Support de cours

Cours:

## Initiation à la programmation (en C++)

Vidéo:

## Structures (partie 3)

Concepts (extraits des sous-titres générés automatiquement) :

**Valeur des champs d'une variable. Valeur de retour de la fonction. Paramètres de type. Valeurs des champs. Exemple complet. Code de la fonction naissance. But de cette fonction. Instruction suivante. Variable locale. Fonctions. Champ sexe de la variable. Valeur de type. Type de retour de la fonction. Cas de mon exemple. Besoin d'aucune valeur.**

vers la recherche de séquences vidéo
(dans Initiation à la programmation (en C++).)

vers la vidéo

EPFL

# Structures
## (Partie 3)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

# Exemple complet (1/3)

```
struct Personne {
  string nom;
  double taille;
  int    age;
  char   sexe;
};
```

```
int main()
{
  Personne untel( naissance() );

  anniversaire(untel); // un an de plus

  affiche(untel);
  cout << endl;

  return 0;
}
```

Je vais maintenant détailler un exemple complet qui met en œuvre, notamment, des fonctions ayant des paramètres de type "structure".

notes

résumé

0m 1s

## Exemple complet (1/3)

```
struct Personne {
  string nom;
  double taille;
  int    age;
  char   sexe;
};
```

```
int main()
{
  Personne untel( naissance() );

  anniversaire(untel); // un an de plus

  affiche(untel);
  cout << endl;

  return 0;
}
```


Je vais commencer par reprendre le code de ma structure "Personne", ici, avec, je vous le rappelle : un nom, une taille, un âge et un sexe comme données. La fonction "main" va commencer par déclarer une variable "untel" de type "Personne" : je peux donc représenter "untel" de la façon suivante.

notes

résumé

0m 12s

# Exemple complet (2/3)

```
Personne naissance() {
  Personne p;

  cout << "Saisie d'une nouvelle personne" << endl;
  cout << "  Entrez son nom : ";
  cin >> p.nom;
  cout << "  Entrez sa taille (m) : ";
  cin >> p.taille;
  cout << "  Entrez son age : ";
  cin >> p.age;
  do {
    cout << "  Homme [M] ou Femme [F] : ";
    cin >> p.sexe;
  } while ((p.sexe != 'F') and (p.sexe != 'M'));

  return p;
}
```

Et "untel" va être initialisé en utilisant la valeur de retour de la fonction "naissance", qui est appelée ici. Voici le code de la fonction naissance. Le but de cette fonction est de demander la valeur des champs d'une variable de type "Personne" à l'utilisateur, c'est-à-dire un nom, une taille, un âge et un sexe, et de retourner les valeurs des champs entrés par l'utilisateur sous la forme d'une structure. Cette fonction n'a donc besoin d'aucune valeur pour pouvoir fonctionner : elle n'a donc pas de paramètre.

notes

résumé

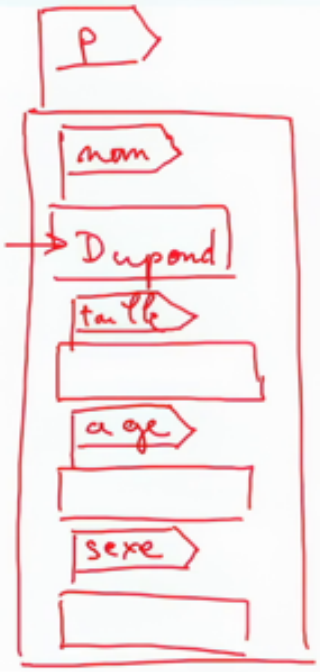
0m 33s

# Exemple complet (2/3)

```
Personne naissance() {
  Personne p;

  cout << "Saisie d'une nouvelle personne" << endl;
  cout << "  Entrez son nom : ";
  cin >> p.nom;
  cout << "  Entrez sa taille (m) : ";
  cin >> p.taille;
  cout << "  Entrez son age : ";
  cin >> p.age;
  do {
    cout << "  Homme [M] ou Femme [F] : ";
    cin >> p.sexe;
  } while ((p.sexe != 'F') and (p.sexe != 'M'));

  return p;
}
```




En revanche, elle doit retourner une valeur de type "structure", plus exactement de type "Personne" ici, et donc le type de retour de la fonction "naissance" et "Personne". Comment fonctionne "naissance" exactement? On commence par déclarer une variable locale qui s'appelle "p" de type "Personne", et "p" peut être représentée comme la variable "untel". Pour l'instant les champs de "p" ne sont pas initialisés, on ne sait donc pas ce qu'ils contiennent mais la fonction "naissance" commence par demander à l'utilisateur d'entrer un nom et ce nom va être rangé directement dans le champ "nom" de la variable "p", c'est-à-dire ici. Supposons, pour l'exemple, que l'utilisateur rentre le nom "Dupond",

notes

résumé

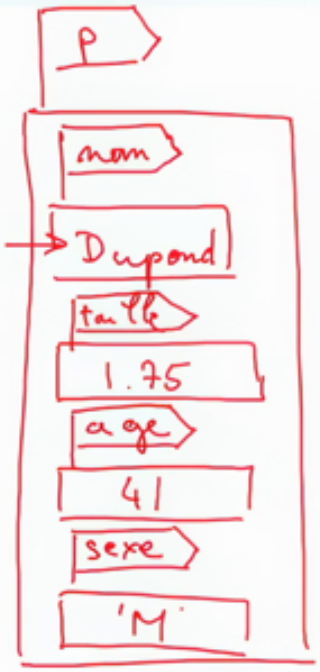
1m 13s

## Exemple complet (2/3)

```
Personne naissance() {
  Personne p;

  cout << "Saisie d'une nouvelle personne" << endl;
  cout << "  Entrez son nom : ";
  cin >> p.nom;
  cout << "  Entrez sa taille (m) : ";
  cin >> p.taille;
  cout << "  Entrez son age : ";
  cin >> p.age;
  do {
    cout << "  Homme [M] ou Femme [F] : ";
    cin >> p.sexe;
  } while ((p.sexe != 'F') and (p.sexe != 'M'));

  return p;
}
```




la fonction continue en demandant une taille et un âge à l'utilisateur : reprenons les mêmes valeurs que dans l'exemple pécédent. La fonction continue en demandant à l'utilisateur d'entrer le caractère "M" ou "F" pour le champ sexe de la variable "p" et s'assure que l'utilisateur rentre bien soit "M", soit "F" en utilisant une boucle "do while", supposons que la personne soit ici un homme,

notes

résumé

1m 59s

## Exemple complet (1/3)

```
struct Personne {
  string nom;
  double taille;
  int    age;
  char   sexe;
};
```

```
int main()
{
  Personne untel( naissance() );

  anniversaire(untel); // un an de plus

  affiche(untel);
  cout << endl;

  return 0;
}
```


et enfin, la fonction finit en retournant la valeur contenue dans la variable "p", c'est-à-dire l'ensemble des valeurs stockées dans ces champs. Revenons à l'appel de la fonction "naissance", c'est-à-dire

notes

résumé

2m 25s

# Exemple complet (3/3)

```
void anniversaire(Personne& p) {
  ++(p.age);
}

void affiche(Personne const& p) {
  cout << p.nom    << ", ";
  switch (p.sexe) {
    case 'M': cout << "homme"; break;
    case 'F': cout << "femme"; break;
    default : cout << "non connu"; break;
  }
  cout << ", "
       << p.taille << " m, "
       << p.age    << " an";
  if (p.age > 1) {
    cout << 's';
  }
}
```

ici, que va-t-il se passe exactement? Les valeurs des champs de la variable "p" vont être recopiés dans les champs de la variable "untel", c'est-à-dire que le champ "nom" d'untel va prendre la valeur "Dupond", le champ "taille": 1,75 Âge: 41 Et le champ sexe, le caractère "M". L'instruction suivante dans la fonction "main" est celle-ci, c'est-à-dire qu'on va appeler la fonction "anniversaire" en lui passant la variable "untel" en argument. Le code de la fonction "anniversaire" est ici. Le but de la fonction "anniversaire" est d'ajouter 1 au champ "âge" de la variable passée en argument : cette fonction n'a donc qu'un seul paramètre qui sera de type "Personne", puisque la variable "untel" était de type "Personne" et on va utiliser un passage par référence puisque la fonction doit modifier

notes

résumé

2m 39s

## Exemple complet (1/3)

EPFL

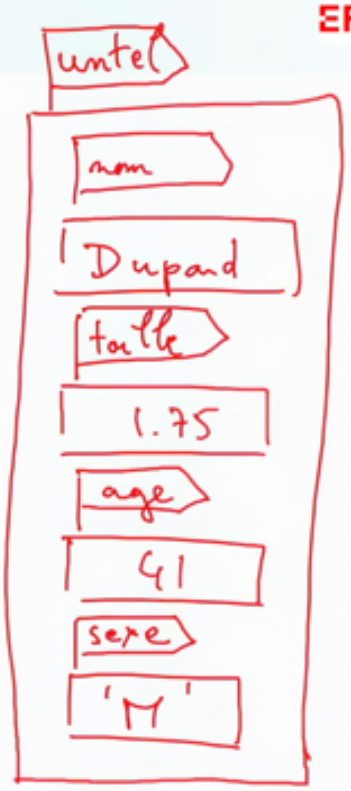
```
struct Personne {
  string nom;
  double taille;
  int    age;
  char   sexe;
};
```

```
int main()
{
  Personne untel( naissance() );

  anniversaire(untel); // un an de plus

  affiche(untel);
  cout << endl;

  return 0;
}
```




la valeur de la variable à passer en argument. Cette fonction ne doit envoyer aucune valeur : son type de retour sera donc "void" et quand on exécute le corps de la fonction sur l'exemple,

notes

résumé

3m 37s

# Exemple complet (3/3)

```
void anniversaire(Personne& p) {
  ++(p.age);
}

void affiche(Personne const& p) {
  cout << p.nom    << ", ";
  switch (p.sexe) {
    case 'M': cout << "homme"; break;
    case 'F': cout << "femme"; break;
    default : cout << "non connu"; break;
  }
  cout << ", "
       << p.taille << " m, "
       << p.age    << " an";
  if (p.age > 1) {
    cout << 's';
  }
}
```

on va changer ce 41 et La valeur contenue par le champ "âge" de la variable "untel", en 42. On continue dans le corps de la fonction "main", c'est-à-dire qu'on va passer à cette instruction et appeler la fonction "affiche" en lui passant la variable "untel" en argument. Le code de la fonction "affiche" est ici. Le but de cette fonction "affiche" est d'afficher la valeur des champs de la variable passée en argument. Cette fonction n'aura donc qu'un seul paramètre de type "Personne" et ne devra pas modifier la valeur de la variable passée en argument. On a utilisé ici le mot-clé "const", suivi de &, comme pour le passage par référence, entre le nom du type du paramètre, le nom du paramètre lui-même : il s'agit là d'une optimisation à laquelle vous pouvez ne pas attacher trop d'importance pour l'instant. Cette optimisation évite de faire une copie car je vous rappelle

notes

résumé

3m 51s

# Exemple complet (3/3)

```
void anniversaire(Personne& p) {
  ++(p.age);
}

void affiche(Personne const& p) {
  cout << p.nom    << ", ";
  switch (p.sexe) {
    case 'M': cout << "homme"; break;
    case 'F': cout << "femme"; break;
    default : cout << "non connu"; break;
  }
  cout << ", "
       << p.taille << " m, "
       << p.age    << " an";
  if (p.age > 1) {
    cout << 's';
  }
}
```

Dupond,

qu'un passage par valeur ferait une copie, une copie coûteuse d'une structure qui contient beaucoup de gens. La fonction va simplement afficher les valeurs des différents champs du paramètre. A noter qu'on utilise ici un "switch" pour afficher si la personne est un homme ou une femme. Cette instruction "switch" est expliquée dans les compléments. On aurait pu utiliser ici aussi une instruction "if". Vous pourrez vérifier par vous-même que dans le cas de mon exemple,

notes

résumé

4m 59s

# Exemple complet (1/3)

EPFL

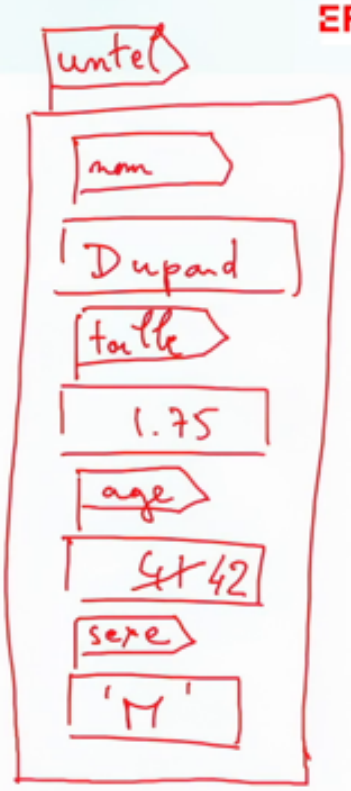
```
struct Personne {
  string nom;
  double taille;
  int    age;
  char   sexe;
};
```

```
int main()
{
  Personne untel( naissance() );

  anniversaire(untel); // un an de plus

  affiche(untel);
  cout << endl;

  return 0;
}
```




la fonction va afficher ceci. Une fois la fonction "affiche" exécutée, on va revenir dans la fonction "main",

notes

résumé

5m 28s

## Affectation de structures

Une variable de type composé `struct` peut être directement affectée par une variable du même type

Exemple :

```
Personne p1 = { "Dupond", 1.75, 41, 'M' };
Personne p2;
p2 = p1;
```

La valeur de chaque champ de `p1` est affectée au champ correspondant de `p2`

☞ L'instruction `p2 = p1` est équivalente à la séquence d'instructions

```
p2.nom = p1.nom;        p2.taille = p1.taille;
p2.age = p1.age;        p2.sexe   = p1.sexe;
```

c'est-à-dire ici. On va exécuter l'instruction "return" et terminer le programme. On peut affecter une variable de type "structure" à une variable du même type, par exemple je déclare ici une variable p1 de type "Personne" en initialisant ces champs à ces valeurs.

notes

résumé

5m 39s

## Remarque

EPFL

Note : l'affection (=) est la seule opération que l'on peut faire de façon globale sur les `struct`.

p2 = p1;

On **NE** peut **NI** les comparer (`p1 == p2`),

**NI** les afficher (`cout << p1`) globalement.

Solutions :
Dans ces cas là, il faut le faire champ par champ

Exemple :

```
cout << p1.nom << ", " << p1.taille << ", "
     << p1.age << ", " << p1.sexe ;
```

MIEUX : faire une *fonction* ! !

Ensuite, je déclare une variable p2, également de type "Personne", mais sans initialiser la valeur de ces champs, et dans cette dernière instruction qui est une affectation, je vais recopier la valeur des champs de la variable p1 dans les champs de la variable p2, c'est-à-dire que cette affectation est équivalente à ces quatre affectations, qui recopieraient, champ par champ, la variable p1 dans p2, c'est-à-dire par exemple ici, je recopie la valeur du champ "nom" de p1 dans le champ "nom" de p2. Mais attention : l'affectation est la seule opération qu'on peut faire de façon globale sur des variables de type "structure". On ne peut pas les comparer, ni les afficher globalement. Si je reprends l'exemple précédent, j'avais le droit d'écrire p2=p1, qui est une affectation. Par-contre, je n'ai pas le droit d'écrire p1==p2 pour comparer p1 et p2, ni d'écrire par exemple coutp1 pour afficher les valeurs des champs de p1. Dans ce cas-là, je suis obligé de procéder

notes

résumé

6m 1s

## Retour à l'exemple du début

EPFL

Les structures sont particulièrement utiles pour les tableaux hétérogènes :

☞ tableaux de structures

Exemple :

```
struct Personne {
  string nom;
  double taille;
  int    age;
  char   sexe;
};

vector<Personne> personnes = {
    { "Dupond",  1.75, 41, 'M' },
    { "Dupont",  1.75, 42, 'M' },
    { "Durand",  1.85, 26, 'M' },
    { "Dugenou", 1.70, 38, 'F' },
    { "Pahut",   1.63, 22, 'F' }
};
```

| Nom | Taille | Âge | Sexe |
|---|---|---|---|
| Dupond | 1.75 | 41 | M |
| Dupont | 1.75 | 42 | M |
| Durand | 1.85 | 26 | M |
| Dugenou | 1.70 | 38 | F |
| Pahut | 1.63 | 22 | F |

champ par champ, par exemple je peux utiliser cette instruction pour afficher les quatre champs de p1 mais évidemment, il vaut mieux écrire une fonction qui ferait cette comparaison ou cet affichage, c'est ce que nous avions fait avec la fonction "affiche" d'un exemple précédent. Notez enfin que si les opérateurs == et ne sont pas définis a priori par les structures, en C++ un programmeur a la possibilité de les définir lui-même, c'est ce qu'on appelle la surcharge d'opérateur mais ceci sort du cadre de ce cours. Pour le moment, rappelez-vous que vous devez faire la comparaison ou l'affichage champ par champ. Revenons sur l'exemple du début de cette vidéo. Si vous vous souvenez, je voulais stocker en mémoire l'ensemble de ces valeurs ; et bien maintenant, je vais pouvoir utiliser un tableau dont les éléments seront de type "structure". Donc je vais commencer par déclarer la structure "Personne" qui regroupe les quatre champs qui m'intéressent, c'est-à-dire le nom, la taille, l'âge et le sexe d'une personne. Et je vais déclarer un tableau que j'ai appelé "Personnes" ici,

notes

résumé

7m 13s

dont les éléments seront de type "Personne". Maintenant, chaque ligne de cette table va correspondre à un élément de mon tableau. Chaque colonne de la table va correspondre à un différent champ des éléments du tableau. Je peux représenter le début de mon tableau "Personnes" de cette façon : Chaque élément du tableau "Personnes" va contenir un champ "nom", un champ "taille", etc. Tout ceci constitue le premier élément du tableau "Personnes". Et cette affectation va mettre, entre autres, ces valeurs dans le premier élément du tableau "Personnes", c'est-à-dire que Dupond va aller dans ce champ, 1,75 dans ce champ, 41 ici et M ici. 41 ici et M ici.

notes

résumé

8m 25s