À la découverte du C++

PRUVOST Benjamin [1] REPORT Tanguy [2]

1. benjamin.pruvost@etu.univ-littoral.fr
2. Tanguy.Report@etu.univ-littoral.fr

Table des matières

Chapitre 1

Introduction

1.1 L'histoire du C++

FIGURE 1.1 – Bjarne Stroustrup (2013)

Le langage C++ a été conçu au début des années 1980 par Bjarne Stroustrup (figure 1.1), un ingénieur d'origine danoise, alors qu'il travaillait au sein des laboratoires Bell Labs, un centre de recherche réputé pour ses innovations. Stroustrup souhaitait améliorer le langage C en y intégrant des fonctionnalités de programmation orientée objet, qui permettent de structurer le code de manière plus modulaire et intuitive, notamment grâce à des concepts tels que les classes et les objets. L'objectif principal était de combiner la puissance, la flexibilité et l'efficacité du C avec une organisation et une maintenabilité accrues, afin de faciliter l'écriture et la gestion de programmes complexes. Aujourd'hui, le C++ demeure l'un des langages de programmation les plus populaires et polyvalents, largement utilisé dans le développement de logiciels, de jeux vidéo, d'applications haute performance, ainsi que dans les systèmes embarqués et d'autres domaines technologiques avancés.

1.2 Différences entre C et C++

Le C et le C++ sont des langages de programmation étroitement liés, mais ils diffèrent par leurs caractéristiques et leurs usages. Le C est un langage procédural qui se concentre sur les fonctions et les structures de données, idéal pour les systèmes bas-niveau comme les systèmes d'exploitation. En revanche, le C++ étend le C en introduisant la programmation orientée objet, avec des concepts comme les classes, l'héritage et le polymorphisme, ce qui le rend plus adapté à des projets complexes nécessitant une organisation modulaire. De plus, le C++ offre des fonctionnalités modernes comme les templates, les exceptions et la gestion automatique des ressources, absentes en C. Malgré leurs différences, *le C++ reste compatible avec le C*, permettant d'utiliser du code C dans des projets C++.

1.3 Installation de l'environnement de développement

Afin d'écrire, compiler et exécuter du code C++ sur notre machine il est nécessaire d'installer deux outils essentiels : un éditeur de texte et un compilateur. Nous ne verrons pas l'installation de l'éditeur de texte dans ce cours il vous suffit de vous rendre sur le site de votre éditeur de texte de préférence, de télécharger le wizard d'installation correspondant à votre système et de suivre les indications pour pouvoir l'utiliser sur votre machine. Ces éditeurs incluent (mais ne se limitent pas à) :

— Visual Studio Code
— Notepad++
— Sublime Text

Concentrons nous donc sur l'installation du compilateur. Pour les utilisateurs de Mac, MacOs possède déjà un compilateur intégré, nous ne nous soucierons donc que des installations sur Windows et Linux.

1.3.1 Windows

1. Tout d'abord, rendez vous sur le site de MSYS2 et téléchargez l'installateur.
2. Lancez l'installateur et suivez les étapes.
3. Dans l'installateur, choisissez votre dossier d'installation, gardez le en mémoire pour plus tard. Dans la plus part des cas le dossier de base est acceptable.
4. Une fois l'installation terminée assurez vous que le choix *Run MSYS2 now* est coché puis sélectionnez *Finish*. Cela vous ouvrira un terminal MSYS2.
5. Dans ce terminal, installez les outils de compilation de MinGW en lançant la commande suivante :

```
pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
```

6. Acceptez l'installation des packets en pressant la touche "Entrée".
7. Entrez 'Y' quand le programme vous demande si vous voulez poursuivre l'installation.
8. Ajoutez le chemin du dossier *bin* de votre installation de MinGW au PATH.
   (a) Dans la barre de recherche Windows, cherchez *Modifier les variables d'environnement système*.
   (b) Dans vos variables d'utilisateur, sélectionnez la variable *Path* et cliquez sur *Modifier*.

(c) Sélectionnez *Nouveau* et ajoutez le chemin du dossier bin. Si vous n'avez pas changé l'adresse de celui proposé lors de l'installation le chemin devrait être le suivant : *C :\msys64\ucrt64\bin*.

(d) Sélectionnez *OK* deux fois.

9. Pour vérifier votre installation ouvrez un nouvel invite de commande puis tapez la commande suivante :

```
g++ --version
```

Si vous n'obtenez pas d'erreur alors félicitations ! Vous pouvez désormais compiler du code C++ !

1.3.2 Linux

Dans un premier temps, mettez à jour votre installateur en exécutant la commande suivante :

```
sudo apt update
```

Ensuite, installez l'outil de compilation *g++* grâce à cette autre commande :

```
sudo apt install g++
```

1.3.3 MacOS

Il est probable que le compilateur *g++* soit déjà installé sur votre système afin d'en être sûr vous pouvez exécuter cette commande et vérifier qu'elle ne vous renvois pas d'erreur :

```
g++ --version
```

Dans le cas contraire vous pouvez l'installer grâce à la commande suivante :

```
xcode-select --install
```

Chapitre 2

Structure d'un programme

Il est important de respecter la structure du code en C++ afin d'assurer sa lisibilité, sa maintenabilité et son efficacité. Une organisation claire et cohérente du programme facilite son développement et sa compréhension, tant pour le programmeur que pour les membres de l'équipe.

2.1 Les Directives de Préprocesseur

Un programme en C++ commence généralement par des directives de préprocesseur, qui sont des instructions pré-traitées avant la phase de compilation. Elles permettent d'inclure des bibliothèques et de définir des constantes ou des macros. Ces directives commencent par *le symbole* #.Voici ci-dessous un exemple :

```
//Implementation de la bibliotheque iostream
#include <iostream>
```

2.1.1 Les implémentations de bibliothèques

Les bibliothèques en C++ jouent un rôle essentiel dans le développement d'applications, car elles offrent des fonctions et des classes pré-définies qui simplifient grandement la gestion des tâches courantes. Une des bibliothèques les plus utilisées est iostream, qui permet la gestion des entrées et sorties, notamment pour la lecture et l'écriture sur la console. L'implémentation de iostream repose sur des flux d'entrée/sortie, tels que std : :cin pour l'entrée et std : :cout pour la sortie mais pas seulement [1].

```
//Implementation de la bibliotheque iostream
#include <iostream>

int main(){
    std::cout << "Hello␣World␣!";
    return 1;
}
```

Ce code permet d'afficher "Hello World" dans la console, ce qui est impossible sans la bibliothèque iostream

1. Documentation de iostream :https://cplusplus.com/reference/iostream/

Il y a bien évidemment plein d'autre bibliothèques utiles, comme par exemple `<cmaths>`,qui fournit un ensemble de fonctions mathématiques standards pour effectuer des calculs numériques.

Par exemple si nous voulons calculer en C++ $\sqrt{\cos(\frac{1}{3})}$, nous avons maintenant accès aux fonctions `sqrt()` et `cos()` :

```
#include <iostream>
#include <cmath>

int main(){
    std::cout << sqrt(cos(1/3));
    return 0;
}
```

2.1.2 Les Espaces de Noms

Les espaces de noms (namespaces) permettent de regrouper des éléments tels que des fonctions, des classes et des variables pour éviter les conflits de noms. Le namespace standard (std) est couramment utilisé :

```
using namespace std;
```

Cette directive permet d'accéder directement aux fonctions et objets de la bibliothèque standard, comme cout ou cin.

```
#include <iostream>
using namespace std;

int main(){
    cout << "Hello␣World␣!";
    return 1;
}
```

On remarquera maintenant qu'il n'est plus nécessaire d'écrire `std::` devant le cout, qu'il est fastidieux d'écrire quand on appelle ces fonctions un grand nombre de fois.

2.1.3 Les macros

Les macros en C++ sont définies avec `#define` et permettent de remplacer des parties du code avant la compilation. Elles sont souvent utilisées pour définir des constantes ou des expressions réutilisables.Exemple : `#define PI 3.14159`

2.2 La fonction main()

La fonction main est le point d'entrée de tout programme en C++. C'est là que l'exécution commence.Il doit être placer à la fin du code, tout code n'étant pas contenu dedans n'est pas exécuté sauf exception [2]. Elle retourne généralement un entier pour indiquer l'état de fin du programme :

2. Les constructeurs de classes par exemple

```
int main(){
        //Corps du programme
        return 0;
}
```

La valeur de retour 0 signale une exécution réussie, tandis qu'une valeur non nulle indique une erreur.

La fonction main peut également accepter des arguments pour traiter les entrées de la ligne de commande :

```
int main(int argc, char* argv[]) {
        // argc : Nombre d'arguments
        // argv : Tableau contenant les arguments
}
```

Et dont voici un exemple très simple d'utilisation, il faut d'abord taper la ligne de commande ci dessous en remplacer par les noms correspondant à votre fichier et vos arguments :

```
./monProgramme arg1 arg2
```

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    //affichage du nombre d'arguments
    cout << "Nombre d'arguments : " << argc << endl;

    // Affichage des arguments
    for (int i = 0; i < argc; ++i) {
        cout << "Argument " << i << " : " << argv[i] << endl;
    }

    return 0;
}
```

2.3 Fonctions et commentaires

Puis finalement pour bien organiser le code, celui-ci va être découper en fonction(détaillées au chapitre 5 p.11)

Mais il faut aussi y insérer des commentaires, notamment pour expliquer les buts des fonctions, mais également pour expliquer des bouts de code qui pourrait être compliqué à comprendre.Cela est nécessaire pour travailler en équipe sur des fichiers de code communs.

```
// Ceci est un commentaire sur une ligne

/*
  Voici un commentaire
  sur plusieurs lignes differentes !
*/
```

Voici finalement à quoi devrait ressembler votre code une fois bien structuré.

```
#include <iostream>
//Ici d'autre inclusion de bibliotheque si necessaire
using namespace std;

void fonction1(){
    //explication fonction1
}

void fonction2(){
    //explication fonction2
}
int main(){
    fonction1();
    fonction2();
    return 0;
}
```

Chapitre 3

Types de données et variables

types primitifs (int, float, char, bool) <- tableau avec nom, encodage et range des valeurs. opérateurs simples

Chapitre 4

Instructions de contrôle

opérateurs de contrôle (==, <=, >=, !=), if/else, while, for, etc..

Chapitre 5

Fonctions

déclaration, passage de paramètres, retour de valeurs, (surcharge des fonctions <- optionnel)

Chapitre 6

Pointeurs

opérations de base sur les pointeurs, explication en utilisant la mémoire, rapport avec les tableaux

Chapitre 7

Structures

introduction à l'OOP