UE 3.1 C++ : Introduction

Simon Rohou

2020/2021

V1.00
Supports de cours disponibles sur `www.simon-rohou.fr/cours/c++/`
Slides inspirées du cours de Olivier Marguin (Univ. Lyon)

ENSTA BRETAGNE

UE 3.1 : C++ pour la robotique

Introduction

Éléments du langage

Fonctions

Compléments

Section 1

UE 3.1 : C++ pour la robotique

Objectifs du cours

UE 3.1 : C++ pour la robotique

Objectif : être autonome sur un projet de robotique développé en C++

- apprendre les bases du C++
- approcher des applications liées à la robotique
- comprendre comment implémenter un driver
- comprendre la gestion de mémoire
- s'interfacer avec des bibliothèques déjà existantes

Planning prévisionnel

UE 3.1 : C++ pour la robotique

1. (4cr) **Lundi 07/09 :** Introduction au C++
2. (4cr) **Lundi 14/09 :** Programmation orientée objet
3. (4cr) **Lundi 28/09 :** Tableaux et pointeurs
4. (4cr) **Lundi 02/11 :** Fichiers et conteneurs
5. (4cr) **Lundi 09/11 :** Outils de développement

6,7,8. (12cr) **Lundi 16/11, 23/11, 30/11 :** Projet Modèle Numérique de Terrain

9. (4cr) **Lundi 09/11 :** Évaluation

ENSTA BRETAGNE

Section 2

Introduction

Le langage C++
Introduction

- début en 1983, aujourd'hui normalisé (C++20)
- langage de programmation **compilé**
- amélioration du langage C
- programmation sous de multiples paradigmes
  - programmation procédurale
  - programmation orientée objet (POO)
  - programmation générique
- **bonnes performances**
- très utilisé, notamment en robotique

Programmer en C++
Introduction

1. Éditer le programme avec votre éditeur de texte favori
   - dans ce cours, nous vous proposons *Sublime Text 3*

2. Compiler le programme
   - en utilisant un **compilateur** tel que g++
   - ou toute surcouche de compilation telle que `cmake`

3. Exécuter le programme

Programmer en C++
Introduction

1. Éditer le programme avec votre éditeur de texte favori
   - dans ce cours, nous vous proposons *Sublime Text 3*

2. Compiler le programme
   - en utilisant un **compilateur** tel que `g++`
   - ou toute surcouche de compilation telle que `cmake`

3. Exécuter le programme

Contrairement à des langages interprétés tels que Python, C++ requiert une phase de compilation avant l'exécution : il y a génération d'un **fichier binaire**.

La compilation
Introduction

Programmer un robot =
Fournir une série d'instructions qu'il doit exécuter :

- instructions écrites dans un langage *évolué*...
- ...puis traduites en **langage machine**

C'est la **compilation**, réalisée par un programme : le **compilateur**.

Nous utiliserons g++, largement utilisé par les développeurs.

Éléments de base d'un programme C++

Introduction

- le C++ est sensible à la casse
  (différence entre minuscules et majuscules)
- pour **rendre un programme lisible**, il faut le commenter :

```
1       int 3; // commentaire sur une ligne
2
3       /* commentaire
4          sur
5          plusieurs
6          lignes */
```

- nous verrons plus tard dans ce cours des outils de **documentation**
- tout programme comporte une fonction appelée `main()`, par laquelle commence l'exécution

Structure d'un programme C++

Introduction

Un programme écrit en C++ se compose généralement de :

- fichiers-sources contenant les **instructions**.
  Par convention, leur extension est `.cpp`.
- fichiers-sources contenant des **déclarations**.
  Par convention, leur extension est `.h` (pour *header*).

Un fichier `.h` regroupe des déclarations communes à plusieurs `.cpp`. Les `.cpp` incluent les `.h` avec une *directive de compilation* `#include` :

```
1  // chevrons pour les headers des bibliothèques standards :
2  #include <unfichier.h>
3
4  // quotes pour les headers du projet :
5  #include "unfichier.h"
```

Hello World

Introduction

Programme principal : `main.cpp`

```
1   #include <cstdlib>
2   #include <iostream>
3
4
5   using namespace std;
6
7
8   int main()
9   {
10
11    cout << "Hello World" << endl;
12
13
14    return EXIT_SUCCESS;
15  }
```

Hello World

Introduction

Programme principal : `main.cpp`

```
1   #include <cstdlib> // bibliothèque générique standard
2   #include <iostream> // bibliothèque d'entrées/sorties
3
4   // On se place dans l'espace de nommage std
5   using namespace std;
6
7   // Programme principal
8   int main()
9   {
10    // Affichage à l'écran :
11    cout << "Hello World" << endl;
12
13    // Retourne une constante entière (succès d'exécution) :
14    return EXIT_SUCCESS;
15  }
```

Hello World

Introduction

Programme principal : `main.cpp`

```
1   #include <cstdlib> // bibliothèque générique standard
2   #include <iostream> // bibliothèque d'entrées/sorties
3
4   // On se place dans l'espace de nommage std
5   //using namespace std;
6
7   // Programme principal
8   int main()
9   {
10    // Affichage à l'écran :
11    std::cout << "Hello World" << std::endl;
12
13    // Retourne une constante entière (succès d'exécution) :
14    return EXIT_SUCCESS;
15  }
```

Hello World : compilation

Introduction

Programme principal : `main.cpp` (1 seul fichier pour le moment)

Compilation avec le programme g++ :

```
g++ main.cpp -o output
```

- en entrée : tous les fichiers sources `.cpp` à compiler
- l'option `-o` spécifie le nom du binaire à générer (*output*)

Résultat :

```
./output
```

```
Hello World
```

Section 3

Éléments du langage

Variables
Éléments du langage

Une variable se caractérise par :

- un **nom**
- un **type** qui détermine la nature de la variable (entier, caractère, etc.)
- une **valeur** qui peut être modifiée par le programme

Durant l'exécution, une **adresse** est attachée à une variable : c'est un nombre entier indiquant l'emplacement de la valeur de la variable dans la mémoire de la machine.

Nous verrons plus tard comment manipuler ces adresses.

Types élémentaires
Éléments du langage

- **vide** : `void`

  (aucune variable ne peut être de ce type)

- **entiers** :

  `char` : 1 octet, valeurs de -2^7 à $2^7 - 1$
  `short` : 2 octets, valeurs de -2^{15} à $2^{15} - 1$
  `long` : 4 octets, valeurs de -2^{31} à $2^{31} - 1$
  `int` : peut être équivalent à `short` ou bien `long`

- **réels** :

  `float` : 4 octets, précision d'environ 7 chiffres
  `double` : 8 octets, précision d'environ 15 chiffres
  `long double` : 10 octets, précision d'environ 18 chiffres

- **caractères** : `char`

  représentation de caractères `'a'`, `'M'`, `'$'`, `'\t'`, `'\n'`...

- **booléens** : `bool` (`true` ou `false`)

Déclaration de variables

Éléments du langage

Le **type** d'une variable se définit à sa déclaration.
Par exemple :

```
1 float heading;
2 int robots_nb = 4, landmarks_nb; // deux variables du même type
3 float robot_speed = 10.; // variable initialisée
4 const float robot_length = 4.; // constante symbolique
```

- toute variable doit être définie avant utilisation
- une déclaration peut se faire à tout moment
  - mais on privilégie les déclarations en en-tête de blocs
- une variable non initialisée peut prendre n'importe quelle valeur

Visibilité d'une variable

Éléments du langage

Les variables déclarées dans un bloc sont **locales** à ce bloc.

```
1  {
2    ... // intérieur d'un bloc
3  }
```

Un exemple :

```
1  int k = 1;
2
3  {
4    k = 2;
5    int i = 3, j; // déclaration de i, j, et initialisation de i
6    j = i + 9;
7  }
8
9  k = 4;
10 i = 3; /* opération impossible, car en dehors du bloc
11           de déclaration de i */
```

Variable globale : déclarée en dehors de tout bloc, visible dans tout le fichier

Opérations
Éléments du langage

Les **opérateurs arithmétiques** sont utilisables : +, -, *, /, %
Exemples :

```
19.0 / 5.0  // 3.8
19 / 5      // 3
19 % 5      // 4
```

Incrémentations :

- `i++` ajoute 1 à la valeur de `i`
- `i--` retranche 1

Opérateurs d'affectation :

```
i = 1;
i = j = k = 1;
i += 3; // équivaut à i = i + 3
```

Opérateurs d'entrées-sorties

Éléments du langage

Opérateurs << et >> sont utilisés en conjonction avec les objets `std::cout` et `std::cin` de la bibliothèque `iostream`.

```
1  #include <iostream.h> // inclusion du header de iostream
2  using namespace std;
3  ...
4  int x;
5  cin >> x; // lecture au clavier de x
6  // (pour l'exemple, on entre 3)
7  cout << "x=" << x << endl; // affiche "x=3" avec saut de ligne
```

Formatage (éléments à insérer entre chevrons) :

- `endl` : retour chariot (passage à la ligne).
- `setprecision(p)` : fixe le nombre de chiffres affichés.

Inclure la librairie `<iomanip.h>`.

Instruction conditionnelle et opérateurs booléens

Éléments du langage

Forme :

```
1   if( <expression booléenne> )
2   {
3
4   }
5
6   else
7   {
8
9   }
```

L'`<expression booléenne>` peut être remplacée par une simple évaluation entière : toute valeur $\neq 0$ est équivalente à `true`.

```
1   if(true) ...
2   if(x > 3) ...
3   if(y != z) ...
4   if(!a && ((i > j) || p)) ...
5   if(3) ...
```

Instruction switch

Éléments du langage

Pour un branchement conditionnel multiple :

```
1   int note; // on suppose note un entier entre 0 et 20
2
3   switch(note / 2)
4   {
5     case 10:
6     case 9:
7     case 8:
8       cout << "TB" << endl;
9       break; // l'instruction 'break' fait sortir du switch ...
10
11    case 7:
12      cout << "B" << endl;
13      break; // ... elle évite de traiter les cas suivants
14
15    case 6:
16      cout << "AB" << endl;
17      break;
18
19    case 5:
20      cout << "P" << endl;
21      break;
22
23    default :
24      cout << "Ajourné" << endl;
25  }
```

Boucles

Éléments du langage

Trois formes de boucles :

```
1   while( <expression booléenne> )
2   {
3     ... // instruction exécutée si condition validée
4         // puis répétée si condition à nouveau valide
5   }
6
7
8   do
9   {
10    ... // exécution de l'instruction, puis éventuelle répétition
11  } while( <expression booléenne> );
12
13
14  for(int i = 0 ; i < 10 ; i++) // var. d'incrémentation locale
15  {
16    ... // nombre d'exécutions prédéfini
17  }
```

Section 4

Fonctions

Déclaration

Fonctions

Une **fonction** se caractérise par :

1. son nom
2. le type de sa valeur de retour (peut être `void`)
3. ses entrées (*paramètres*)
4. l'instruction-bloc qui effectue le travail (*corps* de la fonction)

Les éléments (1)–(3) sont décrits dans la *déclaration* de la fonction.
L'élément (4) figure dans sa *définition*.

```
1    // Déclaration d'une fonction de distance :
2    float dist(float x, float y);
3
4    // Déclaration d'une procédure d'affichage :
5    void print_hello(); // ne retournera pas de valeur
```

Exemple

Fonctions

Le programme suivant affiche la valeur correspondant à $\sqrt{x^2 + y^2}$.

```
1   #include <cstdlib>
2   #include <iostream>
3   #include <math.h> // nécessaire pour le calcul de sqrt, pow...
4
5   float dist(float x, float y) // déclaration et définition
6   {
7     return sqrt(pow(x,2) + pow(y,2));
8   }
9
10  int main()
11  {
12    float pos_x = 2., pos_y = 6.;
13    std::cout << dist(pos_x, pos_y) << std::endl;
14    return EXIT_SUCCESS;
15  }
16
17  // .
```

Exemple (permutation des blocs)

Fonctions

Le programme suivant affiche la valeur correspondant à $\sqrt{x^2 + y^2}$.

```
1   #include <cstdlib>
2   #include <iostream>
3   #include <math.h>
4
5   float dist(float x, float y); // déclaration de la fonction
6
7   int main()
8   {
9     float pos_x = 2., pos_y = 6.;
10    std::cout << dist(pos_x, pos_y) << std::endl;
11    return EXIT_SUCCESS;
12  }
13
14  float dist(float x, float y) // définition de la fonction
15  {
16    return sqrt(pow(x,2) + pow(y,2));
17  }
```

Fonctions procédures

Fonctions

Une *procédure* ne retourne pas de valeurs.
On la déclare de type `void` :

```
1  void write_sqrd_value(float x)
2  {
3    cout << "value: " << pow(x, 2) << endl;
4  }
```

Lorsque plusieurs paramètres sont à retourner,
on fait **un passage par référence**.

```
1  // entrées : x, y ; sorties : rho, theta
2  void polar(float x, float y, float &rho, float &theta)
3  {
4    rho = sqrt(pow(x,2)+pow(y,2));
5    theta = atan2(y, x);
6  }
```

Valeurs, références et pointeurs

Fonctions

- **Passage par valeur** :

```
1    void write_sqrd_value(float x);
2    // x est en lecture seule
```

- **Passage par référence** :

```
1    void polar(float x, float y, float &rho, float &theta);
2    // x et y en lecture, rho et theta en lecture/écriture
```

- **Passage par pointeur** :

```
1    void display_name(string *name);
2    // valeur accessible via l'adresse de la variable name
3    // nous reverrons les pointeurs plus tard dans ce cours
```

Les références pour la modification de valeurs

Fonctions

Pour **modifier la valeur** d'un paramètre dans une fonction, on peut passer ce paramètre par référence.

Une référence sur une variable est un synonyme de cette variable, c'est-à-dire une autre manière de désigner le même emplacement de la mémoire.

On utilise le symbole & pour déclarer une référence.

```
1   void polar(float x, float y, float &rho, float &theta);
2   // x et y en lecture, rho et theta en écriture
```

Section 5

Compléments

Expression conditionnelle

Compléments

Forme concaténée de structure conditionnelle et d'affectation :
`<expression0> ? <expression1> : <expression2>`

Fonctionnement : `<expression0>` est d'abord évaluée.

- si elle est vraie, `<expression1>` est évaluée et donne sa valeur à l'expression conditionnelle
- sinon, `<expression2>` est évaluée et donne sa valeur à `<expression0>`

Exemple :

```
1   int max(int a, int b) // retourne le plus grand des 2 entiers
2   {
3     int c = a > b ? a : b;
4     return c;
5     // ou plus directement : return a > b ? a : b;
6   }
```

Chaînes de caractères

Compléments

Les chaînes de caractères n'apparaissent pas nativement en C++. Il faut utiliser la classe `string`. Exemple :

```
1   #include <string>
2
3   string name; // une chaîne de caractères vide
4   string color = "#3D93C1"; // une couleur en convention HTML
5   string method = "euler";
6
7   // Une variable string est un objet :
8   int size = method.size(); // size == 5
9   char c = method[1]; // c == 'u'
10
11  // Concaténations
12  string b = "robotique";
13  string c = "mobile";
14  string a = b + " " + c; // a == "robotique mobile"
```

Fonctions mathématiques utiles

Compléments

Fonctions de la bibliothèque standard.

Dans `math.h` :

- `floor` (resp. `ceil`) : partie entière par défaut (resp. par excès)
- `fabs` : valeur absolue
- `sqrt` : $\sqrt{}$
- `pow` : puissance (`pow(x,y)` renvoie x^y)
- `exp`, `log`, `log10`
- `cos`, `sin`, `atan`, `cosh`, *etc.*

Nombres pseudo aléatoires

Compléments

En incluant `stdlib.h` et `time.h` :

1. Pour obtenir des **tirages différents** à chaque exécution, il faut initialiser le générateur de nombres aléatoires (graine).

```
1       srand(time(NULL));
```

2. Puis chaque appel de `rand()` donnera un entier aléatoire entre 0 et `RAND_MAX` (constante définie dans `stdlib.h`)

```
1       int n = rand() % 50; // nombre aléatoire entre 0 et 49
```