C - Les structures (`struct`)

Une structure (ou *struct*) permet de **regrouper plusieurs variables** sous un même nom. Une `struct` représenter des données plus complexes qu'un simple entier ou qu'un `float`. Avec une structure, tu peux modéliser un élève, un produit, un point, une date… en associant plusieurs informations dans un même ensemble logique.

5TTR 6TTR Découverte

Ce chapitre t'apprendra à créer, utiliser et comprendre les *struct* en C.

Objectifs

À la fin de cette leçon, tu sauras :

- Définir une structure avec le mot-clé `struct`.
- Déclarer et utiliser des variables de type structure.
- Accéder aux champs d'une structure avec l'opérateur `.`
- Utiliser `typedef` pour simplifier l'écriture.
- Comprendre comment les structures sont stockées en mémoire.

Qu'est-ce qu'une structure ?

Une structure regroupe plusieurs variables (appelées **champs**) qui peuvent être de types différents.

Exemple : représenter un élève avec

- son nom,
- son âge,
- sa moyenne.

Sans `struct`, tu aurais trois variables séparées. Avec `struct`, tu obtiens un seul "bloc" cohérent.

Définir une structure

Exemple simple

```c
typedef struct {
    char nom[30];
    int age;
    float moyenne;
} Etudiant;
```

Ici :

- `nom` est un tableau de 30 caractères,
- `age` est un entier,
- `moyenne` est un `float`.

Tu viens de créer **un nouveau type personnalisé** : `struct Etudiant`.

`typedef` permet d'utiliser ta structure plus simplement par la suite (cela te permet de faire `Etudiant manu = ...` au lieu de `struct Etudiant manu=...` quand tu déclares tes variables).

Simpl

Déclarer et utiliser une structure

```c
Etudiant e1;

strcpy(e1.nom, "Alice");
e1.age = 17;
e1.moyenne = 14.5;

printf("%s a %d ans et %.2f de moyenne.\n", e1.nom, e1.age, e1.moyenne);
```

Accès aux champs

On utilise l'opérateur `.` :

```
e1.nom
e1.age
e1.moyenne
```

Créer et initialiser en une ligne

```c
struct Etudiant e2 = {"Lucas", 18, 12.8};
```

Structures contenant plusieurs tableaux

Oui, c'est possible. Exemple :

```c
typedef struct {
    char nom[30];
    char prenom[40];
    int age;
} Personne;
```

Chaque tableau peut contenir une chaîne de caractères distincte.

Lire des données dans une structure

```c
Personne p;

printf("Nom : ");
scanf("%s", p.nom);

printf("Prénom : ");
scanf("%s", p.prenom);

printf("Âge : ");
scanf("%d", &p.age);
```

💡 Les tableaux (`char[]`) sont automatiquement passés par référence → pas de `&`.

Structures et mémoire

Une structure regroupe ses champs **de manière contiguë en mémoire**.

Exemple :

```
struct Etudiant
{
    char nom[30];   ← 30 octets
    int age;        ← 4 octets
    float moyenne;  ← 4 octets
}
```

La taille totale est au moins 38 octets (souvent un peu plus à cause de l'alignement mémoire).

Structures et fonctions

Une structure peut être passée à une fonction **par valeur** ou **par référence**.

Par valeur (copie)

```c
void afficher(Etudiant e) {
    printf("%s (%d ans)\n", e.nom, e.age);
}
```

Par référence (pointeur)

```c
void modifier(Etudiant *e) {
    e->age += 1;
}
```

💡 On utilise `->` pour accéder aux champs via un pointeur.

À retenir

- Une structure regroupe plusieurs données liées entre elles.
- On accède aux champs avec `.` ou `->` selon les cas.
- `typedef` permet de créer un type plus simple à utiliser.
- Les tableaux et chaînes dans une structure sont tout à fait possibles.
- Une structure peut être passée par valeur (copie) ou par référence (adresse).

Erreurs courantes à corriger

- Oublier le `;` à la fin de la définition de la structure.
- Confondre tableau de `char` et chaîne "classique".
- Mélanger `.` et `->`.
- Tenter de faire `p.nom = "Alice";` au lieu de `strcpy`.