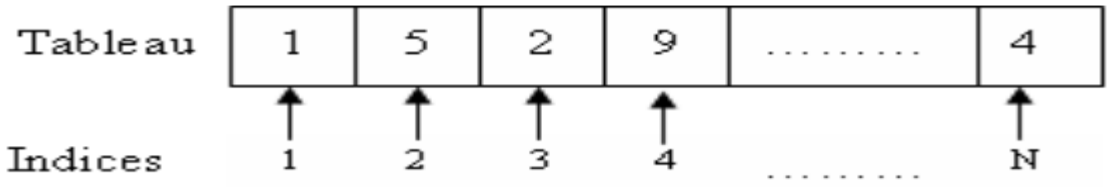
Un tableau à une dimension est un tableau dont les éléments sont accessibles (en lecture ou écriture) en utilisant un seul indice (ou index). Le type des indices est généralement un intervalle sur un type entier.

Un tableau est donc caractérisé par :

- le type des éléments qu'il contient ;
- les indices (ou index) valides pour accéder à ces éléments.

En mémoire centrale, les éléments d'un tableau occupent un espace contiguë. Ainsi, l'élément d'indice 1 est suivie directement de l'élément d'indice 2, qui a son tour est suivie directement de l'élément d'indice 3, etc.

Généralement, on visualise un tableau avec la notation ci-dessous :



1- Déclaration

Un tableau doit être déclaré en précisant le nombre de ses éléments ainsi que le type de ceux-ci. La syntaxe de déclaration est la suivante :

Variable nom_tableau : tableau [inf… Sup] de Type;

- inf et sup représentent respectivement les bornes inférieurs et supérieurs du tableau ;
- Type représente le type des éléments que va contenir le tableau. Il peut prendre les valeurs : entier, réel, caractère, chaine, booléen, etc.

Remarque : L'indice qui sert à repérer les éléments du tableau peut être exprimé directement comme un nombre (valeur littérale) mais aussi comme une variable ou une expression calculée.

La valeur d'un indice doit toujours être un entier compris entre inf et sup.

Exemple : la déclaration d'un tableau de 10 entiers se fera comme suit :

Variable tab : tableau [1..10] de entier ;

Dans cette déclaration, tab est une variable qui pourra contenir 10 entiers.

Remarque : en pratique on choisit souvent inf = 1 comme dans l'exemple ci-dessus.

BIENVENU KENFACK

2- Opérations sur un tableau

Un seul opérateur est défini sur les tableaux. Il s'agit de l'opérateur d'indiçage qui permet de sélectionner un élément du tableau en fonction de son indice. Cet élément se comporte exactement comme une variable de même type que les éléments du tableau. Ainsi, on utilise la notion **t[i]** pour accéder à l'élément du tableau en position **i.**

Exemple

```
Variable vect : Tableau [1 .. 10] de reel ;
i←5 ;
vect[1] ←1 ; // on affecte la valeur 1 à la première case du tableau
Lire(vect[2]) ; // la deuxième case du tableau va contenir la valeur saisie par l'utilisateur
vect[i] ←3 ;//dans la case numéro 5 (ici i =5) on va affecter la valeur 3
vect[i+2] ←4; // OK vect[7] = 7
vect[3] ←vect[5] ;// OK vect[3] = 2
vect[-1] ←5 ; // Erreur : -1∉1..10 !
vect[11] ←6 ; // Erreur : 11∉1..10 !
Ecrire(vect[i]); // Cette instruction va afficher la valeur 3
```

3- Parcourt d'un tableau

Supposons que l'on veille saisir les valeurs d'un tableau de 10 entiers. Une solution simple peut-être la suivante : *variable notes : Tableau [1.. 10] de entier*

```
Lire (notes[1]) ;
Lire (notes[2]) ;
          :
Lire (notes[10]) ;
```

Si le tableau notes ci-dessus comportait un nombre important d'éléments, par exemple 100, il serait laborieux pour le programmeur d'utiliser la méthode ci-dessus et le code deviendra très vite illisible.

Une solution efficace à ce problème passe par l'utilisation des boucles.

```
Variable i : entier ;
pour i de 1 à 10 faire
   Ecrire ("entrer la note", i) ;
   Lire (notes[i]) ;
Fin pour
```

Dans le code ci-dessus, i va successivement prendre les valeurs entre 1 à 10 et à chaque fois, la valeur saisie sera envoyée dans la case numéro i.

BIENVENU KENFACK

De même, pour afficher l'ensemble des éléments du tableau on utilisera une boucle :

```
Pour i de 1 à 10 faire
    Ecrire ("note", i,"=", note[i]) ;
Fin pour
```

Remarque :

De façon générale, lorsqu'on veut effectuer un traitement global sur un tableau, il faut penser à l'utilisation d'une boucle (pour, tantque) pour parcourir l'ensemble des éléments du tableau. Par contre lorsqu'on veut traiter de façon isolé une valeur du tableau, on l'utilise comme une variable ordinaire en précisant son indice dans le tableau.

Exemple récapitulatif

```
Algorithme   moyenne
Variable     notes : tableau [1..10] de réel ;
             i : entier ;
             somme, moy : réel ;
Début
      /*lecture des notes */
      Pour i de 1 à 10 faire
            Ecrire ("entrer la note", i) ;
            Lire (note[i]) ;
      Fin pour
      /*calcul de la moyenne */
       Somme  ←——  0 ;
       Pour i de 1 à 10 faire
           Somme  ←——  somme + note[i];
       Fin pour
       Moy  ←——  somme / 10 ;
       /*affichage de la moyenne calculée */
       Ecrire ("moyenne=", moy) ;
Fin
```

Exercices

1. Compléter le programme ci-dessus afin qu'il affiche aussi la liste des notes plus petites que la moyenne.
2. Ecrire un programme qui demande à l'utilisateur de saisir les valeurs d'un tableau de 10 entiers. Le programme affiche ensuite le nombre d'éléments pair du tableau.

BIENVENU KENFACK

II- ALGORITHMES DE RECHERCHE

Dans un algorithme de recherche on parcourt le tableau dans le but de retrouver les données satisfaisant des conditions particulières. Nous présentons ci-dessous la recherche séquentielle et la recherche dichotomique.

1- Recherche séquentielle

Principe : Pour vérifier l'existence d'un élément dans un tableau, on effectue le parcours de celui-ci et on s'arrête dans deux cas :

- dès qu'on a trouvé l'élément recherché ;
- lorsqu'on a traité le dernier élément.

Algorithme

```
Algorithme  recherche_seq1
Variable      t : tableau [1..10] de entier ;
              val, i : entier ;
              trouver : booléen ;
Début
        /* lecture des valeurs du tableau */
        Pour i   de  1 à 10 faire
              Ecrire ("entrer la valeur", i) ;
              Lire (t[i]) ;
        Fin pour
        Ecrire ("entrer la valeur recherchée") ;
        Lire (val) ;
        trouver    ←——    faux ;
        i ← 1 ;
        /* recherche séquentielle */
        Tantque ((i <= 10) et (trouver = faux)) faire
           Si  (t[i] = val) alors
                    trouve    ←——    vrai ;
            Finsi
           i ← i +1 ;
         FinTantque
         Si (trouve = vrai) alors
            Ecrire ("élément présent") ;
         Sinon
            Ecrire ("élément non présent") ;
         Fin si
Fin
```

BIENVENU KENFACK

Remarque : L'algorithme ci-dessus doit parcourir tout le tableau lorsque l'élément recherché est absent. Dans le cas d'un tableau trié, la recherche séquentielle doit s'arrêter plus tôt, lorsque l'élément recherché ne peut plus être trouvé dans la partie non encore explorée du tableau.

Exercice : modifier l'algorithme ci-dessus en prenant en compte la remarque précédente, dans le ca de recherche dans un tableau trié en ordre croissant.

2- Recherche dichotomique

L'algorithme de recherche dichotomique ne fonctionne que sur des tableaux triés.

Principe

- On divise le vecteur en 2 parties sensiblement égales, et avec l'aide d'une comparaison du critère de recherche avec l'élément médian, on élimine la partie du tableau qui ne peut pas contenir la valeur recherchée
- La partie non éliminée est à son tour divisée en deux et ainsi de suite, jusqu'à ce qu'on obtienne un vecteur à un seul élément. Si cet élément correspond à la valeur recherchée, on a trouvé l'élément sinon l'élément est absent.

Algorithme

```
Algorithme   dichotomique
 Var     t : tableau [1..10] de entier ;
          i, inf , val, sup, med : entier ;
Début
           /* lecture des valeurs du tableau */
          Pour i   de  1   à  10  faire
              Ecrire ("entrer la valeur", i) ;
              Lire (t[i]) ;
          Fin pour
          Ecrire ("entrer la valeur à rechercher") ;
          Lire (val) ;
          inf   ←——      1 ;
          sup   ←——     10 ;
          Tant que (inf < sup) faire
               med   ←——      (inf + sup) div 2 ;
               Si (t[med] < val) alors
                  inf   ←——  med +1 ;
               Sinon
                  sup       ←——  med ;
               Finsi
          Fin tantque
```

BIENVENU KENFACK

```
        Si (t[inf] = val) alors
            Ecrire ("élément présent") ;
          Sinon
                Ecrire ("élément abscent") ;
          Finsi
Fin
```

Remarque : On peut arrêter l'algorithme plutôt dès qu'on a trouvé l'élément recherché.

Exercice : Modifier l'algorithme ci-dessus pour que la recherche s'arrête dès que l'élément recherché a été trouvé.

Exercice1

- Ecrire un algorithme qui permet de saisir les valeurs d'un tableau de 10 entiers. Puis calcul et affiche la somme de tous les nombres plus petit que 11.
- Ecrire un programme qui vérifie si un tableau de 10 entier est trié ou pas par ordre croissant.

III- ALGORITHMES DE TRI

Dans de nombreux problèmes il est fréquent de vouloir organiser les données suivant une relation d'ordre. Un algorithme de tri est un algorithme qui permet de classer les éléments d'un ensemble suivant une relation d'ordre (croissant, décroissant, …). Il existe plusieurs stratégies possibles pour trier les éléments d'un tableau ; nous en verrons trois dans ce chapitre. D'autres algorithmes de tri seront étudiés en TP.

BIENVENU KENFACK

1- Tri par sélection

a- Principe

La technique du tri par sélection est la suivante : on met en bonne position l'élément numéro 1, c'est-à-dire le plus petit. Puis en met en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier.

b- Illustration

Considérons le tableau suivant :

45	122	12	3	21	78	64	53	89	28	84	46

On commence par rechercher, parmi les 12 valeurs, quel est le plus petit élément, et où il se trouve. On l'identifie en quatrième position (c'est le nombre 3), et on l'échange alors avec le premier élément (le nombre 45).

Le tableau devient ainsi :

3	122	12	**45**	21	78	64	53	89	28	84	46

On recommence à chercher le plus petit élément, mais cette fois, seulement à partir du deuxième (puisque le premier est maintenant correct, on n'y touche plus). On le trouve en troisième position (c'est le nombre 12). On échange donc le deuxième avec le troisième :

3	**12**	**122**	45	21	78	64	53	89	28	84	46

On recommence à chercher le plus petit élément à partir du troisième (puisque les deux premiers sont maintenant bien placés), et on le place correctement, en l'échangeant, ce qui donnera in fine :

3	12	**21**	45	**122**	78	64	53	89	28	84	46

Et cetera, et cetera, jusqu'à l'avant dernier.

BIENVENU KENFACK

c- Algorithme

```
Algorithme      tri_selection
Variable    vec : tableau [1..10] de reel ;
            aux : reel ;
            i, j, N : entier ;
Début
            /* saisie des valeurs du tableau */
            Pour i   de  1 à 10 faire
                    Ecrire ("entrer la valeur", i) ;
                    Lire (vec[i]) ;
            Finpour
            /* algorithme de tri par sélection */
            DEBUT
                N ← 10 ;
                i←1
                Tantque (i<N) Faire   //boucle 1
                      j←i+1
                      min←i
                      Tantque (j<=N) Faire   //boucle 2
                         si (vec[j] < vec[min]) Alors
                            min←j
                         Finsi
                         j←j+1
                       Fintantque
                      Si (min ≠ i) Alors
                        //échanger vec[i] et vec[min]
                         aux <-- vec[i];
                         vec[i] <-- vec[min];
                         vec[min] <-- aux;
                      Finsi
                      i←i+1
                 Fintantque
            /* on affiche le tableau trié */
            Ecrire ("tableau trié") ;
            Pour i    de  1 à 10 faire
                 Ecrire (vec[i]) ;
            Finpour

Fin
```

BIENVENU KENFACK

2- Tri par insertion

a- Principe

Chaque valeur V[i] du tableau est insérée à la bonne place parmi les valeur déjà ordonnées du sous vecteur V[1.. I-1]. Au départ le sous vecteur est réduit à V[1] et on a I=2.Chaque insertion se fait par comparaison et décalage successif et a pour conséquence d'agrandir le sous vecteur ordonné d'un élément. La dernière insertion est celle de V[n].

b- Algorithme (tri croissant d'un tableau d'entiers)

```
Algorithme      tri_insertion
Variable    v : tableau [1..10] de reel ;
            val : reel ;
            i, k, n : entier ;
Début
            /* saisie des valeurs du tableau */
            Pour i   de  1 à 10 faire
                    Ecrire ("entrer la valeur", i) ;
                    Lire (v[i]) ;
            Finpour
            /* algorithme de tri par insertion */
            N ←——    10 ;
            Pour i de  2 à n faire
                    val ←——    v[i] ;
                    k   ←——    i – 1 ;
                    Tantque  (k ≥ 1) et (v[k] > val) faire
                            V[k+1] ←——    v[k] ;
                            K      ←——    k – 1 ;
                    FinTantque
                    V[k+1]    ←——    val ;
            Finpour
               /* on affiche le tableau trié */
               Ecrire ("tableau trié") ;
               Pour i    de  1 à 10 faire
                   Ecrire (v[i]) ;
               Finpour
    Fin
```

Exercices

- Modifier l'algorithme ci-dessus pour que le tri se fasse par ordre décroissant.
- Ecrire un programme qui fusionne 2 tableaux triés de 5 entiers en 1 tableau trié de 10 entiers

BIENVENU KENFACK

c- Tri par échange ou tri à bulle

a- Principe

On effectue des passages successifs sur le vecteur V en examinant les éléments par paire (V[k], V[k+1]) et en échangeant les valeurs si V[k] est plus grand que V[k+1]. Chaque passage s'arrête au niveau du dernier passage précédant. Le premier passage s'arrête au niveau du dernier élément du tableau. Les éléments les plus grands « remontent » ainsi peu à peu vers les dernières places, ce qui explique la dénomination de « tri à bulle ».

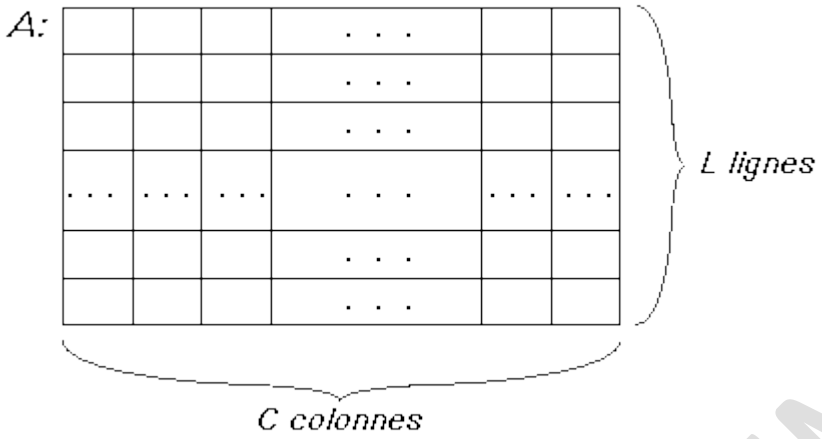
b- Algorithme (tri croissant d'un tableau d'entiers)

```
Algorithme    tri_bulle
 Variable          V : tableau [1..10] de entier ;
                   I, n, k, val: entier;
 Début
          /* Lecture des éléments du tableau */
           n  ←——    10 ;
           écrire ("Entrer les valeurs du tableau") ;
           pour i de 1 à  n faire
               lire (V[i]) ;
           finpour
           /* tri du tableau par échange */
           i  ←——   n ;
           Tantque  (i ≥ 2) faire
                   Pour k  de  1 à (i - 1) faire
                          Si  (V[k] > V [k + 1]) alors
                              Val  ←——   V[k];
                              V[k]  ←——     V[k+1] ;
                              V[k+1]  ←——   val ;
                          Finsi
                    Finpour
                    i ← i -1 ;
           FinTantque
           /* affichage du tableau trié */
           Pour i   de   1 à n faire
              Ecrire (V[i]) ;
           Finpour
   Fin
```

BIENVENU KENFACK

IV- TABLEAU A DEUX DIMENSIONS

Un tableau à deux dimensions est un tableau dont les éléments sont accessibles en utilisant deux indices. Le premier indice est souvent appelé indice des lignes et le deuxième indice appelé indice des colonnes. Un tableau à 2 dimensions qui possède « L » lignes et « C » colonnes aura un total de L x C éléments.



1. Déclaration

La déclaration d'un tableau à deux dimensions se fait de la façon suivante :

variable nom_tableau : tableau [1..L, 1..C] de type ;

- L et C représentent respectivement le nombre de lignes et de colonnes de ce tableau.

Exemple: variable tab: tableau[1..5, 1..6] de entier;

Dans cet exemple, tab est un tableau à deux dimensions de 5 lignes et 6 colonnes.

2. Accès aux éléments

Si **t** est un tableau à deux dimensions alors on accède aux éléments du tableau par la syntaxe **t[i, j]** où **i** représente un numéro de la ligne et **j** un numéro de la colonne.

T[i, j] est donc une variable et peut être manipuler comme telle.

Exemple

```
T[2, 3]  ←——   10 ;
T[1, 1]  ←——   8+ t[2, 1] ;
 Lire (t[1, 3]) ;
 Ecrire(t[1, 2]) ;
```

BIENVENU KENFACK

3. Parcours

De façon générale on va utiliser des boucles imbriquées pour parcourir un tableau à deux dimensions.

```
Exemple
  Algorithme    parcours
  Variable   notes : tableau [1..3, 1..4] de entier ;
                I, j : entier ;
  Début
          /* lecture des éléments du tableau ligne par ligne */
          Pour i   de 1 à 3 faire
               Pour j   de 1 à 4 faire
                    Ecrire ("entrer l'élément [", i, ",", j,"]") ;
                     Lire (notes[i, j]) ;
               Finpour
          Finpour
        /* affichage du tableau ligne par ligne */
       Ecrire ("contenu du tableau") ;
       Pour i   de 1 à 3 faire
            Pour j  de 1 à 4 faire
                 Ecrire (notes [i, j]) ;
            Finpour
       Finpour
  Fin
```

4. Exercices d'application

1. Modifier le programme ci-dessus de sorte que l'affichage des éléments se fait plutôt colonne par colonne.
2. Ecrire un programme qui lit les valeurs de deux matrices carrées 4*4 de réels, puis calcule et affiche la somme de ces deux matrices.

V- LES CHAINES DE CARACTERES

1. Définition et syntaxe

Une chaine de caractère est une séquence finie de caractères.

Chaque caractère de la chaine est accessible à partir de sa position dans la chaine comme dans le cas des tableaux. En fait, une chaine de caractères peut être considérée comme un tableau de caractères à une dimension.

Dans le code, les valeurs littérales de type chaine sont représentées entre simple quotte ou bien entre double quotte. *Exemple : 'bonjour ', "bonjour"*

BIENVENU KENFACK

Opérations sur les chaines

- *ch := ''*; signifie que ch contient une chaine vide (de longueur 0).
- *cha1 ← ch2* ; signifie qu'on recopie la valeur de ch2 dans ch1 ;
- *longueur (ch)* ; fonction qui retourne la longueur courante de la chaine ch, c'est-à-dire le nombre de caractère qu'il contient.
- On accède à un caractère de la chaine avec la syntaxe ch[i] où i est la position de ce caractère dans la chaine ch ;
- *X ← ch1 + ch2* ; représente la concaténation des chaines de caractères.
- Il est possible de comparer les chaines de caractères avec les opérateurs =, <>, <, >, <=, >=.

La comparaison se fait suivant l'ordre lexicaux-graphique du code ASCII.

Exemple : 'bijou' > 'bidon' , 'Bonjour' != 'bonjour'

2. Exercices d'applications:

a- Ecrire un programme qui affiche le nombre d'occurrence du caractère 'a' contenu dans un mot saisie par l'utilisateur.

b- Ecrire un programme qui lit un mot puis affiche toutes les voyelles contenus dans ce mot.

c- Ecrire un programme qui affiche le nombre de mot contenu dans une phrase saisie par l'utilisateur (on supposera que les mots sont séparés par des espaces).

d- Ecrire un programme qui demande un mot à l'utilisateur puis le transforme en majuscule et affiche le résultat.

e- Ecrire un programme qui prend en entrée deux mots (mot1 et mot2) puis affiche le message « oui » si mot1 est une sous chaine de mot2 et « non » dans le cas contraire.

f- Ecrire un programme qui prend en entrée un mot et dit si ce mot est *palindrome* ou pas. Un mot *palindrome* est un mot qu'on peut lire indifféremment de la gauche vers la droite ou inversement (aba, anna).

BIENVENU KENFACK

CHAPITRE 5

TYPES ENUMERES ET STRUCTURES

INTRODUCTION

Considérons une application destinée à gérer la scolarité d'un établissement scolaire. Une telle application doit pouvoir manipuler des entités complexes telles que les étudiants, les filières, les semestres, …ainsi que les relations entre ces entités. L'utilisation des types de base pour résoudre ce problème amènera à créer un très grand nombre de variables et introduirait un plus haut niveau de complexité dans ce système.

En programmation, il est possible pour le programmeur de définir de nouveaux types de données plus adaptés au problème qu'il cherche à résoudre. Ces nouveaux types, construits généralement à partir des types de base sont appelés types définis par l'utilisateur, les plus connus étant les structures et les énumérations.

I- TYPES ENUMERES ET TYPES INTERVALLE

Une énumération (ou type énuméré) est un type dont le domaine de valeurs est défini par le programmeur. Ainsi, le type énuméré va nous éviter de définir plusieurs valeurs pour représenter certaines constantes pourtant sémantiquement liées.

Exemple : pour représenter les différents jours de la semaine dans un programme on peut définir 7 constantes, à raison d'une constante pour représenter un jour de la semaine. Une approche plus élégante serait de définir un type JOUR qui pourra prendre ses valeurs dans l'ensemble des 7 jours de la semaine.

1. Définition d'un type énuméré

Syntaxe de déclaration :

Type Nom-type = (element1, element2, … elementN)

Pour définir un type énuméré, on précise son nom et on fait suivre la liste des valeurs constantes entre parenthèse.

Exemple :

- Type Jour = (Lundi, Mardi, Mercredi, jeudi, Vendredi, Samedi, Dimanche)
- Type Couleur = (VERT, ROUGE, JAUNE, BLEU, VIOLET)

BENJAMIN KENFACK

2. utilisation

On déclara une variable de type énuméré comme toute autre variable :

Exemple : variable c : Couleur ;

3. Type intervalle

Un type intervalle est un type dont les variables prennent leurs valeurs dans une portion de l'intervalle des valeurs d'un autre type (entier, énuméré ou caractère).

Exemples :

- Note=0..20 ; // ici on définit une note comme étant un entier compris entre 0 et 20.
- JOUR_OUVRABLE=lundi..vendredi ; //un jour ouvrable est compris entre lundi et vendredi.

II- LES STRUCTURES (TYPES ENREGISTREMENT)

Les enregistrements sont des structures de données dont les éléments peuvent être de type différent se rapportant à une même entité. Les éléments qui composent un enregistrement sont appelés champs. Le type d'un enregistrement est appelé **type structuré,** c'est pourquoi les enregistrements sont aussi parfois appelés **structures**.

L'intérêt des enregistrements est de pouvoir structurer très proprement les informations qui vont ensemble, de les recopier facilement, de les passer en paramètre et en valeur de retour de sous-programmes.

1. Déclaration d'un type structuré

Pour créer des enregistrements, il faut au préalable déclarer un nouveau type qu'on appelle type **structuré**.

Les variables de type structuré sont appelés **enregistrement.**

a. Syntaxe

```
Type    Nom_type = Enregistrement
          Nom_champ1 = type_champ1 ;
          Nom_champ2 = type_champ2 ;
          :
          Nom_champN = type_champN ;
Fin enregistrement
```

BIENVENU KENFACK

- Le type d'un champ peut-être : un type de base, un tableau ou un enregistrement
- Chaque variable de ce type structuré aura N valeurs, chaque valeur correspondant à un champ de la structure.

Exemple

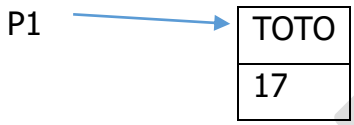
```
Type  personne = Enregistrement
        nom : chaine ;
        age : entier ;
Fin enregistrement
```

Dans cet exemple on crée le nouveau type de donnée appelé Personne. Une variable de type Personne aura deux valeurs : une valeur pour le nom et une valeur pour l'âge.

On peut donc déclarer les variables de type personne comme pour tout autre type avec la syntaxe suivante.

Variable P1, P2 : personne ;

Une variable de type personne contient deux informations un Nom et un Age.



2. Opérations sur un enregistrement

Les champs d'un enregistrement sont accessibles par leur nom grâce à l'opérateur "."

Syntaxe :

nom_variable.nom_champ

Exemple

Variable p1, p2 : Personne ;

- p1.nom permet d'accéder au champ nom de la variable p1
- p1.age permet d'accéder au champ age de la variable p1

Remarque :

- Il est possible d'effectuer l'affectation entre deux enregistrements : dans ce cas une affectation entre champs de même nom est effectuée entre les deux enregistrements.

BIENVENU KENFACK

Exemple

Le programme ci-dessous saisit les informations de deux personnes puis affiche la différence d'âge entre ces deux personnes.

```
Algorithme    diff_age ;
 Type personne = Enregistrement ;
                 nom : chaine ;
                 age : entier ;
  Fin enregistrement
  Variable  p1, p2 : personne ;
             diff : entier
Début
    Ecrire ("entrer le nom et l'âge de la première personne") ;
    Lire (p1.nom, p1.age) ;
    Ecrire ("entrer le nom et l'âge de la deuxième personne") ;
    Lire (p2.nom, p2.age) ;
    Si (p1.age < p2.age) alors
       diff ←———          p2.age – p1.age ;
    Sinon
       diff ←———        p1.age – p2.age ;
    Finsi
    Ecrire ("différence d'âge = ", diff) ;
Fin
```

3. Tableau d'enregistrements

Il arrive que l'on veuille faire des traitements sur un ensemble d'enregistrements de même type. Par exemple pour traiter un groupe de 50 personnes, on ne va pas créer 50 variables de type Personne mais plutôt un tableau d'enregistrement qui va contenir les 50 personnes.

Exemple

```
Const  NP = 50 ;
Type   personne = enregistrement ;
       nom : chaine ;
       age : entier ;
Fin enregistrement ;

Variable  groupe : Tableau [1..NP] de personne ;
```

Chaque élément du tableau groupe est un enregistrement de type Personne. Ainsi :

- groupe[2] représente la deuxième personne du tableau ;
- groupe[1].nom représente le nom de la première personne du tableau.

BIENVENU KENFACK

4. L'imbrication d'enregistrements

Supposons que dans notre type personne, nous voulons plutôt la date de naissance au lieu de l'âge. Une date est constituée de 3 informations (jour, mois, année) indissociable. Une date correspondant donc à une entité du monde réel qu'on peut représenter par un type enregistrement à 3 champs.

On peut utiliser ce nouveau type dans la déclaration du type Personne. On obtient donc une *structure imbriquée.*

Exemple

```
Type   Date = Enregistrement ;
                jour : integer ;
                mois : string ;
                année : integer ;
        FinEnregistrement ;

        Personne = Enregistrement ;
                nom : string ;
                date_naiss: Date;
        FinEnregistrement;

Variable p: Personne:
p.nom = 'toto';  // on affecte la chaine  'toto' au champ nom de la variable p
P.date_naiss.jour = 10; //on affecte 10 au champ jour du champ date_naiss de la variable p
```

BIENVENU KENFACK

Exercice d'application

On se propose de gérer les comptes client d'une banque.

Chaque compte est identifié par un numéro de compte, le solde du compte (supposé toujours positif) et le nom du client possédant ce compte. Pour simplifier, on suppose aussi que deux clients ne peuvent pas avoir le même nom. Par ailleurs le nombre maximal de compte que peut gérer la banque est fixé à 100. On veut écrire un programme permettant de répondre aux questions suivantes :

- Saisie de 100 comptes client ;
- Afficher la liste de tous les comptes ;
- Afficher la liste des comptes dont le solde est inférieur à 100000 ;
- Effectuer un virement de 200 000 du compte 4 vers le compte 9, si cette opération est possible.

TAF :

1. Déclarer les structures de données permettant de résoudre ce problème ;
2. Proposer un algorithme pour la résolution de ce problème ;
3. Implémenter l'algorithme précédant en langage C ;
4. Proposer une amélioration de ce programme avec le menu suivant :

Menu des opérations

1) créer un compte
2) débiter un compte
3) créditer un compte
4) effectuer un virement
5) afficher tous comptes
6) quitter le programme

BIENVENU KENFACK