TP C++ pré-requis : 1° partie

Sommaire

Les objectifs de ce tp sont de comprendre et mettre en oeuvre la fabrication d'un programme simple. Beaucoup de conseils sont issus du livre de référence de Bjarne Stroustrup (www.programmation.stroustrup.pearson.fr).

Remarque : les TP *ont pour but d'établir ou de renforcer vos compétences pratiques. Vous pouvez penser que vous comprenez tout ce que vous lisez ou tout ce que vous a dit votre enseignant mais la répétition et la pratique sont nécessaires pour développer des compétences en programmation. Ceci est comparable au sport ou à la musique ou à tout autre métier demandant un long entraînement pour acquérir l'habileté nécessaire. Imaginez quelqu'un qui voudrait disputer une compététion dans l'un de ces domaines sans pratique régulière. Vous savez bien quel serait le résultat.*

Première partie : Le premier programme

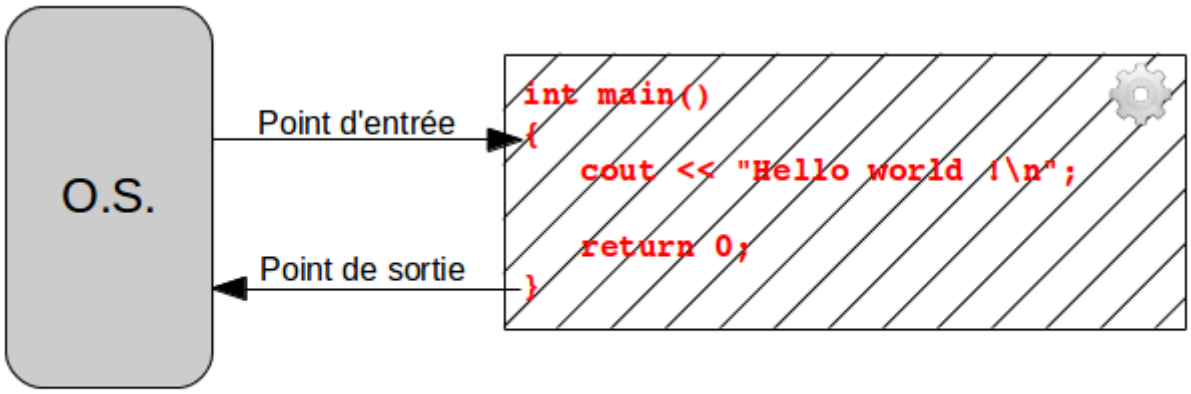
Hello world

Voici une version du premier programme que l'on étudie habituellement. Il affiche "Hello world !" à l'écran :

```
// Ce programme affiche le message "Hello world !" à l'écran

#include <iostream>

using namespace std;

int main()
{
   cout << "Hello world !\n"; // Affiche "Hello world !"

   return 0;
}
```

Hello world (version 1) en C++

Explications



Exécution d'un programme « binaire » par le système d'exploitation

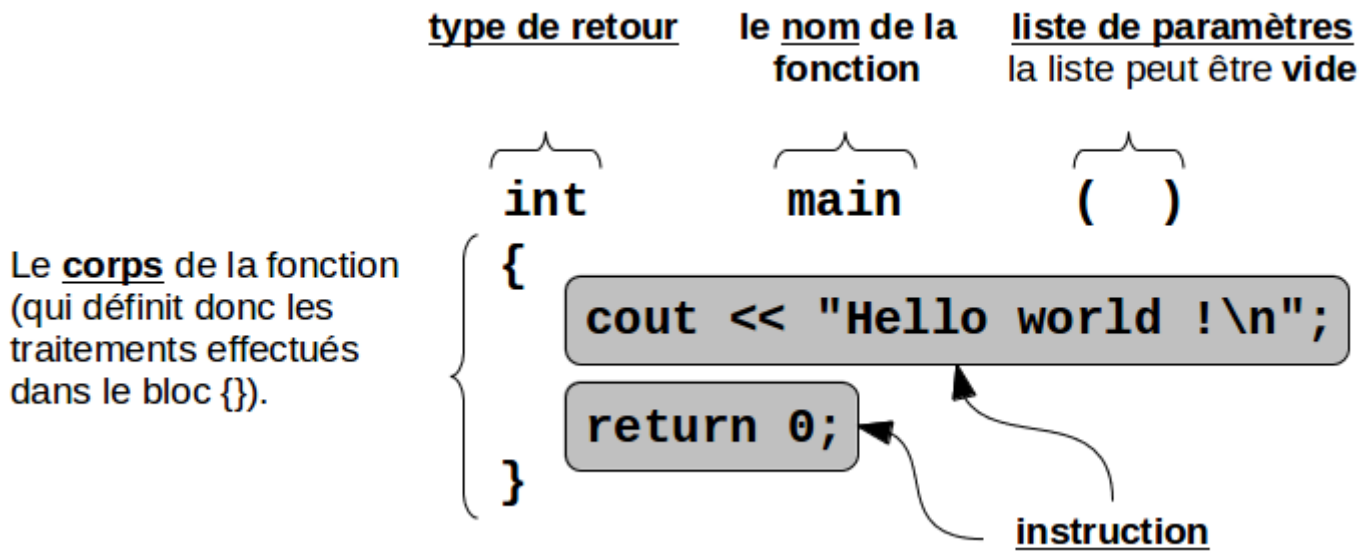
Tout programme C/C++ doit posséder une **fonction** nommée `main` (dite fonction principale) pour indiquer où commencer l'exécution. Une fonction est essentiellement une **suite d'instructions** que l'ordinateur exécutera dans l'ordre où elles sont écrites.

Une fonction comprend quatre parties :

- un **type de retour** : ici `int` (pour *integer* ou entier) qui spécifie le genre de résultat que la fonction retournera lors de son exécution. En C/C++, le mot `int` est un mot réservé (un mot-clé) : il ne peut donc pas être utilisé pour nommer autre chose.
- un **nom** : ici `main`
- une **liste de paramètres** entre parenthèses (que l'on verra plus tard) : ici la liste de paramètres est vide

– un **corps de fonction** entre accolades qui énumère les instructions que la fonction doit exécuter

Remarque : la plupart des instructions C/C++ se terminent par un point-virgule (`;`).



En C/C++, les **chaînes de caractères** sont délimitées par des guillements anglais (`"`). `"Hello world !\n"` est donc une chaîne de caractères. Le code \n est un "caractère spécial" indiquant le passage à une nouvelle ligne (*newline*).

Le nom `cout` (***c****haracter* ***out****put stream*) désigne le **flux de sortie standard** (l'écran par défaut). Les caractères "placés dans cout" au moyen de l'opérateur de sortie `<<` aparaîtront à l'écran.

`// Affiche "Hello world !"` placé en fin de ligne est un **commentaire**. Tout ce qui est écrit après `//` sera ignoré par le compilateur (la machine). Ce commentaire rend le code plus lisible pour les programmeurs. On écrit des commentaires pour décrire ce que le programme est supposé faire et, d'une manière générale, pour fournir des informations utiles impossibles à exprimer directement dans le code.

La première ligne du programme est un commentaire classique :
il indique simplement ce que le programme est censé faire (et pas ce que nous avons voulu qu'il fasse !).
Prenez donc l'habitude de mettre ce type de commentaire au début d'un programme.

La fonction `main` de ce programme retourne la valeur 0 (`return 0;`) à celui qui l'a appelée. Comme `main()` est appelée par le "système", il recevra cette valeur. Sur certains systèmes (Unix/Linux), elle peut servir à vérifier si le programme s'est exécuté correctement. Un zéro (0) indique alors que le programme s'est terminé avec succès (c'est une convention UNIX). Évidemment, une valeur différente de 0 indiquera que le programme a rencontré une erreur. Et sa valeur précisera alors le type de l'erreur.

En C/C++, une ligne qui commence par un # fait référence à une **directive** du préprocesseur (ou de précompilation). Le préprocesseur ne traite pas des instructions C/C++ (donc pas de `";"`). Ici, la directive `#include <iostream>` demande à l'ordinateur de rendre accessible (d'"inclure") les fonctionnalités contenues dans un fichier nommé `iostream`. Ce fichier est fourni avec le compilateur et nous permet d'utiliser `cout` et l'opérateur de sortie `<<` dans notre programme.

Un fichier inclus au moyen de `#include` porte généralement l'extension `.h`. On l'appelle en-tête (*header*) ou **fichier d'en-tête**.

Remarque : En C++, il est maintenant inutile d'ajouter l'extension .h pour les fichiers d'en-tête standard.

La ligne `using namespace std;` indique que l'on va utiliser l'**espace de nom** `std` par défaut.

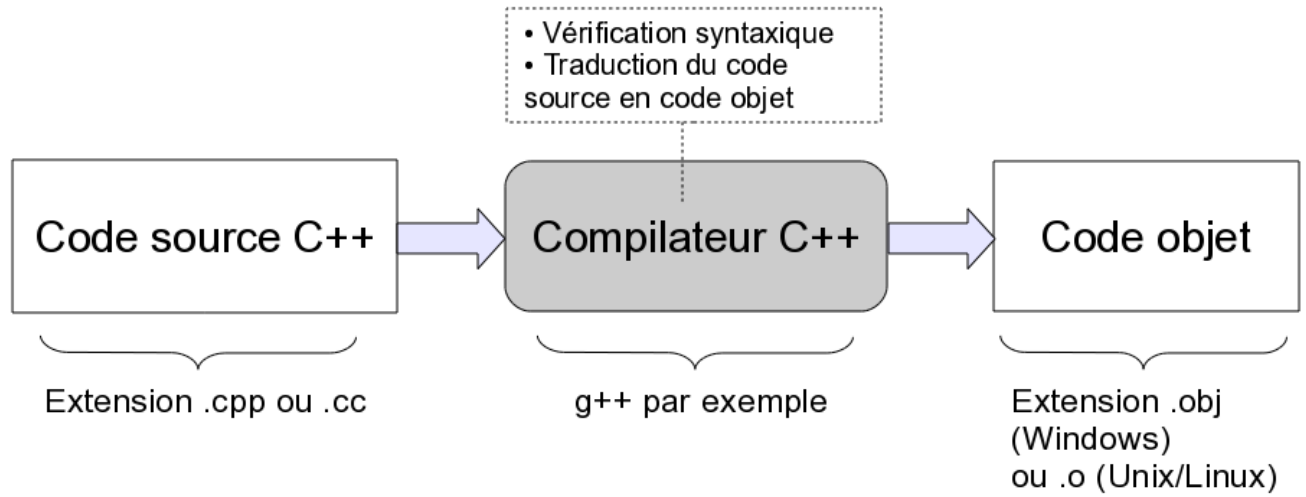
Remarque : `cout` (et `cin`) existe dans cet espace de nom mais pourrait exister dans d'autres espaces de noms. Le nom complet pour y accéder est normalement `std`::`cout`. L'opérateur :: permet la résolution de portée en C++ (un peu comme le `/` dans un chemin !).

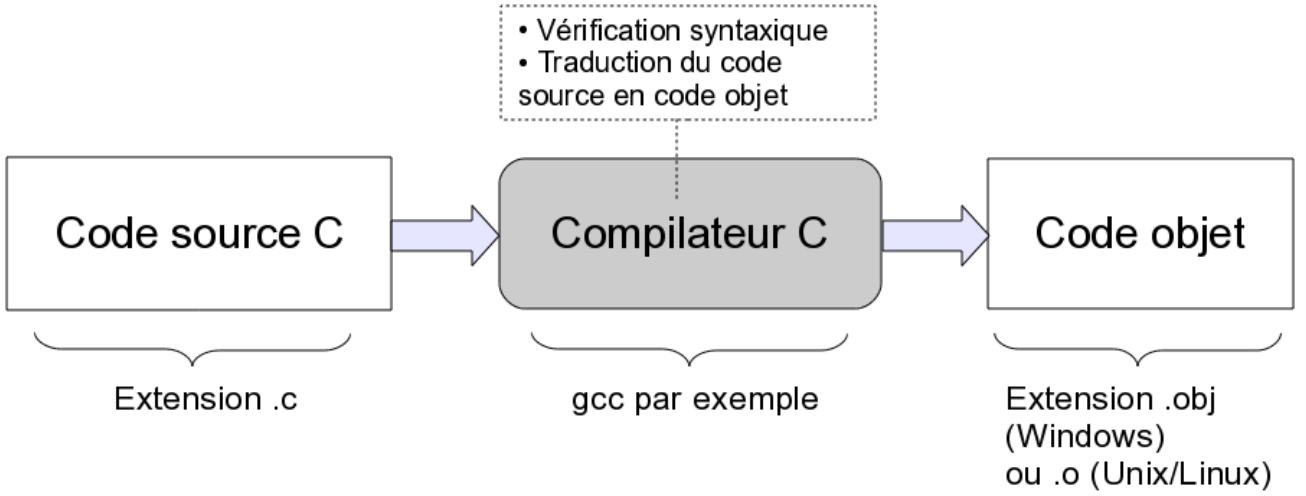
Pour éviter de donner systématiquement le nom complet, on peut écrire le code ci-dessous. Comme on utilise quasiment tout le temps des fonctions de la bibliothèque standard, on utilise presque tout le temps " `using namespace std;` " pour se simplifier la vie !

Compilation

C++ (ou C) est un langage compilé. Cela signifie que, pour pouvoir exécuter un programme, vous devez d'abord traduire sa forme lisible par un être humain (code source) en quelque chose qu'une machine peut "comprendre" (code machine). Cette traduction est effectuée par un programme appelé **compilateur**.

Ce que le programmeur écrit est le `code source` (ou programme source) et ce que l'ordinateur exécute s'appelle `exécutable`, **code objet** ou **code machine**.





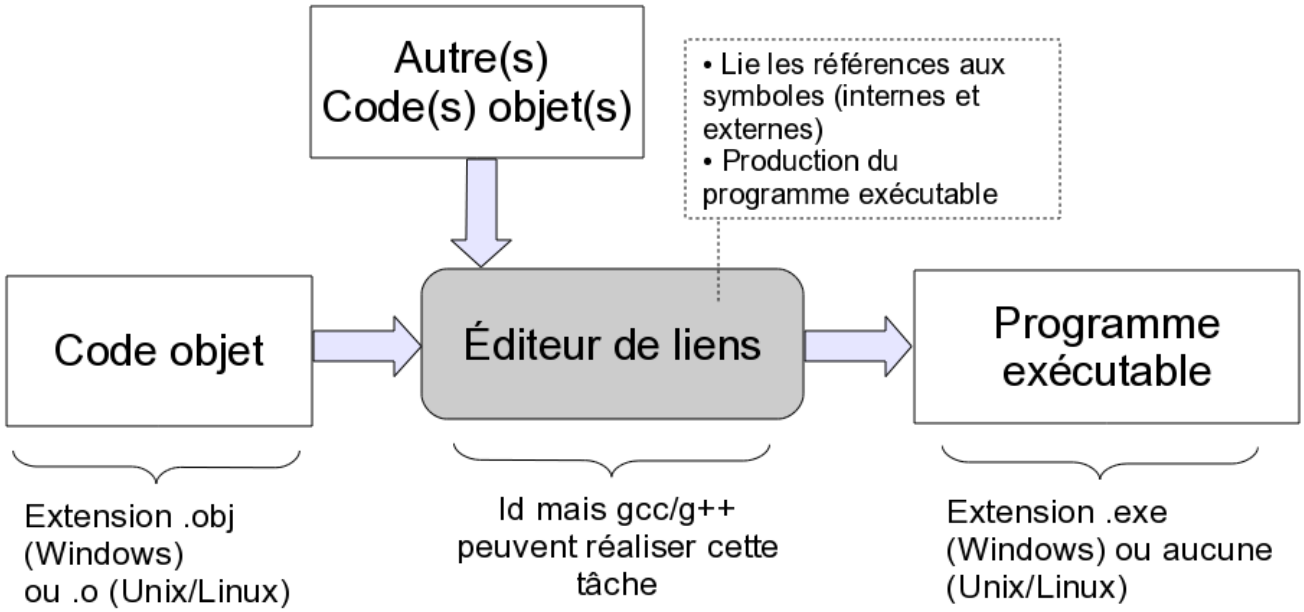
Vous allez constater que le compilateur est plutôt pointilleux sur la syntaxe ! Des manipulations de ce TP sont consacrées à découvrir cette syntaxe du langage C++. Comme tous les programmeurs, vous passerez beacoup de temps à chercher des erreurs dans du code source. Et la plupart de temps, le code contient des erreurs ! Lorsque vous coderez, le compilateur risque parfois de vous agacer. Toutefois, il a généralement raison car vous avez certainement écrit quelque chose qui n'est pas défini précisément par la norme C++ et qu'il empêche de produire du code objet.

Remarque : Le compilateur est dénué de bon sens et d'intelligence (il n'est pas humain) et il est donc très pointilleux. Prenez en compte les messages d'erreur et analysez les bien car souvenez-vous en bien le compilateur est "votre ami", et peut-être le meilleur que vous ayez lorsque vous programmez.

Édition des liens

Un programme contient généralement plusieurs parties distinctes, souvent développées par des personnes différentes. Par exemple, le programme “Hello world !” est constitué de la partie que nous avons écrite, plus d’autres qui proviennent de la **bibliothèque standard** de C++ (`cout` par exemple).

Ces parties distinctes doivent être liées ensemble pour former un programme exécutable. Le programme qui lie ces parties distinctes s’appelle un **éditeur de liens** (*linker*).



Remarque : notez que le code objet et les exécutables ne sont pas portables entre systèmes. Par exemple, si vous compilez pour une machine Windows, vous obtiendrez un code objet qui ne fonctionnera pas sur une machine Linux.

Une bibliothèque n’est rien d’autre que du code (qui ne contient pas de fonction `main` évidemment) auquel nous accédons au moyen de **déclarations** se trouvant dans un fichier d’en-tête. Une déclaration est une suite d’instruction qui indique comment une portion de code (qui se trouve dans une bibliothèque) peut être utilisée. Le débutant a tendance à confondre bibliothèques et fichiers d’en-tête.

*Remarque : Une **bibliothèque dynamique** est une bibliothèque qui contient du code qui sera intégré au moment de l’exécution du programme. Les avantages sont que le programme est de taille plus petite et qu’il sera à jour vis-à-vis de la mise à jour des bibliothèques. L’inconvénient est que l’exécution dépend de l’existence de la bibliothèque sur le système cible. Une bibliothèque dynamique, Dynamic Link Library (.dll) pour Windows et shared object (.so) sous UNIX/Linux, est un fichier de bibliothèque logicielle utilisé par un programme exécutable, mais n’en faisant pas partie.*

Les erreurs détectées :
- par le compilateur sont des erreurs de compilation (souvent dues à des problèmes de déclaration)
- celles que trouvent l’éditeur de liens sont des erreurs de liaisons ou erreurs d’édition de liens (souvent dues à des problèmes de définition)
- Et celles qui se produiront à l’exécution seront des erreurs d’exécutions ou de “logique” (communément appelées *bugs*).

Généralement, les erreurs de compilation sont plus faciles à comprendre et à corriger que les erreurs de liaison, et les erreurs de liaison sont plus faciles à comprendre et à corriger que les erreurs d’exécution et les erreurs de logique.

Environnement de programmation

Pour programmer, nous utilisons un langage de programmation. Nous utilisons aussi un compilateur pour traduire le code source en code objet et un éditeur de liens pour lier les différentes portions de code objet et en faire un programme exécutable. De plus, il nous faut un programme pour saisir le texte du code source (un **éditeur de texte** à ne pas confondre avec un traitement de texte) et le modifier si nécessaire. Ce sont là les premiers éléments essentiels de ce qui constitue la **boîte à outils** du programmeur que l'on appelle aussi **environnement de développement**.

Si vous travaillez dans une fenêtre en mode ligne de commande (appelée parfois "mode console"), comme c'est le cas de nombreux programmeurs professionnels, vous devez taper vous-mêmes les différentes commandes pour produire un exécutable et le lancer.

```
[tv@alias iteration-2]$ vim node.cc
[tv@alias iteration-2]$ make
g++ -c  -o graphviz.o graphviz.cc
g++ -c  -o main.o main.cc
g++ -c  -o node.o node.cc
g++ -o main graphviz.o main.o node.o
[tv@alias iteration-2]$ ./main
```

FIGURE *1 – Exemple de développement sur la console*

Si vous travaillez dans un Environnement de Développement Intégré ou **EDI**, comme c'est aussi le cas de nombreux programmeurs professionnels, un simple clic sur le bouton approprié suffira.

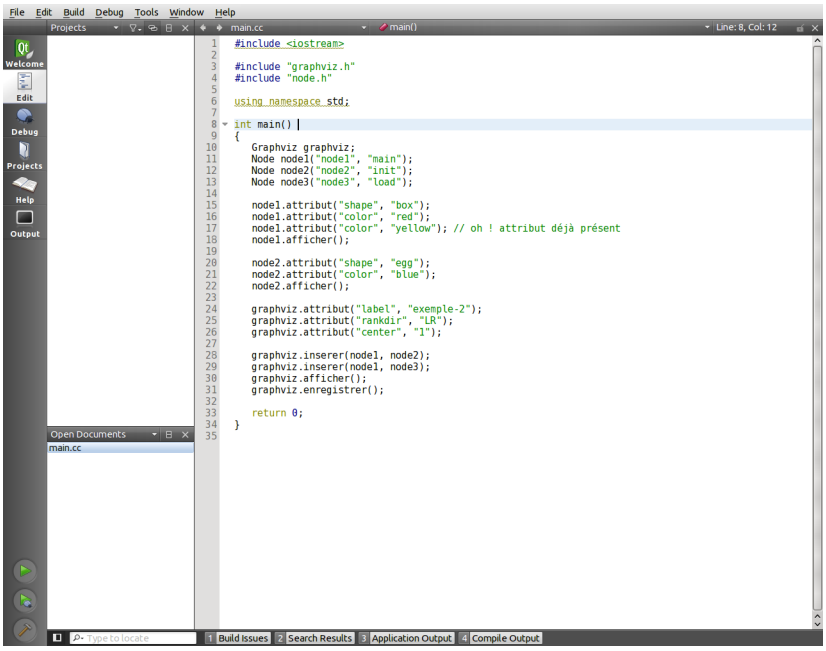


FIGURE *2 – Exemple de développement avec l'EDI Qt Creator*

Un EDI (ou IDE pour *Integrated Development Environment*) peut contenir de nombreux outils comme : la documentation en ligne, la gestion de version et surtout un **débogueur** (*debugger*) qui permet de trouver des erreurs et de les éléminer.

Remarque : Il existe de nombreux EDI (ou IDE) pour le langage C/C++ et on en utilisera certains notamment en projet. On peut citer : Visual C++, Builder, Qt Creator, Code::Blocks, devcpp, eclipse, etc ... Ils peuvent être très pratique mais ce ne sont que des outils et l'apprentissage du C/C++ ne nécessite pas forcément d'utiliser un EDI. Ils améliorent surtout la productivité dans un cadre professionnel.

Manipulations

Objectifs

L'objectif de cette partie est la mise en oeuvre de la chaîne de fabrication `g++` sous Linux.

Étape n°1 : création de votre espace de travail

Dans votre répertoire personnel, créez un répertoire "c++" où vous stockerez l'ensemble des vos TP. Entrez dans ce répertoire et faites un nouveau répertoire dedans nommé "tp1" où vous stockerez vos travaux pour ce premier TP.

Pour réaliser cela, vous pouvez soit utiliser l'**interface graphique avec l'explorateur de fichiers** (`dolphin` par exemple) soit utiliser une **console** (`Konsole` pour une environnement **KDE** ou `Terminal` pour un environnement **Gnome**) en tapant simplement les commandes suivantes :

```
$ mkdir c++
$ cd c++
$ mkdir tp1
$ cd tp1
```

*Remarque : la commande **`mkdir`** premet de créer un nouveau répertoire et la commande **`cd`** de se déplacer à l'intérieur de celui-ci. Le dollar (**`$`**) représente le prompt ou invite de commandes.*

Étape n°2 : édition du programme source

À l'aide d'un éditeur de texte (vi, vim, emacs, kwrite, kate, gedit, **geany** sous Linux ou Notepad, Notepad++, UltraEdit sous Windows, ...), tapez (à la main, pas de copier/coller, histoire de bien le lire et de s'habituer à la syntaxe !) le programme suivant dans un fichier que vous nommerez "`helloworld.cpp`" :

```cpp
#include <iostream>

using namespace std;

int main()
{
   int decompte = 5;

   while(decompte > 0)
   {
      cout << "Hello ";
      cout << "world" << " !" << endl;
      decompte = decompte - 1;
   }

   return 0;
}
```

Hello world (version 2) en C++

Étape n°3 : vérification du bon fonctionnement du compilateur

Tout d'abord ouvrez une console et vérifiez que le compilateur C++ est bien installé en tapant simplement la commande suivante :

```
$ g++
```

Si cela vous répond "`g++: no input file`" ou "`g++: pas de fichier à l'entrée`", alors tout va bien : votre GNU/Linux a bien réussi à exécuter le compilateur... et ce dernier se plaint juste que vous ne lui avez pas dit quoi compiler.

L'installation du compilateur est bien faite et vous pourrez continuer.

Si au contraire GNU/Linux vous répond (en français par exemple) "`bash: g++ : commande introuvable`". Alors c'est que l'interpréteur de commandes (`bash`) n'est pas en mesure de trouver le compilateur installé ou qu'il n'est pas du tout installé. Demandez alors l'aide de l'enseignant.

Étape n°4 : placement dans le bon répertoire

Avant de pouvoir compiler notre premier programme, il nous faut tout d'abord nous déplacer dans la console (la fenêtre noire) pour nous placer dans le repertoire où se trouve le fichier "`helloworld.cpp`" créé précédemment. Pour cela, appprendre quelques commandes `bash` est nécessaire. La première commande à connaitre est "`ls`" ("`ls -l`" pour avoir plus de détails) qui affiche le contenu du répertoire dans lequel vous vous trouvez actuellement. Essayez ! La commande équivalente sous Windows est "`dir`".

La deuxième commande à connaitre est "`cd`" qui change le répertoire où vous vous trouvez : par exemple, pour aller dans le sous-répertoire "c++", vous allez taper :

```
$ cd c++
```

Remarquez que l'invite de commande (en anglais on appelle ça le "prompt", c'est-à-dire le message qui précède le curseur sur la dernière ligne qui vient d'apparaitre) contient toujours le nom du répertoire courrant.

Recommencer l'opération cette fois pour rentrer dans le sous-répertoire "tp1" :

```
$ cd tp1
```

Si vous voulez remonter d'un niveau, rien de plus simple car le nom de répertoire "`..`" indique, où que vous soyez, le répertoire qui se trouve immédiatement au dessus. On l'appelle le **répertoire parent**.

```
$ cd ..
```

Un autre nom de répertoire particulier est "`.`" : c'est le répertoire dans lequel vous êtes actuellement. On l'appelle le **répertoire courant**.

Souvenez-vous que sous Linux, il faut taper "`./helloworld`" pour lancer l'exécution du programme "`helloworld`" (sous Linux, les fichiers exécutables n'ont pas d'extension particulière contrairement à Windows qui possède l'extension "`.exe`" pour les programmes). En fait cela signifie : "dans le répertoire courant" / "le fichier helloworld".

A votre avis, quel est le résultat de la commande suivante :

```
$ cd .
```

Pourquoi ?

Vous avez peut être remarqué qu'une barre oblique sépare les répertoires, et qu'elle n'est pas dans le même sens sous Windows "\" et Linux "/" ? Apprennez à les repérer et à ne pas vous tromper ! Au lieu de faire deux fois la commande "`cd`", on aurait pu aller directement au bon endroit en une seule fois :


```
cd c++\tp1 Sous Windows
cd c++/tp1 Sous Linux
```

Faites maintenant "`ls -l`" (ou "`dir`" et vérifiez que votre fichier "`helloworld.cpp`" apparait bien dans la liste. Si ce n'est pas le cas, appelez l'enseignant à l'aide.

Étape n°5 : fabrication (enfin !) du premier programme

Cela se fait comme vu en cours avec les deux commandes suivantes :

```
g++ -Wall -c helloworld.cpp
```

qui réalise le **"pré-processing", la compilation et l'assemblage** du fichier source "`helloworld.cpp`" en un fichier objet "`helloworld.o`" dans le même repertoire. Faites "`ls -l`" pour vérifier que le fichier "`helloworld.o`" a bien été créé.

Vous devez ensuite faire :

```
g++ -Wall -o helloworld helloworld.o
```

qui réalise l'**édition des liens** entre les divers fichiers ".o" (unification des variables et des fonctions contenues dans ces différents fichiers), puis qui produit le fichier exécutable "`helloworld`".

Vous pouvez maintenant démarer le programme, sous Linux, souvenez-vous qu'il faut indiquer que le programme se trouve dans le repertoire courant en tapant : "./helloworld" pour le démarrer.

```
./helloworld
```

Questions de révision

L'idée de base des questions de révision est de vous donner une chance de voir si vous avez identifié et compris les points clés de cette partie.

Question 1. Quel est le but du programme "Hello world !" ?

Question 2. Quelles sont les quatre parties d'une fonction ?

Question 3. Citez une fonction qui doit apparaître dans tout programme C ou C++.

Question 4. Dans le programme "Hello world !", à quoi sert la ligne `return 0;` ?

Question 5. Quel est le rôle du compilateur ?

Question 6. À quoi sert la directive `#include` ?

Question 7. Que signifie l'extension `.h` à la fin d'un nom de fichier en C/C++ ?

Question 8. Que fait l'éditeur de liens pour votre programme ?

Question 9. Quelle est la différence entre un fichier source et un fichier objet ?

Question 10. Qu'est-ce qu'un environnement de développement intégré (EDI) et que fait-il pour vous ?

Exercice 1 : corriger des erreurs

L'objectif de cet exercice est d'apprendre à interpréter les erreurs signalées par le compilateur.

Question 11. Éditez le fichier "`a-corriger.cpp`" ci-dessous et tentez de le compiler. Quel compilateur faut-il utiliser ? `g++` ou `gcc` ?

```
/Qu'est censé faire ce programme ?

// auteur : e. remy

include <iostrem>

using namespace std;

integer main()
{
 /* Vous devez corriger ce programme et arriver à le compiler puis l'exécuter.
 cout << 'Le programme marche !' << end;
 int valeur = 10
 cout << "valeur =" << valeur << end;
 // Attention : la division de deux entiers est une division euclidienne,
 // c'est-à-dire une division ***ENTIERE*** !
 int quotient = 10 / 3
 cout << "quotient=" << quotient << end;
 int reste = 10 % 3
 cout << "reste=" << reste << end;
 // Si vous voulez faire une division réelle, il faut convertir un des
 // arguments en réel :
 out << "quotient reel =" << valeur / 3.0 <<end; // Cette fois-ci 3.0 est réel
 out << "Fin du programme;
 return 0;
}
```

Un fichier source truffé d'erreurs !

La compilation échoue car le fichier est truffé d'erreurs !

Question 12. Corrigez-les jusqu'à obtenir le bon fonctionnement du programme. Puis, ajouter le commentaire classique au début du programme source.

Quelques consignes importantes :
- Vous pouvez commencer par tenter de corriger toutes les erreurs que vous trouver par vous-même en lisant le programme... mais au delà, c'est au compilateur de vous dire où sont les erreurs de syntaxe.
- Ne considérez que la première erreur signalée par le compilateur : les suivantes peuvent être une conséquence de la mauvaise compréhension de la suite du programme par le compilateur à cause de cette première erreur... il faut donc la corriger en premier ! Une fois qu'elle est corrigée, essayez de recompiler pour voir si vous avez bien corrigé, et s'il reste d'autres erreurs... La programmation est une véritable école de patience !
- Utilisez le numéro de ligne indiqué par le compilateur. Soit l'erreur se trouve à la ligne indiquée... soit un peu avant : le numéro de ligne indiqué est l'endroit où il devient manifeste pour le compilateur que le programme est erroné... mais des fois vous avez pu écrire une bêtise qui n'est pas litéralement fausse et donc que le compilateur accepte pendant quelques lignes... jusqu'à ce que cela devienne clair qu'il y a un problème !

– Si vous ne trouvez pas la source de l'erreur, n'hésitez pas à appeler à l'aide ! L'enseignant est là pour vous aider. Plus tard, quand vous rédigerez vos propres programmes, sachez également que quand on a "le nez dedans", on ne voit pas toujours ses propres fautes, mais qu'elles sont souvent évidentes pour quelqu'un qui a un regard neuf sur votre programme : demander une relecture à un de vos camarades s'avère donc souvent très efficace.

Exercice 2 : faire évoluer un programme

L'objectif de cet exercice est de faire évoluer un programme existant.

Question 13. Testez le programme ci-dessous. Que permet de faire `cin` ? Proposez une définition pour `cin` ?

```
// Affiche à l'écran un entier saisi par l'utilisateur

#include <iostream>

int main (int argc, char **argv)
{
   int n;

   std::cout << "Donnez un entier : " << std::endl;
   std::cin >> n;
   std::cout << "Vous avez donné l'entier : " << n << std::endl;

   return 0;
}
```

Une saisie clavier avec cin

Question 14. Modifiez le programme "Hello world (version 2)" pour qu'il puisse afficher **n** fois le message "Hello world !" (n étant une valeur saisie par l'utilisateur). Que se passe-t-il si l'utilisateur saisi une valeur négative ?

Bilan

Qu'y a-t-il de si important à propos du programme "Hello world !" ? Son objectif était de vous familiariser avec les outils de base utilisés en programmation.

Retenez cette règle : il faut toujours prendre un exemple extrêmement simple (comme "Hello world") à chaque fois que l'on découvre un nouvel outil. Cela permet de diviser l'apprentissage en deux parties : on commence par apprendre le fonctionnement de base de nos outils avec un programme élémentaire puis on peut passer à des programmes plus compliqués sans être distraits par ces outils. Découvrir les outils et le langage simultanément est beaucoup plus difficile que de le faire un après l'autre.

Conclusion : cette approche consistant à simplifier l'apprentissage d'une tâche complexe en la décomposant en une suite d'étapes plus petites (et donc plus faicles à gérer) ne s'applique pas uniquement à la programmation et aux ordinateurs. Elle est courante et utile dans la plupart des domaines de l'existence, notamment dans ceux qui impliquent une compétence pratique.

Descartes (mathématicien, physicien et philosophe français) dans le *Discours de la méthode* :

« diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre. »

« conduire par ordre mes pensées, en commençant par les objets les plus simples et les plus aisés à connaître, pour monter peu à peu comme par degrés jusques à la connaissance des plus composés ... »

