

# Introduction à la programmation en C++

## Les structures

---

Nicolas Audebert

Vendredi 6 octobre 2017



## Rendus de TP et des exercices

Les rendus se font sur **Educnet**.

1. Le code rendu **doit compiler**.
2. Le code rendu doit **être propre** (indentation, noms de variables clairs).
3. Le code rendu doit **être commenté** (réponses aux questions, fonctionnement du code).
4. Rassembler le code dans une seule archive (**.zip**, **.rar**, **.tar.gz**, etc.).

Rappels

Pourquoi les structures ?

Définition

Manipulation

TP du jour

Les tableaux statiques sont caractérisés par le **type** de leurs éléments et leur **taille**.

## Exemple

```
int mon_tableau[10];  
// Initialisation manuelle à l'aide d'une boucle  
for(int i=0; i < 10; i++){  
    mon_tableau[i] = 5;  
}  
  
// Déclaration et initialisation directe  
double tableau_reel[5] = {2, 3.2, 9.76, 6, 1000};  
  
// Déclaration puis initialisation directe  
bool tableau_bool[3];  
tableau_bool = {true, true, false};
```

Si  $n$  est la taille du tableau, alors les indices vont de  $0$  à  $n-1$ .

## Attention

Tenter d'accéder à un élément hors de ces bornes résultera systématiquement en une erreur lors de l'exécution du programme.

## Exemple

```
const int n = 100; // Taille du tableau (constante)
char tab[n]; // Déclaration du tableau
tab[0] = 'a'; // OK
tab[10] = 'd'; // OK
tab[n-1] = 'k'; // OK
tab[n] = 'f'; // ERREUR
tab[-1] = 'z'; // ERREUR
```

Une fonction peut manipuler un tableau dans sa signature :

```
void affiche(int t[5]){  
    for(int i=0; i<5; i++){  
        cout << t[i] << " ";  
    }  
    cout << endl;  
}
```

```
void affiche(int t[], int  
    ↪ taille){  
    for(int i=0; i<taille; i++){  
        cout << t[i] << " ";  
    }  
    cout << endl;  
}
```

## Passage par référence

- ▶ Un tableau est **toujours** implicitement passé par référence (il ne faut donc pas rajouter de  $\&$ ).
- ▶ Une fonction **ne peut pas** retourner de tableau.

On ne peut pas copier directement des tableaux entre eux.

```
int t1[4] = {1,2,3,4}, t2[4];  
t2 = t1 ;  
// ERREUR : pas d'affectation avec le = pour les tableaux
```

Seule solution : itérer sur les éléments.

```
int t1[4] = {1,2,3,4}, t2[4];  
for(int i = 0; i < 4; i++){  
    t2[i] = t1[i];  
}
```

De même, pour tester l'égalité entre deux tableaux.

```
if(a = 3){...} // ERREUR: le symbole d'egalite est '=='  
if(a == 3){...}
```

```
for(int i; i < 10; i++){...} // ERREUR: il faut initialiser i  
for(int i = 0; i < 10; i++){...} // OK
```

```
if(a && for(int i = 0; i < 100; i++){tab[i]}){...} // ERREUR
```

```
bool test = true;  
for(int i = 0; i < 100; i++){  
    if(! tab[i])  
        test=false;  
}
```



```
if(a && test){...}
```

```
void f(double tab[8]){...} // argument : un tableau
```

```
void g(){
```

```
...
```

```
double vec[8]; // tableau de 8 cases
```

```
f(vec[8]); // ERREUR : vec[nombre] est le contenu d'une
```

```
→ case (un nombre)
```

```
    // et en plus, la case 8 n'existe pas !
```

```
f(vec); // OK on appelle la fonction sur la variable vec
```

```
→ (un tableau)
```

```
}
```

Rappels

Pourquoi les structures ?

Définition

Manipulation

TP du jour

## Pour l'instant...

- ▶ factoriser le code : **les fonctions**
- ▶ regrouper les variables de même type : **les tableaux**

## Et maintenant...

Regrouper des variables qui ne sont pas forcément du même type mais qui forment un ensemble cohérent :

- ▶ Contacts : nom, date de naissance, adresse...
- ▶ Dessin : forme, couleur, épaisseur du trait...
- ▶ ...

On utilise des **structures**.

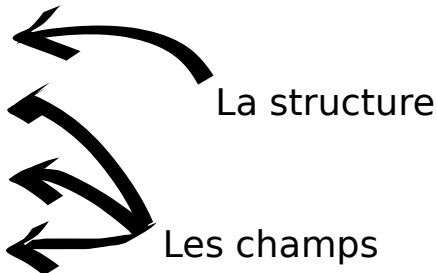
## Variable

sous-variable 1

sous-variable 2

sous-variable 3

...



Les structures définissent de nouveaux types.

Les éléments de la structure sont appelés des **champs**.

Rappels

Pourquoi les structures ?

Définition

Manipulation

TP du jour

## Cas général

```
struct nom_structure{  
    type1 var1; // les champs  
    type2 var2;  
  
}; // À ne pas oublier !
```

La structure définit un nouveau type qui s'utilise comme les autres. On accède aux champs avec un `.`

## Cas général

```
variable_struct.var1 = 1000;  
cout << variable_struct.var2 << endl;
```

## Tableaux dans les structures

Très, très fortement **DÉCONSEILLÉ** : problèmes liés à l'égalité entre tableaux, notamment dans les retours de fonctions.

## Tableaux de structures

Aucun problème, ils se comportent comme des variables classiques.

# Exemple

```
struct Client{
    string nom, prenom;
    string adresse;
    int naissance[3]; // JJ,
    ↪ MM, AAAA
    double taille;
    double poids;
    bool lunettes;
};
```

```
Client cli1;
cli1.nom = "Presley"
cli1.prenom = "Elvis"
cli1.naissance[0] = 8;
cli1.naissance[1] = 1;
cli1.naissance[2] = 1935;
```

```
struct Point{
    double x,y;
};
```

```
struct Cercle{
    Point centre;
    double rayon;
    Color couleur;
};
```

```
Cercle c;
c.centre.x = 0.5;
c.couleur = RED;
Point pt;
pt.x = pt.y = 5.5;
c.centre = pt;
```

```
Point p1={1,2}, p2;
p2 = p1 // OK, recopie champ
↪ a champ
```



Rappels

Pourquoi les structures ?

Définition

Manipulation

TP du jour

```
Point pt;  
pt.x = pt.y = 5.5;
```

```
Cercle c;  
c.couleur = RED;  
c.rayon = 3;  
c.centre.x = 5.5;  
c.centre.y = 5.5  
//ou  
c.centre = pt;
```

```
Point pt = {5.5};  
Cercle c = {pt, 3, RED};
```

```
Cercle c={{5.5,5.5},3,RED};  
// ERREUR
```

L'ordre des éléments pour l'initialisation est l'ordre de définition des champs dans la structure.

Les structures fonctionnent comme des types classiques :

```
// Structure comme argument
void affiche(Point p){
    cout << p.x << " " << p.y << endl;
}
```

```
// Structure comme valeur de retour
Point milieu(Point p, Point q){
    Point m;
    m.x = (p.x+q.x)/2;
    m.y = (p.y+q.y)/2;
    return m;
}
```

Les structures fonctionnent comme des types classiques :

```
// Passage par référence
void init(Point &p){
    p.x = 0;
    p.y = 0;
}
```

Rappels

Pourquoi les structures ?

Définition

Manipulation

TP du jour

- ▶ Simulation de système planétaire
- ▶ Système physique à plusieurs objets
- ▶ Utilisation des structures

