



# Les structures

# 1. Créez vos propres types de variables

En C, lorsque l'on cherche à faire des programmes plus complexes, on peut créer nos propres types de variables

1. Les structures.
2. Et les énumérations.


## ► Définissez une structure

Une structure est un assemblage de variables qui peuvent avoir différents types. Contrairement aux tableaux qui vous obligent à utiliser le même type dans tout le tableau, vous pouvez créer une structure comportant des variables de types **long**, **char**, **int** et **double** à la fois. Les structures sont généralement définies dans les fichiers **.h** , au même titre que les prototypes et les **define**.

Pour définir une structure, il suffit de :

commencer par taper **struct**;

puis écrire le nom de la structure;



**NB:** nommez vos structures en suivant les mêmes règles que pour les noms de variables, sauf que vous mettez la première lettre en majuscule pour pouvoir faire la différence.

Une structure est généralement composée d'au moins deux "sous-variables", sinon elle n'a pas trop d'intérêt.

### Exemple:

```
struct NomDeVotreStructure
{
    int variable1;
    int variable2;
    int autreVariable;
    double nombreDecimal;
};
```

### ➡ Placez des tableaux dans une structure

Les structures peuvent contenir des tableaux. une structure **Personne** qui stocke diverses informations sur une personne :

```
struct Personne
{
    char nom[100];
    char prenom[100];
    char adresse[1000];

    int age;
    int etudiant; // Booléen : 1 = etudiant, 0 = non etudiant
};
```

## ► Utilisez une structure dans une fonction

Maintenant que notre structure est définie dans le **.h**, on va pouvoir l'utiliser dans une fonction de notre fichier **.c** .

Voici comment créer une variable de type **Coordonnees** :

```
#include "main.h" // Inclusion du .h qui contient les prototypes et structures

int main(int argc, char *argv[])
{
    struct Coordonnees point; // Création d'une variable "point" de type Coordonnees

    return 0;
}
```

Nous avons ainsi créé une variable point de type **Coordonnees** . Cette variable est automatiquement composée de deux sous-variables : **x** et **y** (son abscisse et son ordonnée).

Faut-il obligatoirement écrire le mot-clé **struct** lors de la définition de la variable ?

Oui, cela permet à l'ordinateur de différencier un type de base (comme **int** ) d'un type personnalisé, comme **Coordonnees**.

les programmeurs trouvent souvent un peu lourd de mettre le mot **struct** à chaque définition de variable personnalisée. Pour régler ce problème, ils ont inventé une instruction spéciale : le **typedef** .

## ► Créez un alias de structure avec l'instruction typedef

Retournons dans le fichier **.h** qui contient la définition de notre structure de type **Coordonnees**. Nous allons ajouter une instruction appelée **typedef** pour créer un alias de structure et dire qu'écrire telle chose équivaut à écrire telle autre chose. Ajoutons une ligne commençant par **typedef** juste avant la définition de la structure :

```
typedef struct Coordonnees Coordonnees;  
struct Coordonnees  
{  
    int x;  
    int y;  
};
```

1. **typedef**: indique que nous allons créer un alias de structure;
2. **struct Coordonnees** : c'est le nom de la structure dont vous allez créer un alias (c'est-à-dire un "équivalent") ;
3. **Coordonnees** : c'est le nom de l'équivalent.

Écrire le mot **Coordonnees** est désormais équivalent à écrire **struct Coordonnees**.

En faisant cela, vous n'aurez plus besoin de mettre le mot **struct** à chaque définition de variable de type **Coordonnees** . On peut donc retourner dans notre **main** et écrire :

```
int main(int argc, char *argv[])
{
    Coordonnees point; // L'ordinateur comprend qu'il s'agit de "struct Coordonnees" grâce au typedef
    return 0;
}
```

### ► Modifiez les composantes de la structure

Maintenant que notre variable **point** est créée, nous voulons modifier ses coordonnées. Comment accéder au **x** et au **y** de point?

```
int main(int argc, char *argv[])
{
    Coordonnees point;

    point.x = 10;
    point.y = 20;

    return 0;
}
```



Pour accéder à chaque composante de la structure, vous devez donc écrire :

### **variable.nomDeLaComposante**

Si on prend la structure `Personne` que nous avons vue tout à l'heure, et qu'on demande le nom et le prénom, on devra faire comme ça :

```
int main(int argc, char *argv[])
{
    Personne utilisateur;

    printf("Quel est votre nom ? ");
    scanf("%s", utilisateur.nom);
    printf("Votre prénom ? ");
    scanf("%s", utilisateur.prenom);

    printf("Vous vous appelez %s %s", utilisateur.prenom, utilisateur.nom);

    return 0;
}
```

Mais l'intérêt ici est que vous pouvez créer une autre variable de type `Personne` qui aura aussi son propre nom, son propre prénom, etc.

### **Personne joueur1, joueur2;**

et stocker ainsi les informations sur chaque joueur. Chaque joueur a son propre nom, son propre prénom, etc.

On peut même faire encore mieux : on peut créer un tableau de Personne :

**Personne joueurs[2];**

Et ensuite, vous accédez par exemple au nom du joueur n° 0 en tapant :

**joueurs[0].nom**

L'avantage d'utiliser un tableau ici, c'est que vous pouvez faire une boucle pour demander les infos du joueur 1 et du joueur 2, sans avoir à répéter deux fois le même code. Il suffit de parcourir le tableau joueur et de demander à chaque fois nom, prénom, adresse...

### ► **Initialisez une structure**

Pour les structures comme pour les variables, tableaux et pointeurs, il est vivement conseillé de les initialiser dès leur création pour éviter qu'elles ne contiennent n'importe quoi.

Pour une structure, l'initialisation ressemble un peu à celle d'un tableau.

```
Coordonnees point = {0, 0};
```

Cela définira, dans l'ordre :

1. point.x = 0.
2. point.y = 0.



## ► Utilisez un pointeur sur une structure

Un pointeur de structure se crée de la même manière qu'un pointeur de `int` , de `double` ou de n'importe quel autre type de base :

```
Coordonnees* point = NULL;
```

On a ainsi un pointeur de **Coordonnees** appelé **point**.

Je tiens à vous répéter que l'on aurait aussi pu mettre l'étoile devant le nom du pointeur, cela revient exactement au même :

```
Coordonnees *point1 = NULL, *point2 = NULL;
```

## ► Envoyez la structure à une fonction

```
int main(int argc, char *argv[])
{
    Coordonnees monPoint;

    initialiserCoordonnees(&monPoint);

    return 0;
}

void initialiserCoordonnees(Coordonnees* point)
{
    // Initialisation de chacun des membres de la structure ici
}
```

On envoie son adresse à la fonction **initialiserCoordonnees** qui récupère cette variable sous la forme d'un pointeur appelé point. Maintenant que nous sommes dans **initialiserCoordonnees**, nous allons initialiser chacune des valeurs une à une.

```
void initialiserCoordonnees(Coordonnees* point)
{
    (*point).x = 0;
    (*point).y = 0;
}
```

**Remarque:** Il ne faut pas oublier de mettre une étoile devant le nom du pointeur pour accéder à la variable. Si vous ne le faites pas, vous risquez de modifier l'adresse, et ce n'est pas ce que nous voulons faire.

### ➡ Utilisez ce raccourci

Vous allez voir qu'on manipulera très souvent des pointeurs de structures. Comme les pointeurs de structures sont très utilisés, on sera souvent amené à écrire ceci :

```
(*point).x = 0;
```

les programmeurs trouvent ça trop long. Les parenthèses autour de \*point , quelle plaie ! Alors, ils ont inventé le raccourci suivant:

```
point->x = 0;
```

Remarque:

On ne peut utiliser la flèche que sur un pointeur ! Si vous travaillez directement sur la variable, vous devez utiliser le point.

```
int main(int argc, char *argv[])
{
    Coordonnees monPoint;
    Coordonnees *pointeur = &monPoint;

    monPoint.x = 10; // On travaille sur une variable, on utilise le "point"
    pointeur->x = 10; // On travaille sur un pointeur, on utilise la flèche

    return 0;
}
```

On modifie la valeur du x à 10 de deux manières différentes, ici : la première fois en travaillant directement sur la variable, la seconde fois en passant par le pointeur.

Reprenons maintenant notre fonction **initialiserCoordonnees**; nous pouvons alors l'écrire comme ceci :

```
void initialiserCoordonnees(Coordonnees* point)
{
    point->x = 0;
    point->y = 0;
}
```