La langage C++

M4105C

Sylvain Jubertie
sylvain.jubertie@univ-orleans.fr

Organisation du module

8 semaines

- Semaines 1-3 : Cours/TD/TP
- Semaine 4 : Cours/TD/Examen en séance de TP
- Semaines 5-7 : Cours/TD/TP
- Semaine 8 : Cours/TD/Examen en séance de TP

Évaluation surprise

Suivant attitude des étudiants. . .

Historique

Langage développé par Bjarne Stroustrup au début des années 1980 comme extension du C : *C with classes*, langage standardisé

Caractéristiques

Langage multi-paradigmes

- programmation impérative
- programmation objet
- programmation générique

Domaines

- embarqué : Arduino, micro-controleurs
- systèmes d'exploitation : Windows ! (noyau en C)
- jeux vidéo : OpenGL, Vulkan, moteurs physiques
- calcul : banque, finance, simulations scientifiques, ...
- ...

Ressources en ligne

- isocpp.org : standard C++
- cppreference.com : API
- Coliru : compilateur C++ en ligne
- Godbolt : compilateur C++ en ligne + asm

hello.c

```
#include <iostream>

int main()
{
   std::cout << "Hello C++" << std::endl;
   return 0;
}
```

Compilation avec GCC

```
g++ -o hello hello.cpp
```

Exécution

```
./hello
```

Types primitifs

- types entiers signés : `char`, `short`, `int`, `long`, ...
- types entiers non signés : préfixe `unsigned`
- booléen : `bool`
- types flottants : `float`, `double`
- constantes : modificateur `const`

Initialisation des variables de types primitifs

1. Initialisation par affectation :

```
int a = 5;
float f = 1.5f;
```

2. Initialisation par constructeur :

```
int a( 5 );
float f( 1.5f );
```

3. Initialisation uniforme (C++11) :

```
int a{ 5 };
float f{ 1.5f };
```

Tableaux statiques (pile)

Déclaration :

```
elementtype tabname[ tabsize ];
```

Initialisation des tableaux

```
int t0[ 10 ]; // 10 entiers, valeurs non définies
int t1[ 4 ] = {}; // 0, 0, 0, 0
float t2[] = { 1.0f, 2.0f, 3.0f, 4.0f }; // float[4]
float t3[ 4 ] = { 1.0f }; // 1.0f, 0.0f, ...
```

Tableaux et bibliothèque standard

- Privilégier l'utilisation de `std::array` (cf. section Bibliothèque standard) à la place des tableaux statiques C.
- Les tableaux dynamiques sont présentés dans la sous-section *Gestion mémoire dynamique*.

Chaînes de caractères

Type complexe `std::string` de la bibliothèque standard :

Initialisation

```
std::string s0 = "Hello C++";
std::string s1( "Hello" );
std::string s2{ "Hello" };
```

Fonctions

- comme en C.
- passage par référence en plus : la variable passée par référence est modifiable dans la fonction. Equivalent à un passage par pointeur mais avec une syntaxe plus simple. Une référence ne peut être nulle.

Exemple

```
void fct( int a ) { ... } // Passage par copie.
void fct( int * p_a ) { ... } // Passage par pointeur.
void fct( int & a ) { a = 3; ... } // Passage par référence.
```

const

Le mot clé const peut être appliqué aux arguments d'une fonction.

Exemple

```
void fct( int const a ); // a: copie non modifiable.
void fct( int const * p_a ); // copie du pointeur modifiable
                             // mais valeur pointée non modifiable.
void fct( int * const p_a );
// copie du pointeur non modifiable mais valeur pointée modifiable.
void fct( int const * const p_a ); // rien n'est modifiable.
void fct( int & a ) // Passage par référence, a modifiable.
void fct( int const & a )
  // Passage par référence, a non modifiable.
```

Types complexes

2 méthodes de définitions possibles :

1. `struct`
2. `class`

Structures

Déclaration similaire au C.

Par défaut attributs accessibles publiquement.

Définition

```
struct Student
{
  unsigned int id;
  std::string firstname;
  std::string lastname;
  // ...
}; // Attention au ; à la fin de la définition !
```

Instantiation/Accès

```
Student s0;
s0.id = 3;

Student s1{ 42, "Arthur", "Accroc" };
std::cout << s1.firstname << std::endl;
```

Classes

- Définition similaire à une structure.
- Par défaut attributs privés donc non accessibles de l'extérieur de la classe.
- Possibilité de déclarer les attributs publiques : `public`.
- Sinon nécessite des constructeurs/accesseurs pour manipuler les attributs.
- Accès aux attributs publiques comme pour une structure.

Exemple : attributs privés (par défaut)

```
class Student
{
private: // attributs privés
   unsigned int id;
   std::string firstname;
   std::string lastname;
   // ...
};
```

Exemple : attributs publiques

```
class Student
{
public: // attributs accessibles de l'extérieur de la classe.
  unsigned int id;
  std::string firstname;
  std::string lastname;
  // ...
};
```

Exemple : attributs publiques et privés

```
class Student
{
private:
  unsigned int id;
public:
  std::string firstname;
  std::string lastname;
  // ...
};
```

Encapsulation

Encapsulation = Attributs + Méthodes associés.

Vie d'une instance

1. Instanciation : constructeurs, initialisation des attributs
2. Utilisation : méthodes
3. Destruction : destructeur, préparation avant libération de la mémoire

Constructeurs

- Par défaut : sans arguments.
- Avec arguments.
- Par copie : copie des attributs d'une autre instance.
- Par déplacement (move constructor) : appropriation d'une autre instance.

Exemple

```
class Student
{
public:
  // constructeur par défaut.
  Student() { ... }

  // constructeur avec arguments.
  Student( ... ) { ... }

  // constr. de recopie, on ne modifie pas l'original donc const.
  Student( Student const & s ) { ... }

  // move constr., rvalue reference.
  Student( Student && s ) { ... }
```

Constructeur de recopie

- Si le constructeur de recopie n'est pas définie par défaut, alors un constructeur de recopie par défaut est utilisé : tous les attributs sont copiés (comme pour les structures en C).
- L'instance à copier est passée par référence : `&`.
- Si l'instance à copier n'est pas modifiée (attributs non modifiés) par le constructeur de recopie, alors la déclarer `const`

Destructeur

- Appelé automatiquement lors de la sortie de la zone de portée de la variable.
- Pas d'argument.
- Utilisé pour modifier des attributs statiques (compteur), libérer des ressources (mémoire, fichier : RAII), ...

Exemple

```
...
  // Destructeur
  ~Student() { ... }
...
```

Exemple

```
...
{  // Début de la zone de portée.
   Student s( ... ); // Instanciation
   std::cout << s.firstname << std::endl;
   ...
}  // Fin de la zone de portée, appel au destructeur  Student
```