1. Objectif

Le but de ce TP est de manipuler les structures de données.

2. Introduction

En langage C le programme principal et les sous programmes sont définies comme des fonctions.
Pour présenter l'intérêt de ce paragraphe, on va partir de l'exemple de la gestion des notes.

On a proposé alors la structure de données suivante :

- Un tableau de n éléments comportant les numéros de CIN des étudiants TCIN.
- Un tableau de n éléments comportant les notes des étudiants dans la première épreuve TN1.
- Un tableau de n éléments comportant les notes des étudiants dans la deuxième épreuve TN2.
- Un tableau de n éléments comportant les notes des étudiants dans la troisième épreuve TN3.
- Un tableau de n éléments comportant les moyennes des étudiants TMOY.

On suppose que chaque étudiant a passé 3 matières.

Avec les structures vues jusque là (vecteurs), on ne peut pas espérer mieux. Malheureusement, elle peut poser parfois certains problèmes :

- Nombre important de vecteurs en mémoire ce qui entraîne une lourdeur au niveau du fonctionnement de la machine.
- Manipulation pouvant devenir difficile avec ce nombre important.
- Risque d'erreurs lors des calculs : la moindre distraction, ou le moindre oubli d'initialiser un indice, entraîne systématiquement une erreur au niveau du traitement.

Comme solution, on a proposé alors une structure de données capable de regrouper toutes les données de *même type* (notes, moyenne) pour un étudiant donné : les tableaux multidimensionnels ou Matrices.

Mais l'inconvénient de cette solution réside dans le fait que tous les éléments doivent être de même type. Sinon, on est obligé de séparer dans des tableaux différents les groupes de même type et on va revenir aux problèmes mentionnés ci-dessus.

La question qu'on se pose alors est de trouver une structure capable de regrouper des informations non nécessairement de même type dans une même variable ; une telle structure existe et elle appelée le *Type Enregistrement*.

❒ Définition

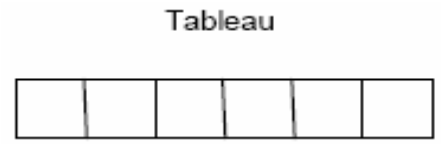
Le type Enregistrement est une structure de données permettant de regrouper un ensemble de données de types différents sous une même variable.

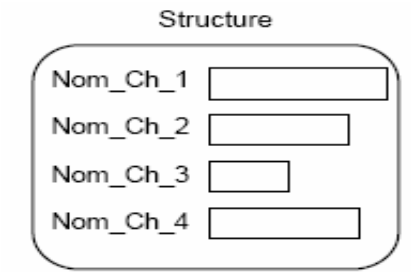
Remarques

(i) Un enregistrement est composé d'une ou plusieurs variables non nécessairement de même type, appelées *Champs* ou *Rubriques*.
(ii) Le type Enregistrement est dit également le type *Structure.*

3. Manipulation de structure

Un enregistrement, appelé **structure** En langage C, est une variable complexe qui permet de désigner sous un seul nom un ensemble de valeurs pouvant être de **type différent.**





Chaque élément de la structure est nommé **champ**

- L'accès à un champ se fait par son nom dans la structure

❒ Déclaration

Lors de la déclaration de la structure, on définit un modèle où on indique les champs de la structure, c'est-à-dire le type et le nom des variables qui la composent :

Syntaxe:

```
struct   <Nom_Structure> {
        <type_champ1> <Nom_Champ1>;
        <type_champ2> <Nom_Champ2>;
        <type_champ3> <Nom_Champ3>;
        ...
};
```

REMARQUES

- Le nom des champs répond aux critères des noms de variables.
- Deux champs ne peuvent avoir le même nom.
- Les données peuvent être de n'importe quel type hormis le type de la structure dans laquelle elles se trouvent.

Exemple:

```
struct  Point {
        int X;
        int Y;
};
```

- La déclaration d'une structure ne fait que donner l'allure de la structure, c'est-à-dire en quelque sorte une définition d'un type de variable complexe.
- La déclaration ne réserve donc pas d'espace mémoire pour une variable structurée (variable de type structure).
- Il faut donc définir une (ou plusieurs) variable(s) structurée(s) après avoir déclarée la structure...

❒ **Initialisation d'une structure**

Lors de sa déclaration on peut initialiser une structure en indiquant entre {} la liste des valeurs séparées par des virgules.

Chaque valeur étant une constante ayant le type du champ correspondant.

Exemple

```
struct UnEtudiant {
        short niveau;
        float moyenne;
};
Struct  UnEtudiant  Etudiant1 = {2, 10.9};
struct  UnEtudiant  Etudiant2 =  {2, 12.5};
```

❒ **Utilisation d'une structure**

Chaque variable de type structure possède des champs repérés avec des noms uniques. Toutefois le nom des champs ne suffit pas pour y accéder étant donné qu'ils n'ont de contexte qu'au sein de la variable structurée...

Pour accéder aux champs d'une structure on utilise l'opérateur de champ (un simple point) placé entre le nom de la variable structurée et le nom du champ.

Syntaxe:

```
<Var_Structure>.<Nom_Champi>;
```

Ainsi, pour affecter des valeurs à la variable **Etudiant1,** on pourra écrire:

```
Etudiant1.niveau = 2;
scanf ("%f", &Etudiant1.moyenne) ;
printf ("%f", Etudiant1.moyenne) ;
```

Il est clair, qu'on peut effectuer des affectations entre deux variables de même type de structure :

- Soit d'une façon individuelle (champs par champs), sur chacun de leurs champs,
- Soit d'une manière globale sur toute la structure.

Exemple

Affectation individuelle :

```
Etudiant1.niveau =  Etudiant2.niveau;
Etudiant1.moyenne =  Etudiant2.moyenne;
```

Affectation Globale :

```
Etudiant1 = Etudiant2;
```

❒ Structure comportant des tableaux

- Une structure peut contenir des champs de type chaîne de caractères ou bien de type tableau.

Exemple

```
struct UnEtudiant {
     char nom [30];
     char prenom [30];
     float  notes [4];
     short niveau;
     float moyenne;
};
struct  UnEtudiant  Etudiant1,  Etudiant2;
```

Notation:

- **Etudiant1.nom[0] :** désigne le premier caractère du champ **nom** de l'étudiant **Etudiant1.**
- **Etudiant2.notes[3] :** désigne la quatrième note du tableau **notes** de l'étudiant **Etudiant2**.

Initialisation:

- **struct UnEtudiant Etudiant1 = {"Ben Salem", "Ali", {10.0, 12.5, 13.6, 7.5}, 2, 10.9};**
- **struct UnEtudiant Etudiant2 = {"selmi ", "Alia", {12.0, 12.5, 14.0, 11.5}, 2, 12.5};**

Utilisation:

```
strcpy (Etudiant1.nom, "Ben Salem");
Etudiant1.notes[2] = 13.6;
printf ("%f ",  Etudiant2.notes[i]);
```

❒ Structure comportant d'autres structures

Une structure peut contenir aussi des champs de type structure. Cette dernière doit être différente de la première.

```
struct date {
          int jour;
          int mois;
          int annee;
};
```

```
struct UnEtudiant  {
            char nom [30] ;
            char prenom [30]  ;
            struct  date  date_naissance ;
} Etudiant1;
```

***Initialisation*:**

```
struct UnEtudiant Eudiant1 = {"Ali", "Ben
Salem",{1,1,2000}};
```

Utilisation:

```
Etudiant1.date_naissance.jour = 1;
scanf("%d %d %d", &Etudiant1.date_naissance.jour,
&Etudiant1.date_naissance.mois,
&Etudiant1.date_naissance.annee);
```

Les déclarations de types synonymes : typedef

Le langage offre la possibilité de donner un nom à un type. Pour cela, on utilise le mot clé **typedef** suivi d'une construction ayant la même syntaxe qu'une déclaration de variable.

Le programmeur a la possibilité de créer ses propres types: Il suffit de les déclarer en début de programme (après les déclarations des bibliothèques et les **« define »**) avec la syntaxe suivante:

Exemple

```
typedef int entier;  /* on définit un nouveau type
"entier" synonyme de "int" */
typedef int vecteur[3];    /* on définit un nouveau
type "vecteur" synonyme */
/* de "tableau de 3 entiers"    */
typedef float *fpointeur; /* on définit un nouveau
type "fpointeur" synonyme */
                /* de "pointeur sur un réel */
```

Utilisation : La portée de la déclaration de type dépend de l'endroit où elle est déclarée: dans main(), le type n'est connu que de main(); en début de programme, le type est reconnu dans tout le programme.

```
#include <stdio.h>
typedef  int entier;
typedef float point[2];
void main()
{
      entier n = 6;
      point xy;
      xy[0] = 8.6;
      xy[1] = -9.45;
      etc ...
}
```


De plus, il est assez courant d'associer un nom de structure avec l'instruction "*typedef*". Cela évite d'avoir à

écrire le mot "*struct*" devant chaque variable que l'on veut déclarer.

Exemple de déclaration :

```
typedef struct fiche          /* On définit un type struct
*/
{
      char nom[10];
      char prenom[10];
      int age;
      float note;
}
```

Utilisation :

On déclare des variables par exemple :

```
fiche f1,f2;
```

4. Manipulation de tableaux de structures

Etant donné qu'une structure est composée d'éléments de taille fixes, il est possible de créer un tableau ne contenant que des éléments du type d'une structure donnée.

Il suffit de créer un tableau dont le type est celui de la structure et de le repérer par un nom de variable:

Déclaration:

```
struct  <Nom_Structure>
{
      <type_champ1> <Nom_Champ1>;
      …
      <type_champN> <Nom_ChampN>;

};

struct  <Nom_Structure> <Tab_Struct1>[N1], ...,
<Tab_StructN>[Nn];
```

Accès:

- A un élément du tableau :
  <Tab_Struct>[i] (structure N° i)

- A un élément d'une structure :
  <TabStruct>[i].<NomChampj>

```
struct UnEtudiant {
          char nom [30]   ;
          char prenom [30]   ;
            float  notes      [4]   ;
} ;
```

```
struct  UnEtudiant  Etudiants [50];
```

On peut écrire les instructions suivantes :

```
for (i = 0; i < 50; i++)
{
      printf ("Entrez le nom de l 'étudiant n°%d ", i+1);
      scanf ("%s", Etudiants[i].nom);
      printf ("Entrez le prénom de l'étudiant n°%d ", i+1);
      scanf ("%s", Etudiants[i].prenom);
      for (j =0; j < 4; j++)
       {
        printf ("Entrez la note %d de l'étudiant n°%d ",
j+1,i+1);
        scanf ("%f", &Etudiants[i].notes[j]);
       }
 }
```

Les énumérations

Ils permettent une définition pratique d'un ensemble fini de valeurs. Ce n'est pas un type structuré. En fait il s'agit simplement de désigner des entiers par des noms Symboliques :

```
enum jour {lundi,mardi, mercredi, jeudi, vendredi,
samedi, dimanche} x;
enum jour y,z;
```

Les noms symboliques donnés dans la définition d'une énumération sont en correspondance avec les entiers, ainsi, dans l'exemple qui précède, lundi correspond à 0, mardi à 1,... On peut altérer cette correspondance de la façon suivante :

```
enum jour {lundi=2,mardi, mercredi=6,jeudi,
vendredi,samedi,dimanche} x;
```

lundi correspond alors à 2, mardi à 3,mercredi à 6, jeudi à 7...

Exemple :

```
Enum boolean {NO , YES}
Enum boolean x ;
```

```
x = YES ; x aura la valeur 1
```

A chaque valeur définie par enum est associée un entier commençant à zéro. Dans l'exemple précédent 0 est associé à NO et 1 est associé à YES.

5. Les Exercices

5.1 Exercice 1

Soit le modèle de structure suivant :

```
Struct UnePersonne
    {
```

```
        char Nom[30] ;
        char Prenom[30] ;
        int NbEnfants;
        char PrenomEnfants[5][30] ;
}
```

Écrire un programme qui remplit les différents champs entrés au clavier d'une variable structure nommée Personne.

5.2 Exercice 2

Créer une structure « **point** »

```
struct point
    {
        float x ;
        float y ;
    }
```

Saisir 4 points, les ranger dans un tableau puis les afficher.

5.3 Exercice 3

Soit le modèle de structure suivant :

```
struct UnePers {
        char  nom[30];
        char  prenom[30];
        float  age;
};
```

Ecrire un programme qui remplit les champs d'une variable structure de type UnePers à partir du clavier puis afficher les champs d'une variable structure de type UnePers

5.4 Exercice 4

On se propose d'écrire un programme permettant de calculer les moyennes des étudiants d'une classe. On suppose qu'un étudiant admet les données suivantes : un nom (limité à 20 caractères), un prénom, une date de naissance, les notes qu'il a obtenues par matière stockés dans un tableau notes Une note est un réel compris entre 0 et 20.
Une classe d'étudiants admet les données suivantes : le nombre d'étudiants, le nombre de matières, les coefficients des matières et les étudiants appartenant à la classe

- **a-** Définir la structure de données « etudiant »
- **b-** Définir la structure « classe ».
- **c-** Ecrire un programme permettant de lire les données relatives à une classe et celles relatives à
- **d-** chaque étudiant de la classe et de calculer leur moyenne.

5.5 Exercice 5

Soit une équipe de basket caractérisée par un nom d'équipe et 12 joueurs. Chaque joueur admet un nom, une date de naissance et le nombre total de but marqué.
Une date de naissance est composée du jour, du mois et de l'année.

Ecrire un programme qui permet de

- **a-** Saisir 3 équipes.
- **b-** Calculer le nombre de but marqués pour une équipe donnée (la recherche se fait selon le nom de l'équipe)
- **c-** Donner le nom du joueur qui a marqué le plus de but pour une équipe donnée.
- **d-** Donner le nom du joueur le plus vieux.