Chapitre 1 : Présentation du langage C++

Objectifs pédagogiques :

- Comprendre le contexte et l'évolution du langage C++.
- Maîtriser la chaîne de compilation spécifique au C++.
- Être capable de mettre en place un environnement de développement (IDE, compilateur).
- Écrire, compiler et exécuter un premier programme simple.

1. Introduction et Contexte

Dans ce cours de programmation en C++, vous serez amenés à concevoir des systèmes de contrôle-commande, à simuler des dynamiques complexes, ou encore à implémenter des algorithmes de traitement du signal en temps réel.

1.1. Pourquoi le C++ en Automatique ?

Le choix du C++ pour une licence d'automatique n'est pas anodin. Il se positionne comme un langage de prédilection dans l'industrie pour plusieurs raisons :

1. **Performance et Contrôle :** Le C++ offre des performances proches du langage C (le langage historique des microcontrôleurs) tout en ajoutant des fonctionnalités de haut niveau. Il permet une gestion fine de la mémoire, indispensable sur des systèmes embarqués aux ressources limitées.
2. **Programmation Orientée Objet (POO) :** Cette approche permet de modéliser le monde réel de manière intuitive. En automatique, on peut facilement représenter un **moteur**, un **capteur**, un **régulateur PID** ou un **filtre** comme des objets avec leurs propres données (attributs : vitesse, température, coefficients) et comportements (méthodes : `calculer_commande()`, `lire_valeur()`).
3. **Abstraction sans surcoût :** Le C++ permet de créer des bibliothèques de calcul numérique très efficaces (ex: Eigen pour l'algèbre linéaire) sans pénaliser les performances. C'est ce qu'on appelle le principe *"zero-overhead abstraction"*.
4. **Écosystème industriel :** De nombreux logiciels de simulation (comme certaines parties de Simulink), de robotique (ROS - Robot Operating System), et de systèmes temps réel sont écrits en C++.

1.2. Bref Historique

- **1979-1983 :** Bjarne Stroustrup, chez Bell Labs, commence à travailler sur "C with Classes". L'objectif est d'ajouter au langage C des concepts de programmation orientée objet pour gérer des projets logiciels plus complexes. Le langage est renommé C++ en 1983 (l'opérateur `++` étant l'incrémentation en C, signifiant "une évolution du C").
- **1998 :** Première normalisation ISO (C++98). Le langage se stabilise.
- **2011 :** Révolution avec C++11. Le langage est modernisé en profondeur (inférence de type `auto`, boucle `for` basée sur les plages, nouveaux pointeurs intelligents, etc.). C'est un tournant majeur.
- **2014, 2017, 2020 :** Depuis, le langage évolue avec un cycle de révision régulier (C++14, C++17, C++20), apportant de nouvelles fonctionnalités qui rendent le code plus sûr, plus expressif et plus performant.

Note pour le cours : Nous nous concentrerons sur un C++ moderne (C++11 et ultérieur) tout en gardant un œil sur les aspects fondamentaux nécessaires à la compréhension des systèmes.

2. Environnement de développement

Pour programmer en C++, nous avons besoin d'un éditeur (pour écrire le code) et d'un compilateur (pour le transformer en programme exécutable).

2.1. Le Compilateur

Le compilateur est le logiciel le plus important. Il lit le code source (fichiers .cpp, .h) et le traduit en langage machine.

Parmi les compilateurs les plus connus, on cite :

- **GCC (GNU Compiler Collection) :** Le compilateur standard sous Linux. La commande pour le C++ est g++.
- **Clang :** Un compilateur réputé pour ses messages d'erreur clairs et ses performances. Très utilisé sur macOS et de plus en plus sous Linux.
- **MSVC (Microsoft Visual C++) :** Le compilateur fourni avec Visual Studio sous Windows.

2.2. Les Editeurs / IDE (Environnements de Développement Intégrés)

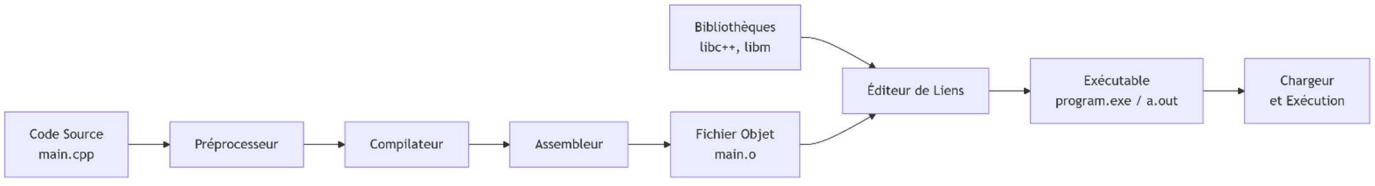
Un IDE regroupe un éditeur de code, un compilateur, un débogueur et d'autres outils. Pour ce cours, nous utiliserons principalement **Visual Studio Code**, car il est léger, multi-plateforme et très flexible.

Parmi les IDE les plus connus, on cite :

- **Visual Studio Code (VS Code) :** Éditeur léger et extrêmement personnalisable. Avec les extensions adéquates (C/C++ de Microsoft, Code Runner), il devient un IDE très complet.
- **Code::Blocks :** IDE simple, open-source et multi-plateforme, souvent utilisé dans l'enseignement.
- **Qt Creator :** Très bon IDE, originellement pour les interfaces graphiques Qt, mais parfait pour le C++ généraliste.
- **Visual Studio (Community) :** L'IDE le plus complet sous Windows, mais plus lourd.
- **CLion :** IDE payant de JetBrains, excellent mais nécessite une licence étudiante.

2.3. La Chaîne de Compilation

Comprendre ce qui se passe entre l'écriture du code et l'exécution du programme est crucial, surtout quand on abordera des projets plus complexes.



1. **Édition du code :** Vous écrivez votre programme dans un ou plusieurs fichiers .cpp (implémentation) et .h ou .hpp (en-têtes).

2. **Prétraitement :** Le préprocesseur exécute les directives commençant par # (comme #include <iostream>). Il va littéralement copier le contenu des fichiers inclus dans le fichier source.
3. **Compilation :** Le compilateur traduit le code prétraité en langage assembleur, puis en code machine, mais pas complètement. Il génère un fichier objet (.o ou .obj). Ce fichier contient le code, mais avec des "trous" : les appels aux fonctions des bibliothèques externes ne sont pas encore résolus.
4. **Édition des liens (Linking) :** L'éditeur de liens (linker) prend votre ou vos fichiers objets, et les combine avec les bibliothèques nécessaires (comme la bibliothèque standard C++). Il résout les "trous" pour produire un fichier exécutable unique.
5. **Exécution :** Le système d'exploitation charge votre programme en mémoire et commence son exécution à partir du point d'entrée (la fonction main()).

3. Premier Programme : "Bonjour le monde"

```
// premier_pas.cpp : Notre premier programme en C++

// 1. Directives de préprocesseur
#include <iostream>  // Pour les entrées/sorties standard (cout)

// 2. Utilisation de l'espace de noms (pour simplifier l'écriture)
using namespace std;

// 3. Fonction principale - Point d'entrée du programme
int main() {
    cout << " Hello world! " << endl;

    // Fin du programme : retourne 0 pour indiquer que tout s'est bien passé
    return 0;
}
```

3.1. Compilation et Exécution

Sous Windows (Invite de commandes avec Visual Studio Code ou MinGW) :

1. Ouvrez l'invite de commandes ou le terminal intégré de VS Code.
2. Placez-vous dans le bon répertoire.
3. Compilez (avec MinGW) :

```
g++ premier_pas.cpp -o premier_pas.exe
```

4. Exécutez :

```
premier_pas.exe
```

3.2. Analyse du Code

- `#include <iostream>` : Indique au préprocesseur d'inclure le fichier en-tête de la bibliothèque d'entrées/sorties standard, nécessaire pour utiliser `cout` et `endl`.
- `using namespace std;` : Cela signifie que l'on va utiliser l'espace de noms `std` (standard). Sans cela, il faudrait écrire `std::cout` au lieu de `cout`. Nous verrons plus tard les implications de cette ligne.
- `int main() { ... }` : La fonction `main` est le point de départ de tout programme C++. Elle doit retourner un entier (`int`).
- `<<` : L'opérateur d'insertion. Il envoie ce qui est à sa droite vers `cout` (la console).
- `endl` : "end line", ajoute un retour à la ligne et vide le tampon de sortie.
- `return 0;` : Signifie que le programme s'est terminé avec succès.

4. Introduction au Débogage

Lors du débogage, les erreurs se divisent en deux catégories :

- Erreurs de compilation (syntaxe) : Le compilateur ne comprend pas votre code. Il vous donnera un message d'erreur avec un numéro de ligne. Par exemple, oublier un ; en fin de ligne.
- Erreurs d'exécution (logique) : Le programme compile, mais ne fait pas ce que vous voulez (ex: il plante, ou donne un résultat faux).

4.1. Utilisation basique d'un Débogueur (GDB)

1. Un débogueur permet d'exécuter le programme pas à pas, d'inspecter les variables, et de voir où il plante.

Pour utiliser un débogueur, il faut compiler avec l'option -g pour inclure les symboles de débogage :

```
g++ -g premier_pas.cpp -o premier_pas
```

2. On peut alors lancer le débogueur gdb :

```
gdb ./premier_pas
```

- break main : Place un point d'arrêt au début de la fonction main.
- run : Lance le programme.
- next ou n : Exécute la ligne suivante.
- print tension_entree : Affiche la valeur de la variable.
- quit : Quitte le débogueur.

En pratique, ces commandes sont souvent intégrées dans les IDE comme VS Code, où l'on peut cliquer à côté d'une ligne pour mettre un point d'arrêt et lancer le débogueur avec une interface graphique.