Information, Calcul, Communication (partie programmation) : Structures

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Rappel du calendrier

		MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11
1	11.09.25	--	-1	prise en main	Bienvenue/Introduction
2	18.09.25	1. variables	0	variables / expressions	variables / expressions
3	25.09.25	2. if	0	if – switch	if – switch
4	02.10.25	3. for/while	0	for / while	for / while
5	09.10.25	4. fonctions	0	fonctions (1)	fonctions (1)
6	16.10.25		1	fonctions (2)	fonctions (2)
-	23.10.25				
7	30.10.25	5. tableaux (vector)	1	vector	vector
8	06.11.25	6. string + struct	1	array / string	array / string
9	13.11.25		2	structures	structures
10	20.11.25	7. pointeurs	2	pointeurs	pointeurs
11	27.11.25		-	entrées/sorties	entrées/sorties
12	04.12.25		-	erreurs / exceptions	erreurs / exceptions
13	11.12.25		-	révisions	théorie : sécurité
14	18.12.25	8. étude de cas	-	révisions	Révisions
					(ne sont pas sur le MOOC)

Jean-Cédric Chappelier
& Jamila Sam

Objectifs du cours d'aujourd'hui

- Rappels sur les `struct` :
  - nouveau **type**
  - initialisation, utilisation
-  Complément C++17 sur `struct`
- Rappels sur `typedef`
- Etude de cas

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Données structurées

Âge
20
35
26
38
22

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

Les tableaux permettent de représenter des structures de données **homogènes**, c'est-à-dire des listes constituées d'éléments qui sont tous du **même type**.

Les **structures** permettent de regrouper des types **hétérogènes** (mais en nombre limité, connu au préalable !).

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Création d'un nouveau type

La syntaxe pour déclarer un type « structure » est la suivante :

```
struct Nom_du_type {
    type_1 identificateur_1 ;
    type_2 identificateur_2 ;
    ...
} ;
```

Exemples :

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

```
struct Complexe {
  double x;
  double y;
};
```

```
struct Simple { int souschamp1; double souschamp2; };

struct Compliquee {
  vector<double> champ1;
  int champ2;
  Simple champ3;
};
```


Jean-Cédric Chappelier
& Jamila Sam
EPFL

Déclaration/Initialisation d'une variable

Une fois le type de la structure déclaré, on peut utiliser son nom comme tout autre type pour déclarer des variables.

```
struct Complexe {
  double x;
  double y;
};

Complexe z;
```

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

Personne untel({ "Dupontel", 1.75, 20, 'M' });
```

On peut également utiliser cette syntaxe pour l'affectation :

```
untel = { "Dupontel", 1.75, 20, 'M' };
```


Jean-Cédric Chappelier
& Jamila Sam
EPFL

Accès aux champs d'une structure

On peut accéder aux champs d'une structure en utilisant la syntaxe suivante :

structure.champ

Exemples :

```
untel.taille = 1.75;

++(untel.age); // déjà un an de plus !

cout << untel.sexe << endl;
```


Jean-Cédric Chappelier
& Jamila Sam

Affectation de structures

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

Exemple :

```
Personne p1 = { "Durand", 1.75, 20, 'M' };
Personne p2;
p2 = p1;
```

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

On **NE** peut **NI** les comparer (`p1 == p2`),
NI les afficher (`cout << p1`) globalement.

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Fonction à plusieurs valeurs de retour

On sait que les fonctions ne peuvent retourner qu'une seule valeur.

Comment faire lorsque l'on veut « retourner » *plusieurs* valeurs avec une fonction ?

Par exemple, quotien et reste d'une division entière :

```
?? quotien, reste ?? = division_entiere( 141, 17 );
```


Jean-Cédric Chappelier
& Jamila Sam

Fonction à plusieurs valeurs de retour

On sait que les fonctions ne peuvent retourner qu'une seule valeur.

Comment faire lorsque l'on veut « retourner » *plusieurs* valeurs avec une fonction ?

Par exemple, quotien et reste d'une division entière :

```
?? quotien, reste ?? = division_entiere( 141, 17 );
```

Solutions :

1. renvoyer une **structure** contenant les valeurs à retourner ;
2. **passer** les « variables retour » **par référence** et les affecter à l'intérieur de la fonction ; f(int use, double& a_remplir)
3. renvoyer un **tableau dynamique** (`vector`), si les valeurs à retourner sont de même type (homogène) ; (ou array)
4. combiner 1 et 3 : structure avec champs `vector` ou (au choix) `vector` de structures (comme dans le tableau illustratif de départ : `vector<Personne>`)

Jean-Cédric Chappelier
& Jamila Sam
EPFL

C++17 : les « structured bindings »

C++17 introduit une nouvelle syntaxe permettant
l'appariment terme à terme de types de données composés
comme par exemple les `struct`
(sans expliciter ces types en question) :

```
auto [ ...variables... ] = expression de type composé ;
```

Cela permet entre autres de simplifier l'écriture de certains appels de fonctions.

Par exemple :

```
auto [ x, y ] = creer_complexe_polaires( 2.5, M_PI / 3.0);
```

```
auto [ quotien, reste ] = division_entiere( 141, 17 );
```

double x;
double y;
Complexe z = creer...(...);
x = z.x;
y = z.y;

Jean-Cédric Chappelier
& Jamila Sam
EPFL

```
Complexe z(creer_complexe_polaires(...));
```

```
Complexe creer_complexe_polaires(double r,
                                 double t)
{
    Complexe z;
    ...
    return z;
}
```

Les structures

Déclaration du type correspondant :

```
struct Nom_du_type {
        type1 champ1 ;
        type2 champ2 ;
        ...
};
```

Déclaration d'une variable :

```
Nom_du_type identificateur;
```

Déclaration/Initialisation d'une variable :

```
Nom_du_type identificateur = { val1, val2, ... };
```

Accès à un champ donné de la structure :

```
identificateur.champ
```

Affectation globale de structures :

```
identificateur1 = identificateur2
```

Alias de types

On peut utiliser la commande `typedef` pour
donner un autre nom (alias) à ce type

Syntaxe :

`typedef` *`type alias`*`;`

Exemples :

```
typedef int Distance;
typedef vector<double> Vecteur;
typedef vector<Vecteur> Matrice;

Distance ma_longueur(0);
Matrice rotation(3, Vecteur(3, 1.0));
Vecteur produit_vectoriel(Vecteur, Vecteur);
```

typedef array<double, 3> Tableau;

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Etudes de cas

- « exercice 0 » : `Personne`
- revisite du 2e exercice (du MOOC et du semestre) : la fondue...

Jean-Cédric Chappelier
& Jamila Sam