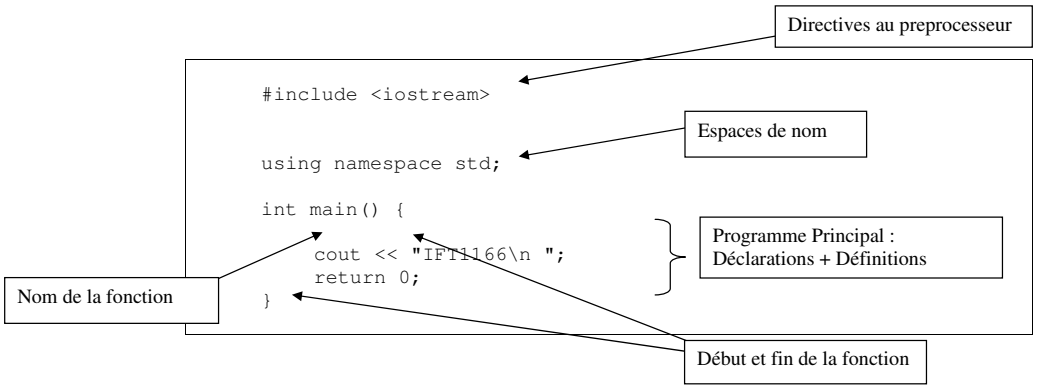
Chapitre 1

Généralités

Le langage C++ est né de l'association des langages C et Simula, au point de l'appeler au départ « C avec les classes » avant de prendre en 1983 le nom de C++. L'inventeur du langage est Bjarne Stroustrup, un informaticien de AT&T.

1. Architecture minimale d'un programme C++



1.1 Directives au processeur

- C'est une ligne de programme commençant par le caractère #.
- Elle permet de manipuler le code du programme source avant sa compilation.

- Parmi les directives, on trouve : inclusions de fichiers, substitutions, macros, compilation conditionnelle.

- La ligne « #include <iostream>» est interprétée par le préprocesseur qui recherche dans des répertoires standards le fichier dont le nom est « iostream ».
- Si le préprocesseur trouve ce fichier, il l'« inclut » en lui faisant subir le même traitement que le fichier initial (traitement des lignes commençant par #, recherche des macros etc.).

- Ce simple programme nécessite l'inclusion du fichier externe « iostream » contenant les méthodes nécessaires pour manipuler les entrées et les sorties.

1.2 Espace de noms

- La bibliothèque standard C++ est définie dans son propre espace de noms, un bloc (ou espace) portant le nom « **std** ».
- La directive « **using namespace std** » dit au compilateur que toutes les choses définies dans « std » doivent être rendues accessibles pour être directement utilisées.
- La bibliothèque « std » contient par exemple la définition de la fonction « cout » utilisée pour l'affichage en sortie.

1.3 Programme Principal

- La fonction « main » est le point d'entrée d'un programme C++.
- C'est à travers cette fonction « main » qu'un programme C++ démarre.
- La signature de la méthode « main » peut-être avec paramètres ou sans, comme suit:

```
int main()

int main(int argc, char* argv[])
```

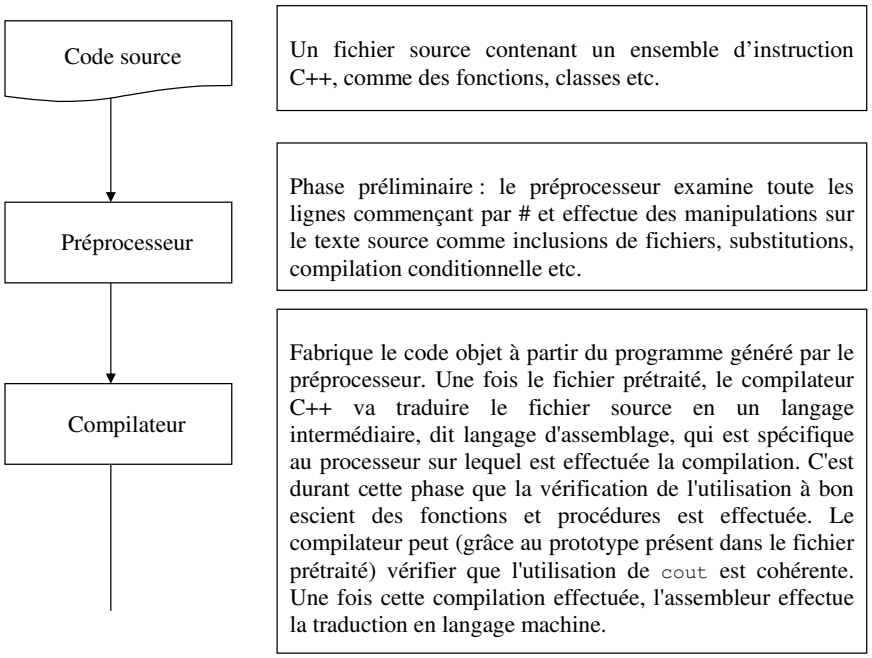
- Les paramètres « *argc* » et « *argv* » permettent de récupérer les arguments de la ligne de commande qui a lancé ce programme.
- La variable « argc » représente le nombre d'arguments, nom du programme compris.
- La variable « argv » est un tableau de chaînes de caractères contenant la liste des arguments.

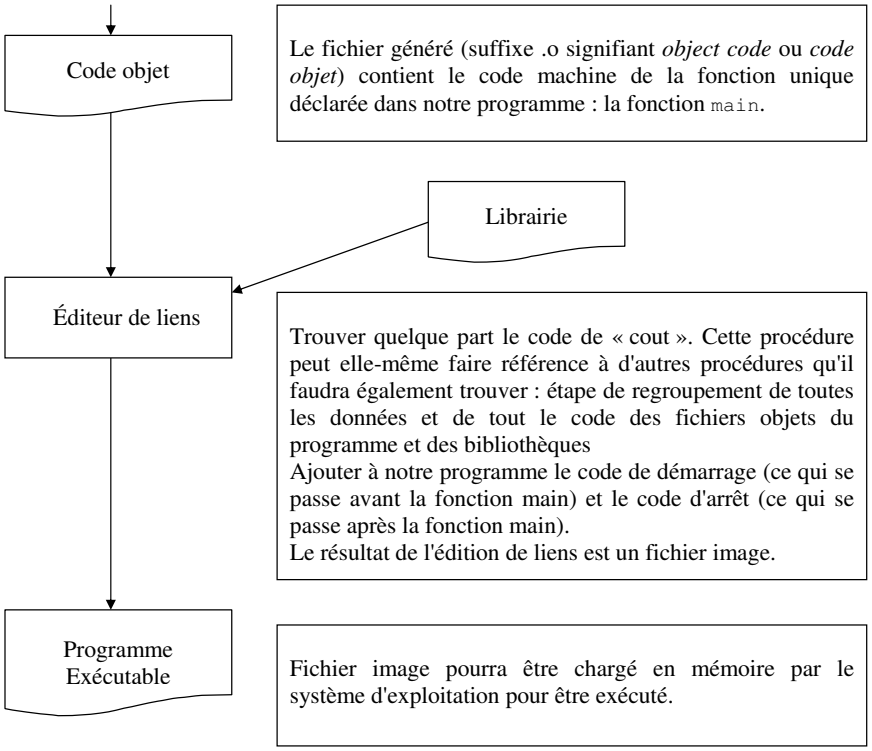
- La méthode « main » retourne une valeur entière représentant l'état de l'exécution du programme.
- La valeur entière retournée est par convention positive non nulle en cas d'erreur.

- Dans notre exemple, la valeur retournée est « 0 » pour signifier que l'exécution du programme s'est faite correctement.

2. Génération d'un programme exécutable

Les différentes étapes intervenant dans le processus d'élaboration d'un programme exécutable sont comme suit :





3. De Java/C vers C++

- Le langage C++ est une extension objet du langage C.
- Il a été développé au début des années 80 par Bjarne Stroustrup.
- Par la suite le langage Java s'est inspiré de la partie objet du C++, avec quelques différences importantes.
- Pour cette raison, aujourd'hui, on dit que le langage C++ est une sorte de mélange C et Java.

3.1 C++ versus C

- Les langages C et C++ ont une même syntaxe de base.
- Il est donc possible de passer facilement du C vers C++ (portabilité).
- Un programme écrit dans un langage C (normalisation du C : standard Ansi-C) peut-être directement compilé comme étant un programme C++.
- Ainsi donc, l'intégration de fichiers C++ et C dans un même programme est facile à faire.
- Il permet ainsi de réutiliser toutes les bibliothèques existantes.
- Lors de l'exécution d'un programme C++, **très bien écrit**, ses performances devraient être comparables à un programme C.

- Cependant, parmi les inconvénients, bien que le langage C++ soit plus strict que le langage C, il a hérité de certains choix malencontreux du langage C!
- Par exemple : *) le fait de passer par un préprocesseur. *) le manque de gestion automatique de la mémoire. *) pas de protection de la mémoire. *) avare en messages d'erreurs. *) manipulation parfois hasardeuse des pointeurs même si elle fait la joie de certains bidouilleurs !
- Cette richesse fait qu'elle peut-être utilisée à mauvais escient !

3.2 C++ versus Java

- Les langages C++ et Java se ressemblent à un certain degré.
- En effet, la syntaxe est en partie similaire, les fonctionnalités objet sont de même nature.

- Parmi les différences, on peut citer le fait que C++ est aussi un langage procédural, vu qu'il est une extension du langage C.
- Alors que Java est un langage purement orienté objet.

- Dans Java, on note la présence du « garbage collector » ou « ramasse-miettes » pour permettre une gestion automatique des ressources mémoires disponibles. En C++, cette gestion est laissée au programmeur.

- Java ne permet pas l'héritage multiple comme C++.
- Ainsi en Java, dans un schéma d'héritage, on ne peut pas écrire par exemple qu'un hydravion hérite des propriétés à la fois de « avion » et de « bateau ».

- C++ permet la redéfinition de la plupart des opérateurs.
- Il permet aussi l'utilisation de la notion de généricité (templates). Cette notion a été introduite, sous une certaine forme, tout récemment dans la version 5 de Java.

- C++ est un langage compilé alors que Java est un langage interprété.
- Un langage compilé est plus rapide lors de l'exécution qu'un langage interprété. Il a cependant l'inconvénient d'être machine dépendante.
- Ceci signifie qu'il faudra le compiler pour chaque type de système d'exploitation (Linux, Windows etc.).

- Alors que pour Java qui est un langage interprété, nécessite la présence d'une machine virtuelle pour chaque architecture.

4. Normalisation et compilateurs

- Les premiers travaux de normalisation du langage C++ ont commencé en 1989.
- Sa première normalisation n'a été validée qu'en 1998.
- Cette normalisation a permis de standardiser la base du langage ainsi que la bibliothèque C++ standard.
- La dernière normalisation du C++ remonte à 2003.
- Elle concernait la bibliothèque de modèles standard (STL) ainsi que la bibliothèque C.

- Malgré le fait que le langage C++ a été normalisé, ils existent des différences notables entre les compilateurs existants.
- Une partie de ces compilateurs n'observent pas totalement la norme définie.

- Pour ce cours, nous allons utiliser deux compilateurs :

    - **GCC** : un compilateur gratuit du domaine public le plus récent.

        - Il faudra faire attention à la version utilisée. Nous allons développer l'ensemble des exemples du cours, des exercices des séances de démonstrations ainsi que les travaux pratiques en utilisant la version du compilateur disponible dans les laboratoires d'enseignement.
        - La première séance de démonstration va être la présentation des différents outils disponibles sur des machines. Cette séance va vous permettre ainsi de voir les différents outils gravitant autour de gcc pour une utilisation simple et efficace !

    - **MicrosoftDotNET 200X** : un compilateur développé par Microsoft.

        - Il est gratuit. Par contre, l'interface graphique (Visual Studio .NET) est payante.
        - Nous allons utiliser la version disponible dans les laboratoires d'enseignement.
        - Malheureusement Microsoft a tendance à définir sa propre norme.
        - Même si cette plate forme sera introduite très tôt dans les séances de démonstration, elle ne sera utilisée de manière soutenue que pour les cours relatifs aux interfaces graphiques.

- Toutes les anciennes versions des compilateurs « gcc » ou « Microsoft », seront considérées comme désuètes !
- Nous pensons par exemple aux versions « 2.95 » de gcc, la version « 6 » de « Visual Studio » etc.

5. GCC sous toutes les formes

- La commande g++ est dérivée de la commande gcc.

- Cette commande met à jour au moment de son appel les chemins nécessaires pour accéder aux différents fichiers d'entêtes et de bibliothèques nécessaires pour une compilation correcte d'un programme écrit dans un langage C++.

Options du compilateur	Description
`g++ -v`	**-v** : pour connaître le numéro de version du compilateur.
`g++ -c toto.cpp -o toto.o`	**-c** : permet de préciser le fichier à compiler. **-o** : permet de préciser le nom du fichier objet. Par défaut, ce fichier porte le nom de : « a.out ».
`g++ toto.o -o toto.exe`	On réalise l'étape d'édition de liens. **-o** : permet de préciser le nom du fichier exécutable. Par défaut, ce fichier porte le nom de : « a.out ».
`g++ toto.cpp -o toto.o`	D'une pierre, deux coups ! Cette instruction permet de compiler et de réaliser l'édition de liens en « une seule » opération. **-o** permet de préciser le nom du fichier exécutable. Par défaut, ce fichier porte le nom de : « a.out ».
`g++ -g toto.cpp -o toto.o`	**-g** permet d'inclure les informations nécessaires pour permettre le débogage du programme.
`g++ -Wall`	**-Wall** demande au compilateur de signaler tous les endroits dans le fichier compilé qui sont des utilisations douteuses du langage C++. L'option Wall vient de Warnings ALL.
`g++ --help`	**--help** affiche l'aide en ligne en rapport avec les différentes options disponibles.
`g++ -pedantic`	**-pedantic** pour un respect strict de la spécification.