Introduction au C++
GRO-300

François Ferland

Été 2018

1 Introduction

Jusqu'à maintenant dans le cadre du programme de génie robotique, vous n'avez utilisé que le langage de programmation C. Or, la suite de votre formation impliquera le langage C++, notamment lors de deux APPs pendant la session 3.
Vous avez probablement déjà vu l'expression «C/C++» pour désigner une certaine classe de langages de programmation. Or, si les langages C et C++ sont effectivement apparentés, ils demeurent tout de même différents. En effet, on croit souvent à tort que le langage C++ n'est qu'une extension du langage C, alors que ce ne sont pas tous les programmes C qui sont également des programmes C++ valides. Une extension simple aurait permis de conserver cette propriété. C'est le cas du langage Objective-C, fortement utilisé par Apple : tout programme C est aussi un programme Objective-C valide.
Le thème central de l'APP lié à GRO-300 est la programmation concurrente. L'apprentissage se fera à l'aide des fonctionnalités de la librairie standard du C++ (version 2011 et plus). Ces fonctionnalités reposent sur certains concepts propres au C++. Ainsi, même si vous reconnaîtrez la majorité de la syntaxe (déclaration de fonctions, contrôle de flux avec `if/else`, boucles `for/while`, etc.), d'autres concepts comme l'initialisation d'objets vous seront moins familiers.
Ce document est donc une courte introduction au langage C++. Son objectif n'est pas de vous apprendre un des multiples paradigmes offerts par le C++, comme la programmation orientée objets ou les techniques de meta-programmation. Il s'agit plutôt de vous aider dans votre compréhension des exemples dans la référence à lire pendant l'APP, ainsi que le code fourni comme point de départ pour la résolution du problème. Ce code a été développé de façon à être le plus près possible du «style C» impératif auquel vous êtes déjà familiers. Les nouveaux concepts sont présentés sous la forme d'un exemple de code qui ressemblera fortement à celui fourni lors de l'APP et où chaque élément sera expliqué et détaillé. Il est également à noter que votre connaissance en C++ ne sera pas évaluée à ce moment-ci de la session. Vous aurez l'occasion d'approfondir vos connaissances du C++ lors d'un autre APP sur la programmation orientée objets.

2 Premier exemple : espaces de noms et objets

L'exemple de code suivant est un programme complet en C++ qui démarre un fil d'exécution (*thread*) en parallèle au fil d'exécution principal. Il peut être compilé de cette manière sous Linux ou macOS :

```
g++ -o exemple1 exemple1.cpp
```

```
1  // Fichier: exemple1.cpp
2
3  #include <thread>
4  #include <cstdio>
5
6  void func()
7  {
8      while (true) {                                  // Boucle infinie.
9          printf("Bonjour␣du␣fil␣secondaire␣!\n");
10     }
11 }
12
13 int main(int argc, char** argv)
14 {
15     std::thread fil(func);                          // Création et démarrage
16                                                     // de l'objet "fil".
17
18     while (true) {                                  // Boucle infinie.
19         printf("Bonjour␣du␣fil␣principal!\n");
20     };
21
22     return 0;
23 }
```

L'exemple est très simple, mais permet d'illustrer deux nouveaux concepts à la ligne 15 : les espaces de noms (*namespaces*) et l'initialisation d'un objet sur la pile (*stack*).

2.1 Espace de noms

Un *namespace* est un outil pour regrouper des définitions en C++ et éviter de «polluer» l'espace global, qui est accessible en tout temps. Par exemple, tous les éléments de la librairie standard sont définis dans `std`. Pour faire référence à un membre d'un espace de noms, on utilise le séparateur «`::`». Ainsi, à la ligne 15, l'expression `std::thread` fait référence au type `thread` de l'espace de nom `std`.
Regrouper les définitions par espaces de noms permet d'éviter les collisions de noms dans l'espace global. Par exemple, nous définissons une fonction `func()` à partir de la ligne 6. Si un des fichiers inclus (`thread` ou `cstdio`) déclarait déjà une fonction du même nom, ce serait une erreur. En effet, ce serait une tentative de redéfinition de la fonction. Regrouper des éléments par espaces de noms est une bonne façon d'organiser des programmes C++ complexes.

2.2 Déclaration et initialisation d'objets

À la ligne 16, nous avons un exemple type de déclaration et initialisation de variable objet. Vous reconnaîtrez un exemple similaire dans vos lectures lors de l'APP. Ainsi, la variable `fil` représentera un objet de type (ou classe) `std::thread`. La syntaxe est similaire à si on avait déclaré une variable `int` en C, sauf pour l'ajout du paramètre d'initialisation «(func)».

Une des particularités importantes du C++ est de pouvoir définir de nouveaux types de données et la façon dont ils sont construits et détruits. Lorsqu'on initialise un objet en C++, on déclare une variable, ce qui a pour effet de réserver une quantité de mémoire suffisante sur la pile pour contenir l'objet. Tout de suite après, on appelle une fonction de construction qui reçoit les paramètres d'initialisation afin de préparer l'état initial de l'objet. Ces deux étapes sont automatiques et indissociables lorsqu'on utilise la syntaxe de la ligne 16. Ici, l'objet de type `thread` reçoit le nom de la fonction à démarrer dans le nouveau fil d'exécution qui sera créé. À sa destruction, c'est-à-dire lorsque la variable `fil` sera libérée à la fin de l'exécution de la fonction `main`, une fonction de destruction sera appelée pour terminer l'exécution du fil [1].
Cette approche, où un objet est initialisé dès l'acquisition de la mémoire se nomme *Resource Acquisition Is Initialisation* (RAII), et est fréquemnent utilisée en C++. Vous verrez plus en détails comment définir des classes d'objets dans un autre APP.

2.3 Inclusion de fonctions de la librairie standard C

À la ligne 4, vous avez peut-être remarqué que nous faisons l'inclusion du fichier `cstdio` plutôt que `stdio.h` pour utiliser `printf`. En C++, la norme veut que les entêtes de la librairie standard C++ ne portent pas d'extension «`.h`», et que les en-têtes importés du C soient préfixés par la lettre C. C'est une façon de distinguer les fonctionnalités du langage de celles fournies par le système ou votre propre programme. Il est donc toujours recommandé d'utiliser une extension pour vos propres fichiers, par exemple «`.cpp`» et «`.hpp`».

1. Attention ! Si le fil secondaire n'a pas terminé son exécution, il peut s'agir d'une erreur de le détruire. Vous apprendrez comment éviter cela pendant l'APP.

3 Deuxième exemple : Déclaration d'espaces de noms et fonctions membres

Dans ce deuxième exemple, nous allons voir comment on peut définir des espaces de noms, des classes et utiliser des fonctions membres d'un objet. Vous n'aurez pas à comprendre comment une classe est créée, mais l'exemple vous aidera à mieux comprendre ceux de vos autres lectures de l'APP.

```
1  #include <cstdio>
2  #include <thread>
3
4  namespace gro
5  {
6      class Exemple                                // Définition de classe.
7      {
8      private:
9          int membre_;
10
11     public:
12         Exemple(int i)                           // Constructeur.
13         {
14             membre_ = i;
15         }
16
17         void print() const
18         {
19             printf("Exemple␣%i\n", membre_);
20         }
21
22         void operator()() const
23         {
24             printf("Objet-fonction␣%i\n",
25                    membre_);
26             while (true) {};                     // Boucle infinie.
27         }
28
29     };
30 }
31
32 int main(int argc, char** argv)
33 {
34     gro::Exemple ex1(1);
35     gro::Exemple ex2(2);
36
37     ex1.print();                                 // Affiche "Exemple 1".
38     ex2.print();                                 // Affiche "Exemple 2".
39
40     // ex1.membre_ = 3;                          // Erreur: membre_ est privé.
41
42     gro::Exemple ex3(3);
43     std::thread fil(ex3);                        // Démarre un nouveau fil,
44                                                  // affiche "Objet-fonction 3".
45
46     while (true) {};                             // Boucle infinie.
47
48     return 0;
49 }
```

3.1 Définition d'un espace de nom et d'une classe

À la ligne 4, nous voyons comment le mot-clé `namespace` peut être utilisé pour déclarer un espace de nom, ici nommé `gro`. Ainsi, toutes les déclarations et définitions contenues dans

le bloc délimité par {...} qui suit appartiendra à ce nouvel espace de nom. C'est le cas de la classe `Exemple`. Les classes se définissent de façon similaire aux structures en C [2]. Par contre, on peut y définir des sections privées (lignes 8 à 10) et publiques (11 à 29) pour contrôler l'accès aux membres. Par exemple, l'accès en ligne 40 est erroné, car l'expression tente d'accéder à un membre privé depuis l'extérieur de la classe.

3.2 Définition de fonctions membre

À la ligne 17, on retrouve une définition de fonction membre. Lorsqu'une fonction membre d'une classe est appelée, c'est toujours en référence à une instance de classe (un objet). Le contenu de la fonction membre a accès à tous les membres (publics ou non) associés à cette instance. Le mot-clé `const` indique que la fonction ne modifiera pas les données de l'instance.
À la ligne 22, on retrouve un autre type de fonction membre : un opérateur. En effet, il est possible de redéfinir le comportement des opérateurs tels que «`+`» ou «`*`», mais aussi les parenthèses d'appels de fonction. En définissant un opérateur ainsi, une instance de la classe Exemple pourra être appelé comme n'importe quelle fonction. On appelle ce type d'instance un objet-fonction (ou *functors*).

3.3 Appel de fonctions membre

Pour appeler une fonction membre, il faut nécessairement avoir un objet prédéfini. C'est le cas des définitions aux lignes 34 et 35. Dans ces deux cas, le constructeur de la classe Exemple prend en paramètre un `int`, qui sera affecté à la variable `membre_` séparément pour chacune des instances.
L'appel se fait en utilisant l'opérateur infixe «`.`» entre la variable représentant l'objet et l'appel de la fonction, comme aux lignes 37 et 38. Par le résultat à l'écran, on voit bien que chaque instance a une existence propre, séparées l'une de l'autre.
À la ligne 43, on construit à nouveau un fil d'exécution, mais cette fois-ci en passant un objet en paramètre plutôt qu'un nom de fonction. En effet, comme l'objet `fil` est de classe `Exemple` et possède donc une définition de l'opérateur «`()`», l'objet de classe `thread` pourra se servir de l'objet `fil` comme point de démarrage.

2. D'ailleurs, les mots-clés `class` et `struct` sont pratiquement interchangeables en C++. La seule différence est qu'une structure a un accès publique par défaut pour ses membres au lieu de privé dans le cas des classes.