Les structures

➢ Définitions et syntaxe

Structure est le nom donné en C et C++ à ce qu'on appelle enregistrement en algorithmique.
Ce nom peut porter à confusion car **les structures ne représentent qu'un sous-ensemble des structures de données**. Les tableaux sont aussi des structures de données mais ce ne sont pas des structures. Nous verrons l'année prochaine d'autres type de structures de données (comme les listes chaînées).

Une structure (au sens enregistrement) permet de désigner sous un seul nom un ensemble de valeurs pouvant être de type différent. L'accès aux éléments d'une structure (appelés champs) ne se fait pas par un indice contrairement à l'accès aux éléments d'un tableau, mais par le nom du champ (précédé du nom de la structure et d'un point).

☝ **Comme en algorithmique, il faut bien distinguer le type de structure de la structure elle-même.**
Une structure est une variable dont on a préalablement défini le type.

Ainsi, pour déclarer une structure, deux étapes sont nécessaires:
- définir[1] le type de structure (appelé aussi modèle de structure, correspondant au type enregistrement)
- déclarer une ou plusieurs variables de ce type, appelées structures

! Remarque:
Dans certains ouvrages, vous trouverez écrit qu'une structure est un type et que les variables de ce type sont des variables structures ou variables structurées. Cela dépend des auteurs. **L'important est de bien faire la distinction entre le type structuré et les variables de ce type.**

➢ Syntaxe

En C++

```
// déclaration (ou définition) du type de structure[2]
struct ttoto
{
int tutu;
float titi;
} ;

// déclaration des structures du type ttoto(variables)
ttoto toto1, toto2;

// accès au champ titi de toto1 (pour affichage)
cout << toto1.titi;
```

En algorithmique

Type

```
ttoto = enregistrement
        | tutu: entier
        | titi: réel
        finenreg
```

Var

```
toto1, toto2: ttoto
```

Début

```
...
Aff toto1.titi
```

Les champs d'une structure peuvent être de n'importe quel type, y compris d'un "type utilisateur" ou d'un autre type de structure. **La seule contrainte est que les types des champs soient connus du compilateur**. Les types crées par le programmeur doivent avoir été définis (déclarés) préalablement à leur utilisation comme type d'un champ.

---

[1] dans le cas des types de structure, la déclaration correspond à la définition. On peut utiliser ces deux termes indifféremment
[2] Cette syntaxe est valable en C++ mais pas en C où il faudrait utiliser typedef. Pour plus de renseignements, voir les différents ouvrages de C à la bibliothèque.

ε Exemple

Déclaration de 3 clients caractérisés par leur nom, leur prénom et leur adresse, composée d'un numéro de rue, d'un nom de rue, d'un code postal et d'une ville.
Pour simplifier l'utilisation des chaines de caractères, on déclare tout d'abord un type utilisateur chaine

```
typedef                                  // définition d'un type chaine


struct tadresse                  // déclaration d'un type de structure tadresse
{
        num_rue;
        nom_rue;
        cp;
        ville;
};
struct tclient                   // déclaration d'un type de structure tclient
{
          nom;
          prenom;
           adresse;
};

                                                  // déclarations des 3 clients
```

Application

Ecrire un programme en C++ qui saisit les caractéristiques de 2 personnes (nom, prénom, date de naissance) puis qui affiche le nombre d'années d'écart qui sépare la naissance de ces deux personnes (pour simplifier, on considère que deux personnes nées sur deux années différentes ont un an d'écart, même si l'une est née en janvier et l'autre en décembre). Vous pourrez utiliser la fonction abs( ) déclaré dans le fichier d'en-tête math.h.

➢ Les tableaux de structures

Il est bien entendu possible de regrouper sous un même nom, plusieurs structures de même type pour former un tableau de structure.

ε Exemple
Déclaration d'un tableau de 20 clients:

```
tclient client[20];
```

Application
On suppose que le tableau client déclaré ci-dessus a été initialisé.
Ecrire l'instruction qui permet d'afficher le nom, le prénom et l'adresse du quatrième client du tableau, sous la forme suivante:

```
Pierre Dupond
12, rue Jean Monnet
93000 Bobigny
```

Ecrire maintenant la séquence qui permettrait d'afficher ainsi tous les clients, séparés par une ligne.

➢ Passage d'une structure en paramètre d'une fonction et portée des types de structure

Les structures, comme toutes variables, peuvent être passés en paramètre d'une fonction. **L'intégralité de la structure est passée en paramètre**, ce qui évite d'avoir à passer tous les champs uns à uns (cela est possible car l'affectation globale entre deux structures de même type est autorisée).

Le mode de passage des structures est le même que les variables simples (contrairement aux tableaux):
- **par défaut, le passage s'effectue par valeur**, donc la structure paramètre ne sera pas modifiée.
- si la structure est un paramètre résultat ou un paramètre modifié, il faut alors utiliser le **passage par référence (avec &)**.

ε Exemple

Voilà une fonction void qui permute les valeurs des champs de deux clients.
Attention, pour que la permutation se fasse sentir sur les paramètres effectifs, il faut utiliser le passage **par référence** (paramètres en entrée/sortie)

```
void permut (tclient & toto, tclient & titi)
{
  tclient temp;       // déclaration d'une personne temporaire pour la permutation

  temp = toto;        // affectation globale possible entre 2 structures de même type
  toto = titi;
  titi = temp;
}
```

➢ Attention à la portée des types de structures!

Si le type de structure tclient est déclaré (défini) à l'intérieur du programme principal main( ), alors la fonction permut ne va pas fonctionner. En effet, la portée d'un type de structure (ou d'un type utilisateur) suit les mêmes règles que la portée d'une variable:

- s'il est déclaré au sein d'une fonction (y compris main( ) bien entendu), il **n'est accessible (utilisable) qu'au sein de cette fonction,** même pas par les fonctions appelées. Il n'est donc pas possible de passer en paramètre à une autre fonction,.une structure déclarée en local

- s'il est déclaré avant toute fonction, le type structuré est alors accessible par toutes les fonctions contenues dans le fichier source (y compris main( ) bien entendu). Dans ce cas, il est possible de passer une structure en paramètre d'une fonction.

Ainsi, en général, les types structuré sont déclarés avant la fonction main (avant prototypes et les constantes globales.

Dans notre exemple, il faudrait déclarer le type tclient en dehors des fonctions pour pouvoir appeler permut.

Remarque: **une structure peut aussi être la valeur retournée d'une fonction.**

Application

Ecrire un programme qui permute les données concernant deux personnes caractérisées par leur nom, leur prénom et leur date de naissance. La permutation est réalisée à l'aide d'une fonction void.

```
# include …

                                        // déclaration du type utilisateur chaine
                                        // déclaration de la structure date




                                        // déclaration de la structure personne




//prototype (déclaration) de la fonction void permut
main( )
{
// déclaration et initialisation de 2 personnes



// appel de la fonction permut


// affichage pour vérification
cout << pers1.nom << " " << pers1.prenom <<" "<< pers1.ddn.jour << "/ " << pers1.ddn.mois <<"/ "<<
pers1.ddn.annee;
cout<<pers2.nom<<" "<<pers2.prenom<<" "<<pers2.ddn.jour<<"/ "<<pers2.ddn.mois<<"/ "<<
pers2.ddn.annee;
}
// définition de la fonction void permut
```

Exercice

Le but de cet exercice est d'écrire un petit programme qui permette de mémoriser et de traiter différentes informations sur les équipes de rugby du tournoi des 6 nations (France, Angleterre, Pays de Galles, Irlande, Ecosse, Italie). Pour chaque nation, on veut mémoriser le nom, le prénom et le poids de chacun des 15 joueurs de l'équipe (on ne tient pas compte des remplaçants). Chaque joueur a un numéro de poste (de 1 à 15) qui détermine son rôle dans l'équipe (le numéro 1 est pilier gauche, le numéro deux est talonneur, …).

Cette application devra dans un premier temps saisir les informations concernant tous les joueurs de la façon suivante:

```
France
joueur 1: Christian Califano 95
...
joueur 15: Thomas Castaignede 76

Angleterre
joueur 1: ...
```

Une fois les informations saisies, l'application affiche

- le poids, le numéro de poste, le nom et le prénom du joueur le plus lourd, et la nation pour qui il joue, selon le modèle suivant:

```
Avec 130 kg, le numéro 5 William Smith de l'equipe Angleterre est le joueur le plus lourd du tournoi
```

- le poids moyen des joueurs pour chacun des postes

```
les joueurs du poste 1 pèsent en moyenne 98 kg
les joueurs du poste 2 pèsent en moyenne 89 kg
…
les joueurs du poste 15 pèsent en moyenne 73 kg
```

Travail à Faire:

Après avoir choisi des structures de données appropriées pour mémoriser les informations, écrire en C++ le programme demandé de façon modulaire.