TP Programmation C/C++
Fabrication d'un programme simple
Éléments de cours

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
Bilan 8
Annexe 1 : environnement de développement 8
Annexe 2 : éditer un fichier texte avec `vim` 9
Annexe 3 : Une liste succinte de commandes de base 10

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

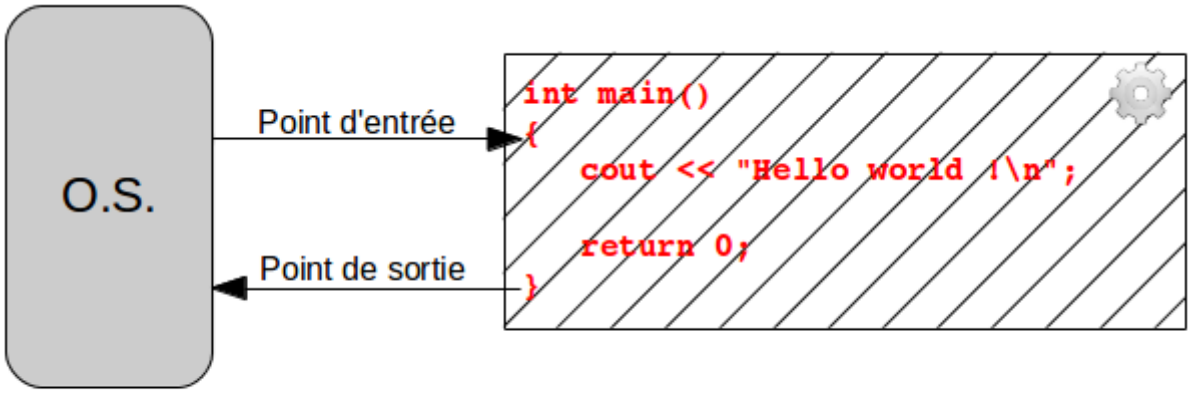
Hello world en C++

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



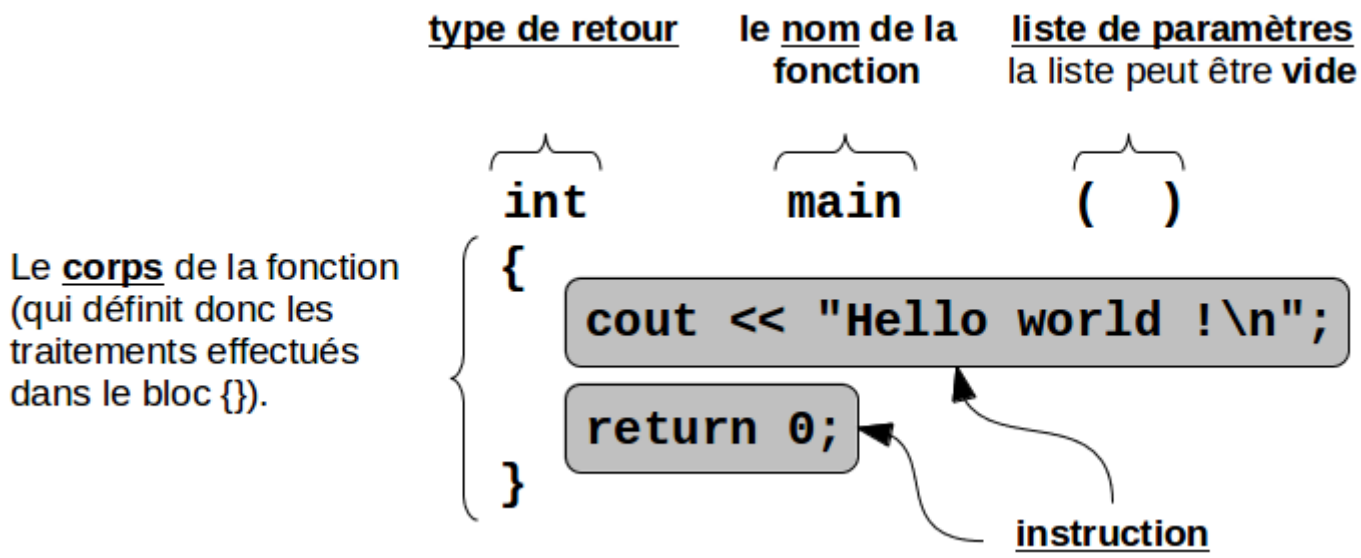
Exécution d'un programme « binaire » par le système d'exploitation

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 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` 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 :

```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 :

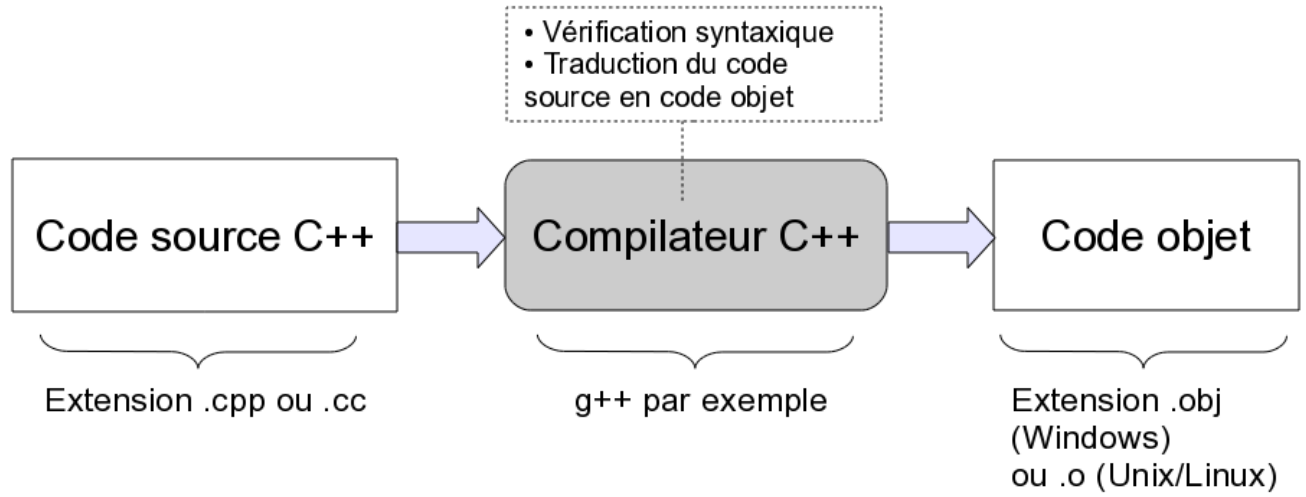
1. inclure les fichiers d'en-tête contenant les déclarations de fonctions externes à utiliser (`#include`)
2. définir la fonction principale (`main`)

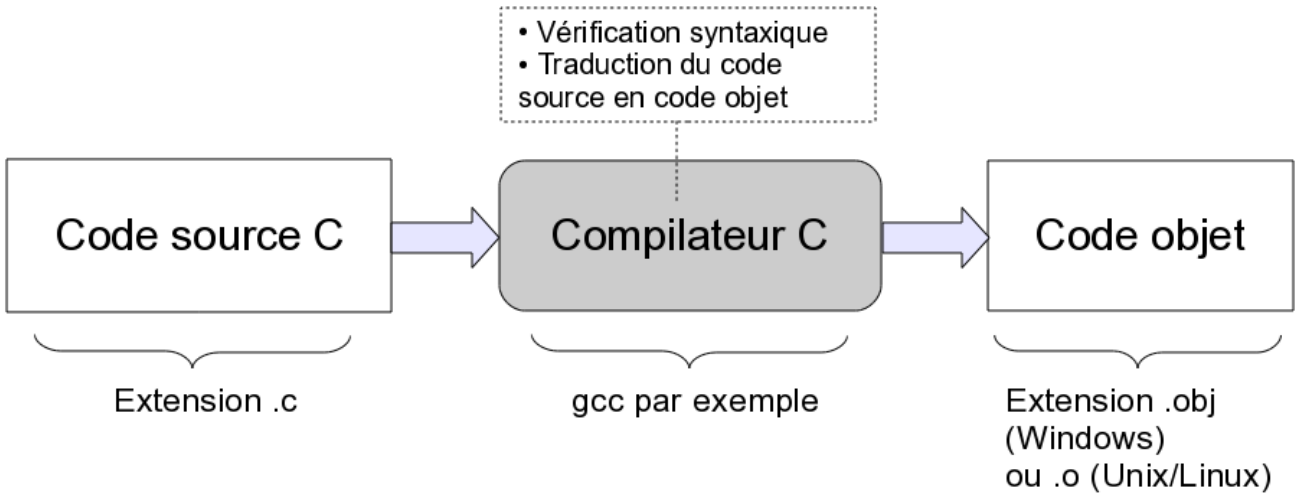
✍ *Par la suite, on y ajoutera des déclaration de constantes et de structures de données, des fonctions et on le décomposera même en plusieurs fichiers ! Chaque chose en som temps …*

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**.





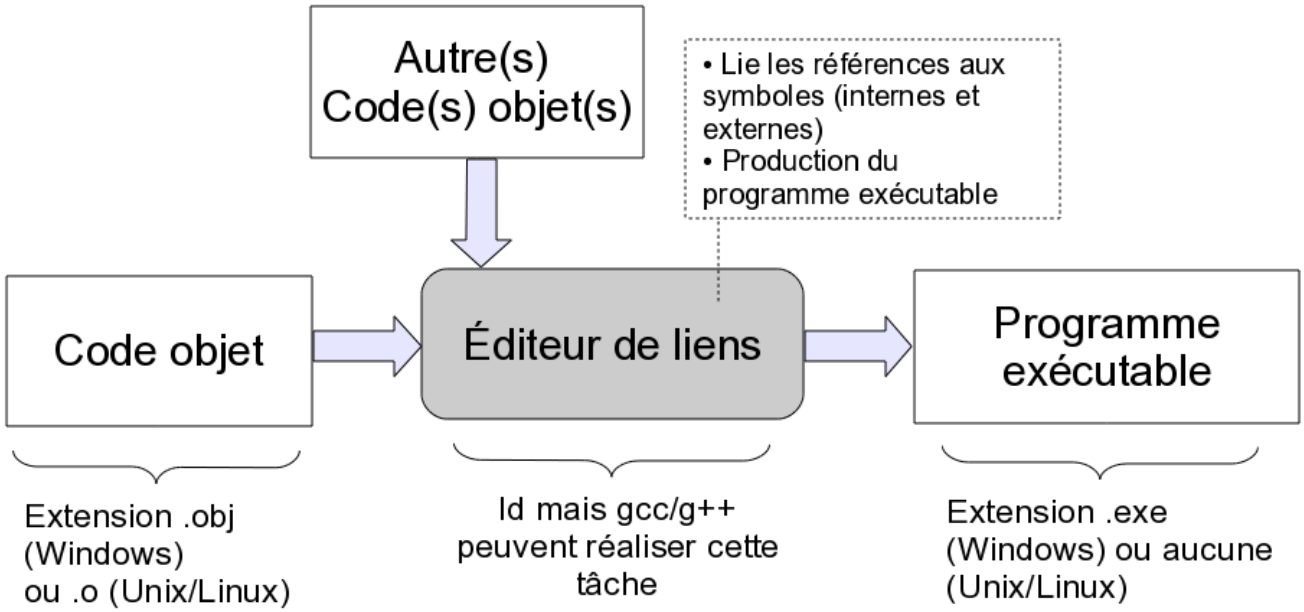
Vous allez constater que le compilateur est plutôt pointilleux sur la syntaxe ! Des manipulations de ce TP sont consacrées à découvrir cette syntaxe du langage C++. Comme tous les programmeurs, vous passerez beacoup de temps à chercher des erreurs dans du code source. Et la plupart de temps, le code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

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.

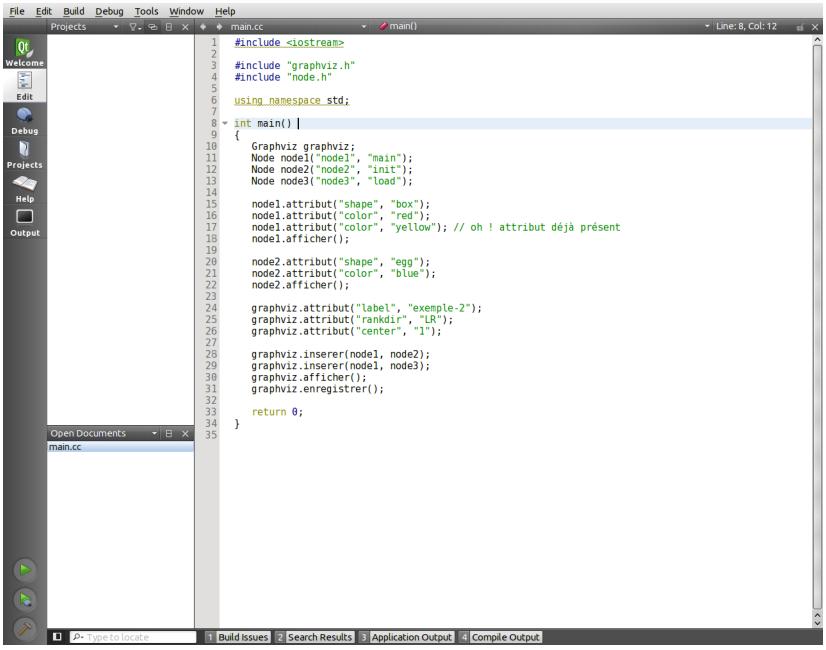


FIGURE *2 – Exemple de développement avec l'EDI Qt Creator*

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 8) sur les environnements de développement.**

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/`

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

– ...

✍ *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)
– 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
```

