Apprenez à programmer en C !

40 heures Moyenne

Mis à jour le 29/07/2019

Créez vos propres types de variables

Le langage C nous permet de faire quelque chose de très puissant : créer nos propres types de variables. Des « types de variables personnalisés », nous allons en voir deux sortes : les structures et les énumérations.

Créer de nouveaux types de variables devient indispensable quand on cherche à faire des programmes plus complexes.

Ce n'est (heureusement) pas bien compliqué à comprendre et à manipuler. Restez attentifs tout de même parce que nous réutiliserons les structures tout le temps à partir du prochain chapitre.
Il faut savoir que les bibliothèques définissent généralement leurs propres types. Vous ne tarderez donc pas à manipuler un type de variable `Fichier` ou encore, un peu plus tard, d'autres de types `Fenetre`, `Audio`, `Clavier`, etc.

Définir 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 donc que les prototypes et les `define`. Voici un exemple de structure :

c

```c
struct NomDeVotreStructure
{
    int variable1;
    int variable2;
```

```
    int autreVariable;
    double nombreDecimal;
};
```

Une définition de structure commence par le mot-clé `struct`, suivi du nom de votre structure (par exemple `Fichier`, ou encore `Ecran`).

> J'ai personnellement l'habitude de nommer mes structures en suivant les mêmes règles que pour les noms de variables, excepté que je mets la première lettre en majuscule pour pouvoir faire la différence. Ainsi, quand je vois le mot `ageDuCapitaine` dans mon code, je sais que c'est une variable car cela commence par une lettre minuscule. Quand je vois `MorceauAudio` je sais qu'il s'agit d'une structure (un type personnalisé) car cela commence par une majuscule.

Après le nom de votre structure, vous ouvrez les accolades et les fermez plus loin, comme pour une fonction.

> Attention, ici c'est particulier : vous DEVEZ mettre un point-virgule après l'accolade fermante. C'est obligatoire. Si vous ne le faites pas, la compilation plantera.

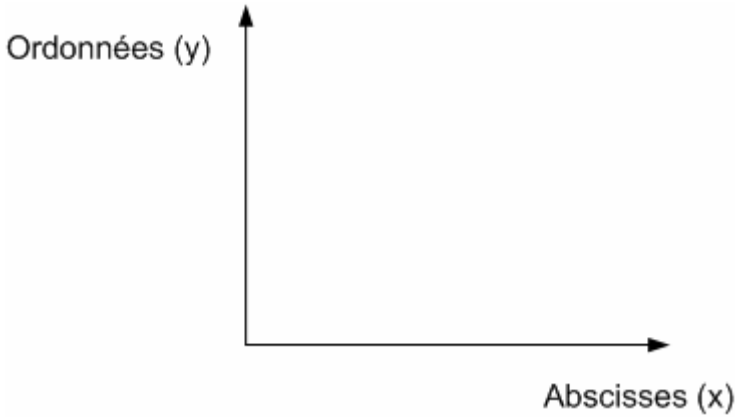
Et maintenant, que mettre entre les accolades ?
C'est simple, vous y placez les variables dont est composée votre structure. Une structure est généralement composée d'au moins deux « sous-variables », sinon elle n'a pas trop d'intérêt.

Comme vous le voyez, la création d'un type de variable personnalisé n'est pas bien complexe. Toutes les structures que vous verrez sont en fait des « assemblages » de variables de type de base, comme `long`, `int`, `double`, etc. Il n'y a pas de miracle, un type `Fichier` n'est donc composé que de nombres de base !

Exemple de structure

Imaginons par exemple que vous vouliez créer une variable qui stocke les coordonnées d'un point à l'écran. Vous aurez très certainement besoin d'une structure comme cela lorsque vous ferez des jeux 2D dans la partie suivante, c'est donc l'occasion de s'avancer un peu.

Pour ceux chez qui le mot « géométrie » provoque des apparitions de boutons inexplicables sur tout le visage, la fig. suivante va faire office de petit rappel fondamental sur la 2D.



Lorsqu'on travaille en 2D (2 dimensions), on a deux axes : l'axe des abscisses (de gauche à droite) et l'axe des ordonnées (de bas en haut). On a l'habitude d'exprimer les abscisses par une variable appelée `x`, et les ordonnées par `y`.

Êtes-vous capables d'écrire une structure `Coordonnees` qui permette de stocker à la fois la valeur de l'abscisse (`x`) et celle de l'ordonnée (`y`) d'un point ?
Allons, allons, ce n'est pas bien difficile :

c

```c
struct Coordonnees
{
    int x; // Abscisses
    int y; // Ordonnées
};
```

Notre structure s'appelle `Coordonnees` et est composée de deux variables, `x` et `y`, c'est-à-dire de l'abscisse et de l'ordonnée.

Si on le voulait, on pourrait facilement faire une structure `Coordonnees` pour de la 3D : il suffirait d'ajouter une troisième variable (par exemple `z`) qui indiquerait la hauteur. Avec ça, nous aurions une structure faite pour gérer des points en 3D dans l'espace !

Tableaux dans une structure

Les structures peuvent contenir des tableaux. Ça tombe bien, on va pouvoir ainsi placer des tableaux de `char` (chaînes de caractères) sans problème !
Allez, imaginons une structure `Personne` qui stockerait diverses informations sur une personne :

c_cpp

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

    int age;
    int genre; // Booléen : 1 = garçon, 0 = fille
};
```

Cette structure est composée de cinq sous-variables. Les trois premières sont des chaînes qui stockeront le nom, le prénom et l'adresse de la personne.
Les deux dernières stockent l'âge et le sexe de la personne. Le sexe est un booléen, 1 = vrai = garçon, 0 = faux = fille.

Cette structure pourrait servir à créer un programme de carnet d'adresses. Bien entendu, vous pouvez rajouter des variables dans la structure pour la compléter si vous le voulez. Il n'y a pas de limite au nombre de variables dans une structure.

Utilisation d'une structure

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` (la structure qu'on a définie plus haut) :

c

```c
#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`.
Toutefois, 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`.

Le `typedef`

Retournons dans le fichier `.h` qui contient la définition de notre structure de type `Coordonnees`.
Nous allons ajouter une instruction appelée `typedef` qui sert à créer un alias de structure, c'est-à-dire à dire qu'écrire telle chose équivaut à écrire telle autre chose.

Nous allons ajouter une ligne commençant par `typedef` juste avant la définition de la structure :

c

```c
typedef struct Coordonnees Coordonnees;
```

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

Cette ligne doit être découpée en trois morceaux (non, je n'ai pas bégayé le mot `Coordonnees` !) :

- `typedef` : indique que nous allons créer un alias de structure ;
- `struct Coordonnees` : c'est le nom de la structure dont vous allez créer un alias (c'est-à-dire un « équivalent ») ;
- `Coordonnees` : c'est le nom de l'équivalent.

En clair, cette ligne dit « 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 tout simplement :

c

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

Je vous recommande de faire un `typedef` comme je l'ai fait ici pour `Coordonnees`. La plupart des programmeurs font comme cela. Ça leur évite d'avoir à écrire le mot `struct` partout. Un bon programmeur est un programmeur fainéant ! Il en écrit le moins possible.

Modifier 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` ? Comme ceci :

c

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

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

    return 0;
}
```

On a ainsi modifié la valeur de `point`, en lui donnant une abscisse de 10 et une ordonnée de 20. Notre point se situe désormais à la position (10 ; 20) (c'est la notation mathématique d'une coordonnée.

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

c

```c
variable.nomDeLaComposante
```

Le point fait la séparation entre la variable et la composante.

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 :

c

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

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

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

    return 0;
}
```

```
Quel est votre nom ? Dupont
Votre prenom ? Jean
Vous vous appelez Jean Dupont
```

On envoie la variable `utilisateur.nom` à `scanf` qui écrira directement dans notre variable `utilisateur`.
On fait de même pour `prenom`, et on pourrait aussi le faire pour l'adresse, l'âge et le sexe, mais je n'ai guère envie de me répéter (je dois être programmeur, c'est pour ça ).

Vous auriez pu faire la même chose sans connaître les structures, en créant juste une variable `nom` et une autre `prenom`.
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. On peut donc faire :

c

```c
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` !
C'est facile à faire :

c

```c
Personne joueurs[2];
```

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

```c
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...

Exercice : créez ce tableau de type `Personne` et demandez les infos de chacun grâce à une boucle (qui se répète tant qu'il y a des joueurs). Faites un petit tableau de 2 joueurs pour commencer, mais si ça vous amuse, vous pourrez agrandir la taille du tableau plus tard.
Affichez à la fin du programme les infos que vous avez recueillies sur chacun des joueurs.

Initialiser 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 ». En effet, je vous le rappelle, une variable qui est créée prend la valeur de ce qui se trouve en mémoire là où elle a été placée. Parfois cette valeur est 0, parfois c'est un résidu d'un autre programme qui est passé par là avant vous et la variable a alors une valeur qui n'a aucun sens, comme -84570.

Pour rappel, voici comment on initialise :

- **une variable** : on met sa valeur à 0 (cas le plus simple) ;
- **un pointeur** : on met sa valeur à `NULL`. `NULL` est en fait un `#define` situé dans `stdlib.h` qui vaut généralement 0, mais on continue à utiliser `NULL`, par convention, sur les pointeurs pour bien voir qu'il s'agit de pointeurs et non de variables ordinaires ;
- **un tableau** : on met chacune de ses valeurs à 0.

Pour les structures, l'initialisation va un peu ressembler à celle d'un tableau. En effet, on peut faire à la déclaration de la variable :

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

Cela définira, dans l'ordre, `point.x = 0` et `point.y = 0`.

Revenons à la structure `Personne` (qui contient des chaînes). Vous avez aussi le droit d'initialiser une chaîne en écrivant juste `""` (rien entre les guillemets). Je ne vous ai pas parlé de cette possibilité dans le chapitre sur les chaînes, mais il n'est pas trop tard pour l'apprendre.
On peut donc initialiser dans l'ordre `nom`, `prenom`, `adresse`, `age` et `garcon` comme ceci :

```c
Personne utilisateur = {"", "", "", 0, 0};
```

Toutefois, j'utilise assez peu cette technique, personnellement. Je préfère envoyer par exemple ma variable `point` à une fonction `initialiserCoordonnees` qui se charge de faire les initialisations pour moi sur ma variable.
Pour faire cela il faut envoyer un pointeur de ma variable. En effet si j'envoie juste ma variable, une copie en sera réalisée dans la fonction (comme pour une variable de base) et la fonction modifiera les valeurs de la copie et non celle de ma vraie variable. Revoyez le fil rouge du chapitre sur les pointeurs si vous avez oublié comment cela fonctionne.

Il va donc falloir apprendre à utiliser des pointeurs sur des structures. Les choses vont commencer à se corser un petit peu !

Pointeur de structure

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

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

On a ainsi un pointeur de `Coordonnees` appelé `point`.
Comme un rappel ne fera de mal à personne, je tiens à vous répéter que l'on aurait aussi pu mettre l'étoile devant le nom du pointeur, cela revient exactement au même :

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

Je fais d'ailleurs assez souvent comme cela, car pour définir plusieurs pointeurs sur la même ligne, nous sommes obligés de placer l'étoile devant chaque nom de pointeur :

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

Envoi de la structure à une fonction

Ce qui nous intéresse ici, c'est de savoir comment envoyer un pointeur de structure à une fonction pour que celle-ci puisse modifier le contenu de la variable.

On va faire ceci pour cet exemple : on va simplement créer une variable de type `Coordonnees` dans le `main` et envoyer son adresse à `initialiserCoordonnees`. Cette fonction aura pour rôle de mettre tous les éléments de la structure à 0.

Notre fonction `initialiserCoordonnees` va prendre un paramètre : un pointeur sur une structure de type `Coordonnees` (un `Coordonnees*`, donc).

c
```c
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
}
```

Ma variable `monPoint` est donc créée dans le `main`.
On envoie son adresse à la fonction `initialiserCoordonnees` qui récupère cette variable sous la forme d'un pointeur appelé `point` (on aurait d'ailleurs pu l'appeler n'importe comment dans la fonction, cela n'aurait pas eu d'incidence).

Bien : maintenant que nous sommes dans `initialiserCoordonnees`, nous allons initialiser chacune des valeurs une à une.
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.

Oui mais voilà, problème... On ne peut pas vraiment faire :

c

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

Ce serait trop facile... Pourquoi on ne peut pas faire ça ? Parce que le point de séparation s'applique sur le mot `point` et non sur `*point` en entier. Or, nous ce qu'on veut, c'est accéder à `*point` pour en modifier la valeur.
Pour régler le problème, il faut placer des parenthèses autour de `*point`. Comme cela, le point de séparation s'appliquera à `*point` et non juste à `point` :

c

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

Ce code fonctionne, vous pouvez tester. La variable de type `Coordonnees` a été transmise à la fonction qui a initialisé `x` et `y` à 0.

> En langage C, on initialise généralement nos structures avec la méthode simple qu'on a vue plus haut.
> En C++ en revanche, les initialisations sont plus souvent faites dans des « fonctions ».

> Le C++ n'est en fait rien d'autre qu'une sorte de « super-amélioration » des structures. Bien entendu, beaucoup de choses découlent de cela et il faudrait un livre entier pour en parler (chaque chose en son temps).

Un raccourci pratique et très utilisé

Vous allez voir qu'on manipulera très souvent des pointeurs de structures. Pour être franc, je dois même vous avouer qu'en C, on utilise plus souvent des pointeurs de structures que des structures tout court. Quand je vous disais que les pointeurs vous poursuivraient jusque dans votre tombe, je ne le disais presque pas en rigolant !

Comme les pointeurs de structures sont très utilisés, on sera souvent amené à écrire ceci :

c

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

Oui mais voilà, encore une fois les programmeurs trouvent ça trop long. Les parenthèses autour de `*point`, quelle plaie ! Alors, comme les programmeurs sont des gens fainéants (mais ça, je l'ai déjà dit, je crois), ils ont inventé le raccourci suivant :

c

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

Ce raccourci consiste à former une flèche avec un tiret suivi d'un chevron `>`.

Écrire `point->x` est donc STRICTEMENT équivalent à écrire `(*point).x`.

> N'oubliez pas qu'on ne peut utiliser la flèche que sur un pointeur !
> Si vous travaillez directement sur la variable, vous devez utiliser le point comme on l'a vu au début.

Reprenons notre fonction `initialiserCoordonnees`. Nous pouvons donc l'écrire comme ceci :

c

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

Retenez bien ce raccourci de la flèche, nous allons le réutiliser un certain nombre de fois. Et surtout, ne confondez pas la flèche avec le « point ». La flèche est réservée aux pointeurs, le « point » est réservé aux variables. Utilisez ce petit exemple pour vous en souvenir :

c

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

```c
    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.

Les énumérations

Les énumérations constituent une façon un peu différente de créer ses propres types de variables.

Une énumération ne contient pas de « sous-variables » comme c'était le cas pour les structures. C'est une liste de « valeurs possibles » pour une variable. Une énumération ne prend donc qu'une case en mémoire et cette case peut prendre une des valeurs que vous définissez (et une seule à la fois).

Voici un exemple d'énumération :

c

```c
typedef enum Volume Volume;
enum Volume
{
    FAIBLE, MOYEN, FORT
};
```

Vous noterez qu'on utilise un `typedef` là aussi, comme on l'a fait jusqu'ici.

Pour créer une énumération, on utilise le mot-clé `enum`. Notre énumération s'appelle ici `Volume`. C'est un type de variable personnalisé qui peut prendre une des trois valeurs qu'on a indiquées : soit `FAIBLE`, soit `MOYEN`, soit `FORT`.

On va pouvoir créer une variable de type `Volume`, par exemple `musique`, qui stockera le volume actuel de la musique.
On peut par exemple initialiser la musique au volume `MOYEN` :

c

```c
Volume musique = MOYEN;
```

Voilà qui est fait. Plus tard dans le programme, on pourra modifier la valeur du volume et la mettre soit à `FAIBLE`, soit à `FORT`.

Association de nombres aux valeurs

Vous avez remarqué que j'ai écrit les valeurs possibles de l'énumération en majuscules. Cela devrait vous rappeler les constantes et les `define`, non ?

En effet, c'est assez similaire mais ce n'est pourtant pas exactement la même chose.
Le compilateur associe automatiquement un nombre à chacune des valeurs possibles de l'énumération.

Dans le cas de notre énumération `Volume`, `FAIBLE` vaut 0, `MOYEN` vaut 1 et `FORT` vaut 2. L'association est automatique et commence à 0.

Contrairement au `#define`, c'est le compilateur qui associe `MOYEN` à 1 par exemple, et non le préprocesseur. Au bout du compte, ça revient un peu au même.
En fait, quand on a initialisé la variable `musique` à `MOYEN`, on a donc mis la case en mémoire à la valeur 1.

> En pratique, est-ce utile de savoir que `MOYEN` vaut 1, `FORT` vaut 2, etc. ?

Non. En général ça nous est égal. C'est le compilateur qui associe automatiquement un nombre à chaque valeur. Grâce à ça, vous n'avez plus qu'à écrire :

c

```c
if (musique == MOYEN)
{
    // Jouer la musique au volume moyen
}
```

Peu importe la valeur de `MOYEN`, vous laissez le compilateur se charger de gérer les nombres.

L'intérêt de tout ça ? C'est que de ce fait votre code est très lisible. En effet, tout le monde peut facilement lire le `if` précédent (on comprend bien que la condition signifie « Si la musique est au volume moyen »).

Associer une valeur précise

Pour le moment, c'est le compilateur qui décide d'associer le nombre 0 à la première valeur, puis 1, 2, 3 dans l'ordre.
Il est possible de demander d'associer une valeur précise à chaque élément de l'énumération.

Quel intérêt est-ce que ça peut bien avoir ? Eh bien supposons que sur votre ordinateur, le volume soit géré entre 0 et 100 (0 = pas de son, 100 = 100 % du son). Il est alors pratique d'associer une valeur précise à chaque élément :

c

```c
typedef enum Volume Volume;
enum Volume
{
    FAIBLE = 10, MOYEN = 50, FORT = 100
};
```

Ici, le volume `FAIBLE` correspondra à 10 % de volume, le volume `MOYEN` à 50 %, etc.
On pourrait facilement ajouter de nouvelles valeurs possibles comme `MUET`. On associerait dans ce cas `MUET` à la valeur... 0 ! Vous avez compris.

En résumé

- Une structure est un type de variable personnalisé que vous pouvez créer et utiliser dans vos programmes. C'est à vous de la définir, contrairement aux types de base tels que `int` et `double` que l'on retrouve dans tous les programmes.
- Une structure est composée de « sous-variables » qui sont en général des variables de type de base comme `int` et `double`, mais aussi des tableaux.
- On accède à un des composants de la structure en séparant le nom de la variable et la composante d'un point : `joueur.prenom`.
- Si on manipule un pointeur de structure et qu'on veut accéder à une des composantes, on utilise une flèche à la place du point : `pointeurJoueur->prenom`.
- Une énumération est un type de variable personnalisé qui peut seulement prendre une des valeurs que vous prédéfinissez : `FAIBLE`, `MOYEN` ou `FORT` par exemple.

J'AI TERMINÉ CE CHAPITRE ET JE PASSE AU SUIVANT

‹ LE PRÉPROCESSEUR

LIRE ET ÉCRIRE DANS DES FICHIERS ›

Le professeur

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Découvrez aussi ce cours en...

Livre

PDF