TP Programmation C/C++
Fabrication d'un programme simple

© 2017 *tv* `<tvaira@free.fr>` v.1.1

Sommaire

Premier programme **2**
Hello world en C++ 2
Explications 2
Hello world en C 4
Structure d'un programme source 4
Compilation 5
Édition des liens 6
Environnement de programmation 7
Manipulations 8
Objectifs 8
Étape n°1 : création de votre espace de travail 8
Étape n°2 : édition du programme source 8
Étape n°3 : vérification du bon fonctionnement du compilateur 9
Étape n°4 : placement dans le bon répertoire 9
Étape n°5 : fabrication (enfin !) du premier programme 10
Étape n°6 : analyse des fichiers 10
Questions de révision 13
Exercice 1 : à vous de jouer 13
Exercice 2 : corriger des erreurs 14
Exercice 3 : faire évoluer un programme 15
Bilan 15
Annexe 1 : environnement de développement 16
Annexe 2 : éditer un fichier texte avec `vim` 17
Annexe 3 : Une liste succinte de commandes de base 18

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).

Les Travaux Pratiques 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 compétition dans l'un de ces domaines sans pratique régulière. Vous savez bien quel serait le résultat.

Premier programme

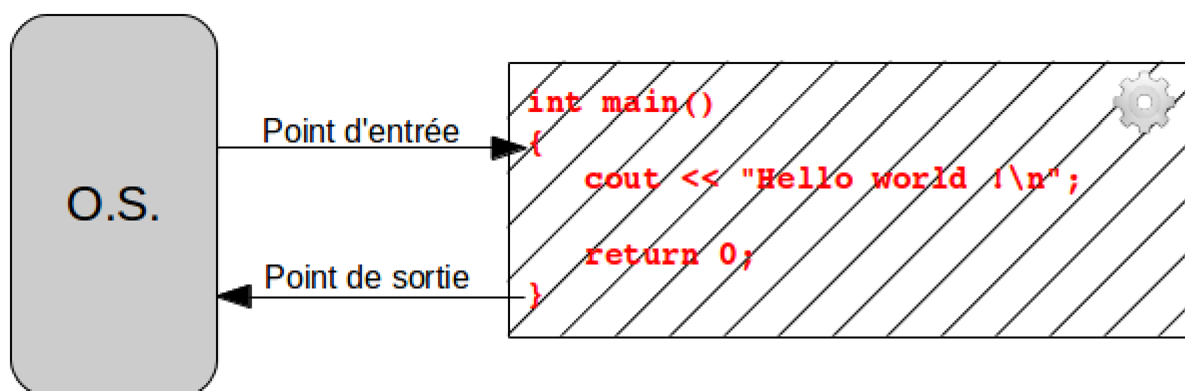
Hello world en C++

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



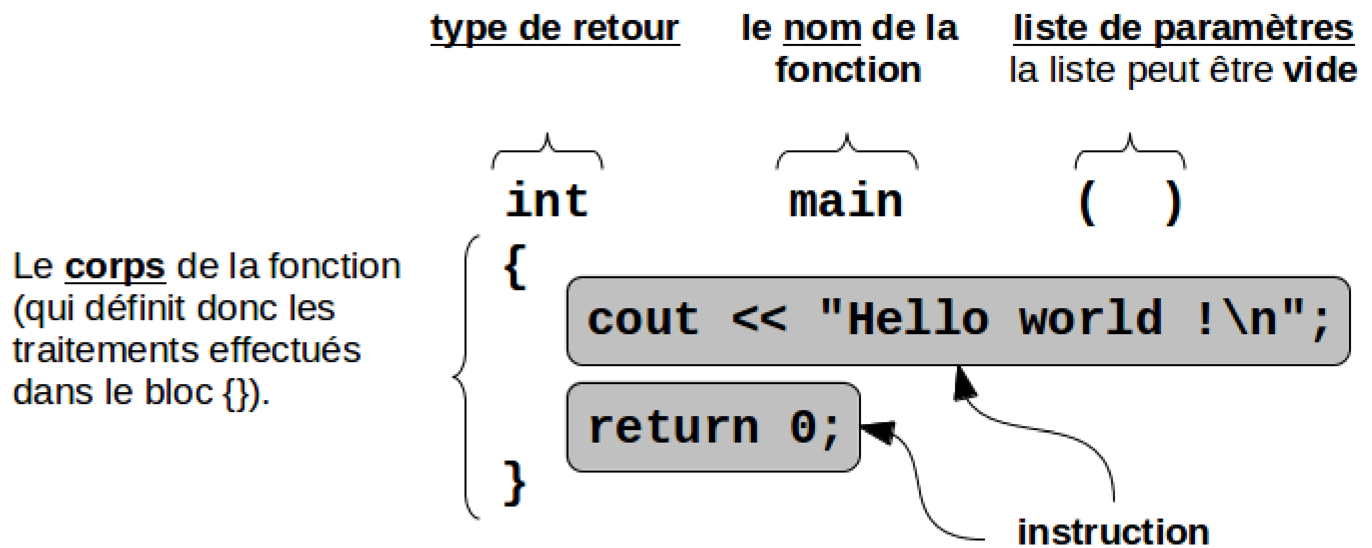
Exécution d'un programme « binaire » par le système d'exploitation

Tout programme C/C++ doit posséder une (et une seule) **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 type 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`, c'est le nom donné à la fonction (attention `main` est un mot-clé.
- une **liste de paramètres (ou d'arguments)** 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

✍ *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. Les langages C/C++ admettent aussi les commentaire multi-lignes avec : `/* ... */`.

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

 © 2012-2017 <tvaira@free.fr>

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 pré-compilation). 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` ou `.hpp`. On l'appelle en-tête (*header*) ou **fichier d'en-tête**.

✍ *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.

✍ *`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 !

Hello world en C

Voici la version du programme précédent pour le langage C :

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

#include <stdio.h> /* pour printf */

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

   return 0;
}
```

Hello world (version 1) en C

✍ *`cout` (et `cin`) n'étant pas disponible en C, on utilisera `printf` (et `scanf`) pour afficher sur le flux de sortie standard (et lire sur le flux d'entrée). Les fonctions `printf` (et `scanf`) sont bien évidemment utilisables en C++. Il est même parfois conseillé de les utiliser car `cin` et `cout` peuvent être jusqu'à 10 fois plus lents.*

Structure d'un programme source

Pour l'instant, la structure d'un programme source dans un fichier unique est donc la suivante :

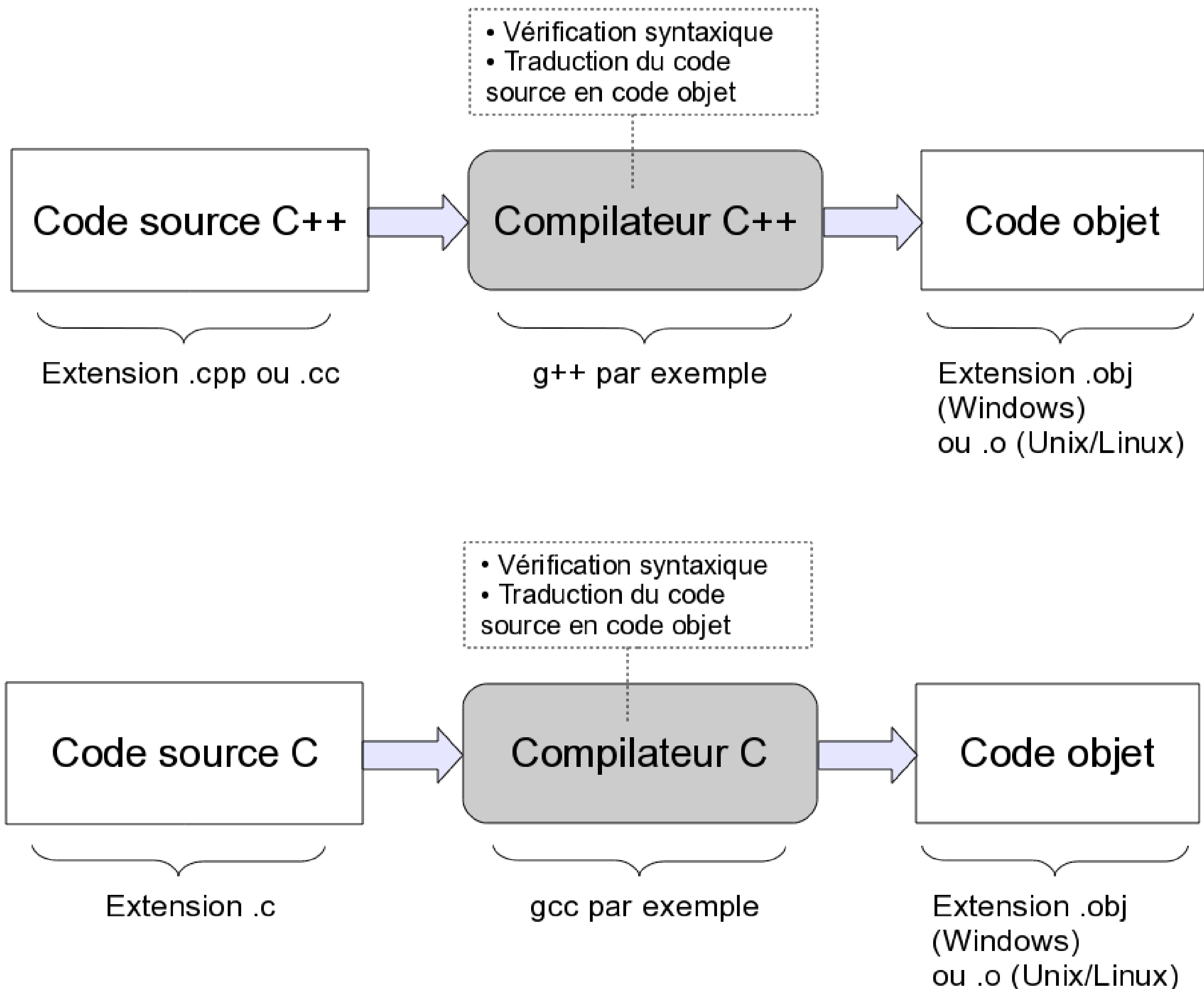
1. inclure les fichiers d'en-tête contenant les déclarations de fonctions externes à utiliser (`#include`)
2. définir la fonction principale (`main`)

✍ *Par la suite, on y ajoutera des déclaration de constantes et de structures de données, des fonctions et on le décomposera même en plusieurs fichiers ! Chaque chose en som temps ...*

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**.



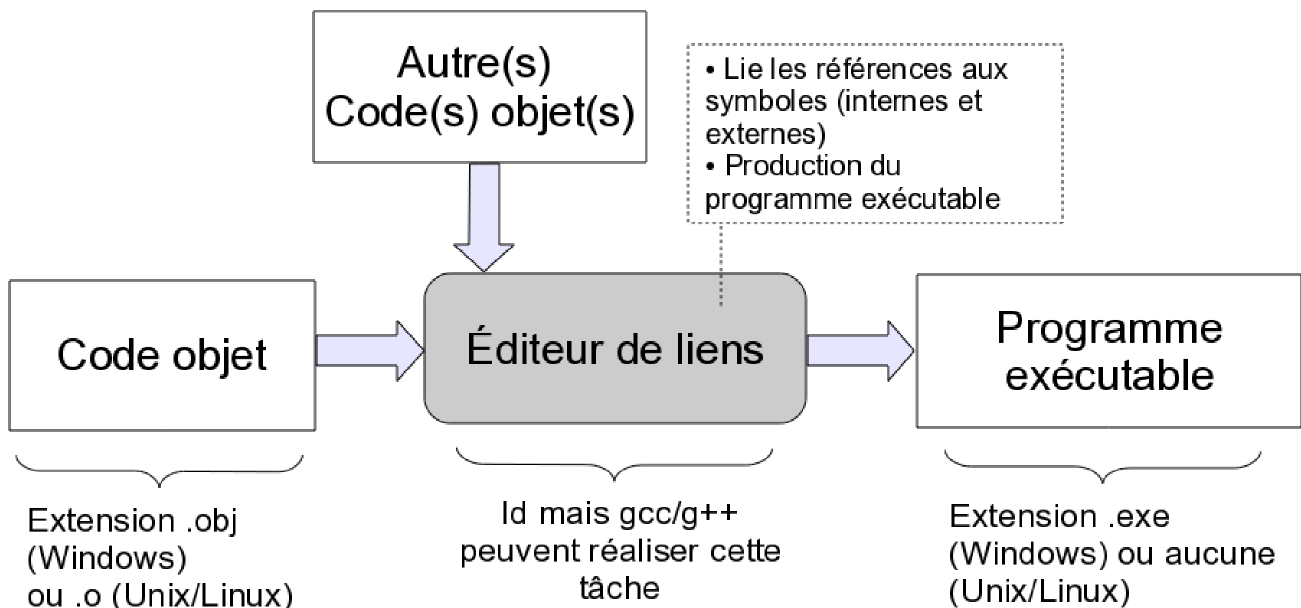
Vous allez constater que le compilateur est plutôt pointilleux sur la syntaxe ! Des manipulations de ce TP sont consacrées à découvrir cette syntaxe du langage C++. Comme tous les programmeurs, vous passerez beacoup de temps à chercher des erreurs dans du code source. Et la plupart de temps, le code 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.

✍ *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*).



✍ *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.

✍ *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.

 © 2012-2017 <tvaira@free.fr>

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.

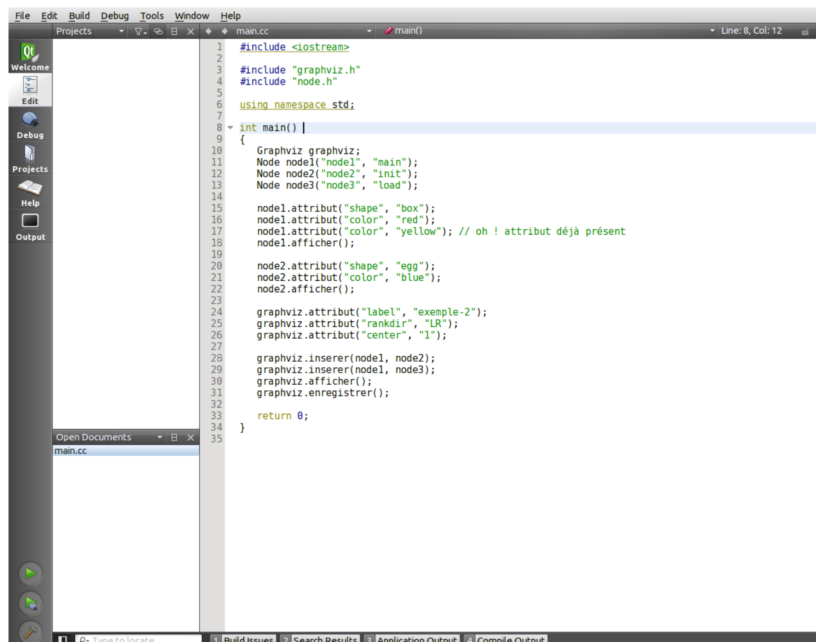


FIGURE *2 – Exemple de développement avec l'EDI Qt Creator*

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.

✍ *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.*

☞ **Voir l'Annexe n°1 (page 16) sur les environnements de développement.**

 © 2012-2017 <tvaira@free.fr>

Manipulations

Objectifs

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

Étape n°1 : création de votre espace de travail

Dans votre répertoire personnel, créez un répertoire "dev" où vous stockerez l'ensemble des vos TP de développement. 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** (`nautilus` par exemple) soit utiliser une **console** (`Konsole` pour une environnement **KDE** ou `Terminal` pour un environnement **Gnome**) en tapant simplement les commandes suivantes :

```
$ mkdir dev
$ cd dev
$ mkdir tp1
$ cd tp1
```

✍ *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.*

☞ **L'Annexe n°3 (page 18) fournit une liste de commandes de base sous GNU/Linux.**

Étape n°2 : édition du programme source

À l'aide d'un éditeur de texte (vi, **vim**, emacs, kwrite, kate, gedit, **geany** sous GNU/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; // je suis une variable entière initialisée avec la valeur 5

   while(decompte > 0) // je suis une boucle qui fait ... tant que la variable decompte est
       supérieur à 0
   {
      cout << "Hello ";
      cout << "world" << " !" << endl;
      decompte = decompte - 1; // je suis une instruction qui effectue deux opérations : une
           soustraction puis une affectation
   }

   return 0;
}
```

Hello world (version 2) en C++

☞ **Si vous l'utilisez (et c'est fortement conseillé), voir l'Annexe n°2 (page 17) sur l'éditeur de texte `vim`.**

 © 2012-2017 <tvaira@free.fr>

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 "dev", vous allez taper :

```
$ cd dev
```

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 dev\tp1 Sous Windows
cd dev/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
```

Étape n°6 : analyse des fichiers

Vous avez manipulé différents types de fichiers. En informatique, on distingue essentiellement deux types de fichiers :

- Les fichiers « texte » qui ont un contenu pouvant être interprété comme du texte (une suite de bits représentant un caractère encodé à l'origine en *ASCII*).
- Les fichiers « binaire » qui respectent un format de fichier (convention normalisée ou non) utilisé pour représenter et stocker des données. Tout ce qui n'est pas un fichier texte est un fichier binaire ! Exemples : code machine (exécutable), fichiers multimédias (images, sons, vidéos, traitement de texte, etc.), fichiers archives, fichiers compressés, ...

✍ *Le système GNU/Linux vous donne accès à une documentation riche et variée. Il existe différentes commandes pour accèder à cette documentation : `help`, `man`, `info` ... Pour l'instant, prenez l'habitude de lire le **manuel** en utilisant la commande `man`. Par exemple : `man ascii`.*

Commençons par utiliser la commande `file` pour déterminer les différents types de fichiers :

➟ **Quel est le type du fichier `helloworld.cpp` ?**

```
$ file helloworld.cpp
helloworld.cpp: UTF-8 Unicode C program text
```

C'est un fichier « texte » (ici endodé en UTF-8 : cf. `man utf-8`).

➟ **Quel est le type du fichier `helloworld.o` ?**

```
$ file helloworld.o
helloworld.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

C'est un fichier « binaire » (ici le format est ELF « *Executable and Linking Format* » qui correspond à du code machine pour une plate-forme GNU/Linux 64 bits : cf. `man elf`).

➟ **Quel est le type du fichier `helloworld` ?**

```
$ file helloworld
helloworld: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses
   shared libs), for GNU/Linux 2.6.24, ... not stripped
```

C'est un fichier « binaire » (ici un exécutable lié dynamiquement pour une plate-forme GNU/Linux 2.6.24 64 bits).

Petite différence : le `helloworld.o` est un fichier **objet** (`relocatable`) et `helloworld` est un exécutable (`executable`). Les deux fichiers contiennent du code machine.

Pour visualiser le contenu d'un fichier, on va utiliser la commande `hexdump` qui va nous afficher sur 3 colonnes : l'adresse des octets dans le fichier, leurs valeurs en hexadécimal et la traduction en ASCII.

➟ **Que contient le fichier `helloworld.cpp` ?**

```
$ hexdump -C helloworld.cpp
00000000 2f 2f 20 43 65 20 70 72 6f 67 72 61 6d 6d 65 20 |// Ce programme |
00000010 61 66 66 69 63 68 65 20 6c 65 20 6d 65 73 73 61 |affiche le messa|
00000020 67 65 20 22 48 65 6c 6c 6f 20 77 6f 72 6c 64 20 |ge "Hello world |
00000030 21 22 20 c3 a0 20 6c 27 c3 a9 63 72 61 6e 0a 0a |!" .. l'..cran..|
00000040 23 69 6e 63 6c 75 64 65 20 3c 69 6f 73 74 72 65 |#include <iostre|
00000050 61 6d 3e 0a 0a 75 73 69 6e 67 20 6e 61 6d 65 73 |am>..using names|
00000060 70 61 63 65 20 73 74 64 3b 0a 0a 69 6e 74 20 6d |pace std;..int m|
00000070 61 69 6e 28 29 0a 7b 0a 20 20 20 63 6f 75 74 20 |ain().{. cout |
00000080 3c 3c 20 22 48 65 6c 6c 6f 20 77 6f 72 6c 64 20 |<< "Hello world |
00000090 21 5c 6e 22 3b 20 2f 2f 20 41 66 66 69 63 68 65 |!\n"; // Affiche|
000000a0 20 22 48 65 6c 6c 6f 20 77 6f 72 6c 64 20 21 22 | "Hello world !"|
000000b0 0a 20 0a 20 20 20 72 65 74 75 72 6e 20 30 3b 0a |. . return 0;.|
000000c0 7d 0a                                           |}.|
```

Pour visualiser seulement le contenu « texte », on peut utiliser les commandes `cat`, `more`, `strings` ...

```
$ cat helloworld.cpp
// 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;
}
```

➟ **Que contient le fichier `helloworld.o` ?**

```
$ hexdump -C helloworld.o
00000000 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00 |.ELF............|
00000010 01 00 3e 00 01 00 00 00 00 00 00 00 00 00 00 00 |..>.............|
00000020 00 00 00 00 00 00 00 00 e0 01 00 00 00 00 00 00 |................|
00000030 00 00 00 00 40 00 00 00 00 00 40 00 0f 00 0c 00 |....@.....@.....|
```

```
00000040 55 48 89 e5 be 00 00 00 00 bf 00 00 00 00 e8 00 |UH..............|
00000050 00 00 00 b8 00 00 00 00 5d c3 55 48 89 e5 48 83 |........].UH..H.|
00000060 ec 10 89 7d fc 89 75 f8 83 7d fc 01 75 2a 81 7d |...}..u..}..u*.}|
00000070 f8 ff ff 00 00 75 21 bf 00 00 00 00 e8 00 00 00 |.....u!.........|
00000080 00 b8 00 00 00 00 ba 00 00 00 00 be 00 00 00 00 |................|
00000090 48 89 c7 e8 00 00 00 00 c9 c3 55 48 89 e5 be ff |H.........UH....|
000000a0 ff 00 00 bf 01 00 00 00 e8 ad ff ff ff 5d c3 00 |.............]..|
000000b0 48 65 6c 6c 6f 20 77 6f 72 6c 64 20 21 0a 00 00 |Hello world !...|
000000c0 00 00 00 00 00 00 00 00 00 47 43 43 3a 20 28 55 |.........GCC: (U|
000000d0 62 75 6e 74 75 2f 4c 69 6e 61 72 6f 20 34 2e 36 |buntu/Linaro 4.6|
000000e0 2e 33 2d 31 75 62 75 6e 74 75 35 29 20 34 2e 36 |.3-1ubuntu5) 4.6|
...
```

✍ *Un fichier « binaire » peut contenir du texte ! La commande* `strings` *permet d'afficher seulement le contenu texte d'un fichier binaire.*

Le fichier `helloworld.o` contient du code objet, c'est-à-dire du code machine. Le code machine est une suite d'instructions connues et exécutées par le processeur. Ce code machine peut être « désassemblé » :

```
$ objdump -d helloworld.o

helloworld.o:   file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   be 00 00 00 00          mov    $0x0,%esi
   9:   bf 00 00 00 00          mov    $0x0,%edi
   e:   e8 00 00 00 00          callq  13 <main+0x13>
  13:   b8 00 00 00 00          mov    $0x0,%eax
  18:   5d                      pop    %rbp
  19:   c3                      retq
...
```

Question 1. Expliquer les 2 colonnes de la ligne `13:` ?

Question 2. Quel est le nom du langage de programmation qui a été obtenu après désassemblage ?

Question 3. Est-il possible à partir d'un code objet de « remonter » jusqu'à un code en langage C ?

Question 4. Donner la commande qui permet de passer d'un code source en langage C à un code source en assembleur ?

➟ Que contient le fichier `helloworld` ?

La « même chose » que le fichier `helloworld.o` : c'est-à-dire du code machine. La seule différence est que le fichier `helloworld` a été lié dynamiquement pour le rendre **exécutable** sur le système d'exploitation GNU/Linux.

Il est possible de visualiser les bibliothèques logicielles dont dépend l'exécutable avec la commande `ldd` :

```
$ ldd helloworld
        linux-vdso.so.1 => (0x00007fffd41ab000)
        libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007fd0a8f12000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fd0a8b51000)
```

```
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fd0a8854000)
/lib64/ld-linux-x86-64.so.2 (0x00007fd0a923b000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007fd0a863e000)
```

✍ *Les bibliothèques logicielles à lien dynamique ont l'extension `.so` (Shared Object) sous GNU/Linux et `.dll` (Dynamic Link Library) sous ©Windows.*

Question 5. À la suite de ces manipulations, lister les noms et rôles des différentes commandes GNU/Linux utilisées ?

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 6. Quelles sont les quatre parties d'une fonction ?

Question 7. Citez une fonction qui doit apparaître dans tout programme C ou C++.

Question 8. Quel est le rôle du compilateur ? Que fait l'éditeur de liens pour votre programme ?

Question 9. À quoi sert la directive `#include` ? Que signifie l'extension `.h` à la fin d'un nom de fichier en C/C++ ?

Question 10. Quelle est la différence entre un fichier source et un fichier objet ?

Question 11. Qu'est-ce qu'un environnement de développement intégré (EDI) et que fait-il pour vous ?

Exercice 1 : à vous de jouer

À partir des manipulations précédentes, réaliser la fabrication du programme "Hello world" en C :

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

#include <stdio.h> /* pour printf */

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

   return 0;
}
```

Hello world (version 1) en C

Question 12. Quelle est l'extension à donner au fichier source ?

Question 13. Donner les différentes commandes permettant la fabrication de l'exécutable ?

Question 14. Pourrait-on utiliser le compilateur C++ pour fabriquer l'exécutable ? Pourrait-on utiliser le compilateur C pour fabriquer l'exécutable à partir du code source `helloworld.cpp` ?

Exercice 2 : corriger des erreurs

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

Question 15. 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 16. 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 littéralement 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 3 : faire évoluer un programme

L'objectif de cet exercice est de faire évoluer un programme existant. *C'est une pratique que l'on détaillera par la suite.*

Question 17. Testez le programme ci-dessous. Indiquer (comme vous l'avez appris) ce que fait ce programme ?

```
//

#include <stdio.h>

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

   printf("Donnez un entier : ");
   scanf("%d", &n); // le & signifie l'adresse de la variable n et, c'est nécessaire ici
      pour que la fonction puisse modifier la variable passée en argument
   printf("Vous avez donné l'entier : %d\n", n);

   return 0;
}
```

Une saisie clavier

✍ *Le* ***%d*** *est un spécificateur de conversion de l'argument* ***n****, un* ***int*** *ici, qui sera converti en un chiffre décimal signé. La fonction* ***scanf()*** *lit une donnée depuis le flux d'entrée standard* ***stdin*** *et l'affecte à la variable* ***n****. Ces fonctions sont décrites dans le chapitre 3 du manuel :* ***man 3 printf****.*

Question 18. Quel est le nom donné au chapitre 3 du manuel ? Quel est la valeur de retour de la fonction `printf` ? `scanf` ?

Question 19. Écrire la version C++ de ce programme en utilisant `cout` (à la place de `printf`) et `cin` (à la place de `scanf`). Que permet de faire `cin` ?

Question 20. Modifiez le programme précédent 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 saisit 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 ... »

Annexe 1 : environnement de développement

Si vous souhaitez développer sur un autre système d'exploitation que GNU/Linux, il vous faudra installer sur votre système une **chaîne de développement** comprenant au minimum un éditeur de texte et un compilateur.

Quelques solutions :

➟ Éditeur de texte GUI [1] (libre et gratuit) :

- Komodo Edit : `https://www.activestate.com/komodo-ide/downloads/edit`
- Notepad++ : `https://notepad-plus-plus.org/download/`

✍ *voir aussi Framapack qui est un outil qui vous permet d'installer une collection de logiciels libres pour ©Windows de votre choix en une seule fois.*

➟ Compilateur (libre et gratuit) pour ©Windows :

- MinGW : `http://www.mingw.org/`
- Cygwin : `http://www.cygwin.com/`

➟ EDI [2] (IDE [3]) :

- Code::Blocks : `http://www.codeblocks.org/`
- Netbeans : `https://netbeans.org/downloads/`
- Eclipse : `https://www.eclipse.org/downloads/`
- Dev-C++ ©Windows : `http://orwelldevcpp.blogspot.fr/`
- ...

✍ *L'ensemble de ces logiciels existent aussi sous GNU/Linux à l'exception de Dev-C++ et évidemment de MinGW/Cygwin.*

Si vous voulez disposer d'un environnement de développement GNU/Linux, il vous faut :

- utiliser une distribution "live" (DVD ou clé USB)
- installer GNU/Linux sur votre disque dur (en double *boot* éventuellement)
- installer GNU/Linux dans une machine virtuelle (avec VirtualBox par exemple)

1. GUI : *Graphical User Interface*
2. EDI : Environnement de Développement Intégré
3. IDE : *Integrated Development Environment*

– installer Cygwin sur votre machine ©Windows

Si vous voulez programmer sans installer d'environnement de développement sur votre machine, vous pouvez utiliser des "compilateurs" en ligne (cf. CodingGround) :

– C : `https://www.tutorialspoint.com/compile_c_online.php`
– C99 : `https://www.tutorialspoint.com/compile_c99_online.php`
– C++ : `https://www.tutorialspoint.com/compile_cpp_online.php`
– C++ 11 : `https://www.tutorialspoint.com/compile_cpp11_online.php`

Annexe 2 : éditer un fichier texte avec vim

`vi` est l'éditeur de texte standard d'Unix et il a été l'éditeur favori de nombreux *hackers* jusqu'à l'arrivée d'`Emacs` en 1984. Tout système se conformant aux spécifications Unix intègre `vi` et il est donc encore largement utilisé par les utilisateurs (surtout les administrateurs et programmeurs) des différentes variantes d'Unix.

La version incluse actuellement dans les **Linux** est le plus souvent `vim` (*vi improved*), un clone de `vi` qui comporte quelques différences avec celui-ci. `vi/vim` comprend trois modes de fonctionnement : le mode normal, le mode **commande** et le mode **insertion**. Après le lancement de `vi/vim`, c'est le mode normal qui est actif. Pour passer en mode insertion (de texte évidemment) il faut appuyer sur la touche [i] ou [o] . On sait que l'on est en mode insertion par l'affichage de `INSERT` en bas de la fenêtre. Pour sortir de ce mode, il faut appuyer sur la touche [Esc] et l'affichage de `INSERT` en bas de la fenêtre disparaît. Pour passer en mode commande, il faut taper ':'.

Quelques commandes intéressantes :

```
:q! : sortie sans sauvegarde
:wq : sortie avec sauvegarde
:x  : sortie avec sauvegarde
ZZ  : sortie avec sauvegarde
:w  : sauvegarde sans sortie
$   : se déplacer sur le dernier caractère de la ligne
Ctrl f : afficher la page suivante
Ctrl b : afficher la page précédente
Ctrl d : afficher la demi-page suivante
Ctrl u : afficher la demi-page précédente
e  : se déplacer à la fin du mot
b  : se déplacer au début du mot
w  : se déplacer au début du mot suivant
H  : se déplacer en haut de l'écran
L  : se déplacer en bas de l'écran
M  : se déplacer au milieu de l'écran
```

```
z. : décaler l'affichage avec la ligne courante au centre
z (return) : décaler l'affichage avec la ligne courante en haut
z- : décaler l'affichage pour que la ligne courante
:num_ligne : se déplacer à la ligne num_ligne
G (ou :$)  : aller à la fin du fichier
u   : annulation de la dernière modification
dd  : suppression de la ligne courante
2dd : suppression des deux lignes suivantes
D   : suppression de la fin de la ligne à partir du curseur
:3,7 d : suppression des lignes 3 à 7
:3,7 t 10 : copie des lignes 3 à 7 après la ligne 10
:3,7 m 10 : transfert des lignes 3 à 7 après la ligne 10
yy   : mémorisation de la ligne courante (copier)
3yy  : mémorisation des 3 lignes suivantes (copier)
p    : copie ce qui a été mémorisé après le curseur
P    : copie ce qui a été mémorisé avant le curseur
:set nu : affichage des numéros de ligne
/mot : recherche le mot mot (on se déplace avec n ou N ou *)
```

✍ *Il existe en réalité une quantité astronomiques de commandes dans* ***vi****, et en particulier dans* ***vim****, et chaque personne utilise, en général, qu'un petite partie d'entre elles en fonction de ses habitudes (et souvent, pas les mêmes que vous...).*

Annexe 3 : Une liste succinte de commandes de base

Voici quelques commandes usuels :

```
dpkg    : un gestionnaire de paquet pour Debian
apt-get : utilitaire APT pour la gestion des paquets (voir aussi aptitude)
alias   : crée ou supprime des alias de commandes
pwd     : affiche le chemin d'accès au répertoire courant
man     : permet de consulter les manuels de référence
clear   : efface l'écran
echo    : affiche une ligne de texte (et aussi des variables)
cd      : permet de se déplacer dans une arborescence
ls      : liste le contenu d'un répertoire
rm      : supprime un fichier (voir aussi rmdir)
cp      : permet la copie de fichier (voir aussi cp -a)
mv      : déplace ou renomme une partie d'une arborescence
mkdir   : crée un répertoire dans une arborescence
touch   : modifie l'horodatage d'un fichier (permet aussi de créer un fichier vide)
file    : affiche le type des fichiers
type    : indique le type pour une commande
locate  : localise un fichier
find    : recherche des fichiers sur le système
cat     : affiche et/ou concatène le(s) fichier(s) sur la sortie standard
more    : affiche à l'écran l'entrée standard (page par page)
less    : idem avec possibilité de retour en arrière
cut     : permet d'isoler des colonnes dans un fichier
head    : affiche les n première lignes
join    : joint les lignes de deux fichiers en fonction d'un champ commun
sort    : trie les lignes de texte en entrée
paste   : concatène les lignes des fichiers
```

```
tail   : affiche les n dernières lignes d'un fichier
tac    : concatène les fichiers en inversant l'ordre des lignes
uniq   : élimine les doublons d'un fichier trié
rev    : inverse l'ordre des lignes d'un fichier
diff   : compare des fichiers texte
cmp    : compare deux fichiers octet par octet
tr     : remplace ou efface des caractères
grep   : recherche des chaines de caractères dans des fichiers
sed    : éditeur de flux pour le filtrage et la transformation de texte
awk    : manipulation avancée de fichiers texte
md5sum : génère et vérifie un hachage MD5
whereis : permet de trouver l'emplacement d'une commande
whatis : donne une description d'une commande
which  : donne le chemin complet d'une commande
du     : affiche une arborescence et sa taille (du -h)
df     : fournit la quantité d'espace occupé par les systèmes de fichiers (df -Th)
od     : affiche le dump d'un fichier (voir aussi hexdump)
wc     : compte les caractères, les mots et les lignes en entrée
date   : affiche et modifie la date et l'heure
cal    : affiche le calendrier
bc     : calculatrice
ln     : crée des liens physiques et symboliques
lsattr : liste les attributs des fichiers
chattr : change les attributs des fichiers
stat   : affiche des informations sur un fichier ou un système de fichier
lsof   : affiche des informations sur les fichiers ouverts
fuser  : identifie les processus utilisant des fichiers
fdisk  : gère les tables de partitions pour Linux
cfdisk : manipule les table de partitions pour Linux (voir aussi sfdisk)
dd     : convertit et copie un fichier physiquement
sync   : vider les tampons du système de fichiers (finalise les opérations d'écriture)
id     : affiche les identifiants d'utilisateur et de groupe effectifs et réels
whoami : affiche l'identifiant d'utilisateur
who    : montre qui est connecté (voir aussi w et users)
last   : affiche une liste des utilisateurs dernièrement connectés
su     : change l'identifiant d'utilisateur ou permet de devenir un superutilisateur (root)
sudo   : exécute une commande sous un autre compte (voir /etc/sudoers)
uname  : affiche des informations sur le système
ps     : affiche les processus en cours
top    : affiche les tâches
kill   : envoye un signal à un processus
time   : exécute un programme et affiche un résumé des ressources utilisées
uptime : indique depuis quand le système a été mis en route
free   : affiche les quantités de mémoire libre et utilisée du système
printenv : affiche l'ensemble ou une partie des variables d'environnement
```