# Chapitre 5
# Les structures

✓**Les données d'un programme sont rarement dispersées.**

✓**Elles peuvent en général être pensées sous la forme de groupes plus ou moins importants, ayant une cohérence significative.**

. ✓**Par exemple, dans une gestion de personnel, on utilisera des fiches contenant le nom, le prénom, l'âge, l'adresse, etc., de chaque employé.**

✓**Il serait peu logique de placer chacun des éléments de ces fiches dans des tableaux différents, car cela compliquerait la recherche de l'ensemble des caractéristiques d'un employé donné.**

## Utilisation des structures

✓ **En C++, un regroupement de telles variables différentes peut se faire à l'aide d'une *structure*, définie par exemple comme ceci :**

```
struct fiche {
        char *nom, *prenom;
        int age;
        // etc ...
        };
```

**On a ainsi en fait défini un type, non une variable.**

✓**Des variables de type fiche peuvent être déclarées de la même façon que pour tout autre type :**

```
fiche  employe1, employe2;
```

✓ **Un tel type peut être manipulé comme n'importe quel autre, et l'on peut très bien définir des pointeurs sur ce type, ou des tableaux.**

```
fiche employes[];
```

## Adressage des champs

✓ **Les différents éléments d'une structure sont appelés des *champs*, ou des *données membres*.**

✓ **Lorsqu'on veut accéder à l'un de ces champs, dans une fiche déterminée, il suffit d'indiquer le nom de la variable de type fiche, puis celui du champ (tel qu'il a été défini dans la structure fiche), reliés par l'opérateur . (point) :**

```
fiche employe;
        strcpy(employe.nom ,"Dupont");
        strcpy(employe.prenom ,"Alain");
        employe.age = 34;
```

✓ **Dans le cas de pointeurs, on dispose d'un autre opérateur noté -> :**

```
fiche *pempl;
  strcpy(employe->nom ,"Dupont"); // plus simple que (*pempl).nom
     pempl->age = 25;        // etc.
```

## Définition

✓ **La syntaxe générale d'une définition de structure est la suivante (les crochets indiquent les éléments facultatifs) :**

struct *[nom_struct]*
{ *type1 champ1*;
*type2 champ2*;
.....
*typeN champN*;
} *[var1, var2, ..., varM]*;

✓ **Les termes var1, etc., sont des variables déclarées simultanément.**

✓ **Il est en effet possible de déclarer des variables structure en même temps que le type, comme ceci :**

```
struct fiche
    {char *nom, *prenom;
     int age;
     } employe1, employes[];
```

✓**On a pu voir ci-dessus que le nom de la structure était facultatif.**

✓**En effet, il se peut que l'on n'ait que quelques variables de ce type structure à utiliser dans le programme, auquel cas il n'est pas nécessaire de nommer la structure :**

```
struct {
        char *nom, *prenom;
        int age;
       } employes[100];
```

.

✓**Dans ce cas, il n'est plus possible par la suite de déclarer d'autres variables du même type.**

✓**Il est même possible d'utiliser une déclaration *typedef* pour définir une structure :**

```
typedef  struct {
                 char *nom, *prenom;
                 int age;
                } fiche;
fiche var1, var2;
```

## Arguments de fonctions

✓**Les structures peuvent être passés comme arguments de fonctions de la même façon que tout autre type :**

```
void fonction(struct fiche employe, int  n);
```

**et le mot struct est facultatif.**

✓ **Cependant, le passage d'une structure peut poser des problèmes, puisqu'il s'agit souvent d'éléments volumineux en mémoire. C'est pourquoi on préfère en général passer les structures sous la forme de pointeurs :**

```
void fonction(fiche *pempl, int n);
```

**ou de références :**

```
void fonction(fiche &employe, int n);
```

✓ **Évidemment, de tels passages sont obligatoires si la fonction doit avoir un effet de bord modifiant la structure.**

✓ **Dans le cas contraire, on aura intérêt à préciser le pointeur ou la référence comme désignant une structure constante, à l'aide du mot const.**

✓ **Lorsqu'une structure est passée entièrement, c'est-à-dire par valeur, certains compilateurs C++ émettent un avertissement :**

Warning : Structure passed by value.

✓ **C'est le moment ou jamais de réfléchir si ce passage par valeur est réellement nécessaire. Dans l'affirmative, il suffit d'ignorer le message.**

## Exercice :

**Soit une structure point{int num; float x; float y;}**

**a- Déclarer une telle structure, saisir un point, afficher ses champs.**

**b- Même exercice mais en créant une fonction de prototype**

void saisie(point &fx)

**et une fonction de prototype**

void affiche(const point &fx).

**c- Reprendre la question b en utilisant les pointeurs.**

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