Travaux pratiques n° 1
Introduction

1 Organisation de l'espace de travail

1.1 Écran

– Ouvrir un Terminal : menu "Applications > Accessoires" > "Terminal"
– Ouvrir un éditeur de texte pour saisir un programme : saisir `gedit&` dans le terminal

Disposer les deux fenêtres ouvertes de manière à les voir simultanément. **Ne pas** afficher ces fenêtres en mode "Pleine page".

1.2 Mémoire

L'espace de travail de votre ordinateur doit être organisé afin d'y ranger vos fichiers.

1.2.1 Définitions

Un **fichier** est un document électronique : il peut s'agir d'un programme que vous avez réalisé dans le cadre de ce module, mais aussi d'un document collecté pour un autre module ou pour votre usage personnel.

Un **répertoire** peut inclure d'autres répertoires ainsi que des fichiers.

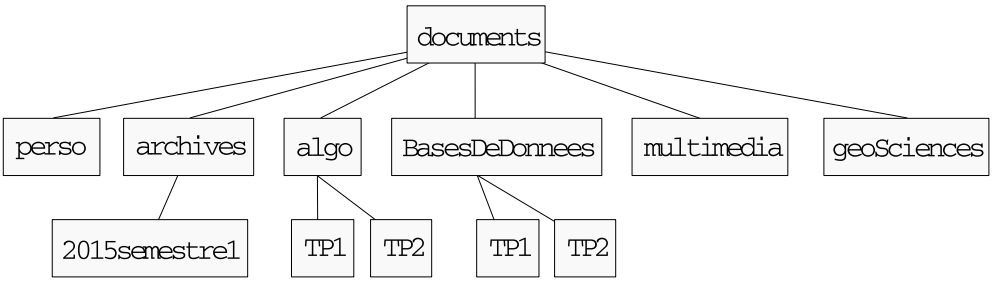
L'**arborescence des répertoires** représent les inclusions entre répertoires.

Le **répertoire courant** est le répertoire de travail, c'est-à-dire le répertoire où est située un programme qui est exécuté.

Un **chemin entre répertoire** permet de situer un répertoire par rapport à un autre. Il existe deux façons de spécifier un chemin : chemin absolu ou chemin relatif. Un **chemin absolu** part de la racine du système de fichier, indiquée par un '/' ; un **chemin relatif** décrit la chaîne de répertoires à traverser depuis le répertoire courant pour atteindre un fichier ou répertoire donné. Il faut écrire '..' pour indiquer le répertoire parent, '../..' pour le grand-parent, et ainsi de suite.

Exemple

Le répertoire `documents` inclut les répertoires `perso`, `archives`, `algo`, `BasesDeDonnees`, etc. Les répertoires `archives`, `algo` et `BasesDeDonnees` incluent d'autres répertoires.



Le chemin absolu pour désigner le répertoire `TP1` d'`algo` est : `/documents/algo/TP1`

Le chemin relatif pour aller du répertoir `TP1` d'`algo` vers le répertoire `TP2` d'`algo` est : `../TP2`

1.2.2 Lignes de commande

Voici quelques commandes acceptées par un terminal et dont vous aurez besoin en travaux pratiques :

cd *(change directory)* - cette commande suivie du nom d'un répertoire change le répertoire courant pour celui indiqué.

mkdir *(make directory)* - cette commande suivie du nom d'un répertoire crée un répertoire.

mv (remove) - cette commande suivie du nom d'un fichier efface le fichier.

mkdir (remove) - cette commande suivie du nom d'un fichier efface le fichier.

pwd *(print working directory)* - cette commande affiche le nom complet du répertoire courant ; la dernière partie est généralement reprise dans l'invite de commande de votre terminal.

ls *(list)* - affiche le contenu du répertoire courant, fichiers et sous-répertoires ; l'option `--color` peut s'avérer pratique pour distinguer répertoire, fichiers exécutables et autres fichiers.

Pratique
La touche de tabulation du clavier permet la complétion automatique des noms de répertoires et fichiers.

Conseil
Eviter les accents, espaces, signes de ponctuation dans les noms de fichiers et de répertoires.

Exercice
Créez quelques répertoires pour organiser votre espace de travail en mémoire. Rangez les documents déjà présents.

2 Introduction à C++

Les TP du module X2I0010 "Algorithmique et programmation" sont réalisés en C++. À l'inverse de Javascript, C++ est un langage *compilé* : le programme doit être écrit dans un fichier texte dont le nom est libre (éviter les espaces et caractères accentués toutefois) et dont l'extension est `.cpp`, par exemple `monprogramme.cpp` ; ce fichier devra être lu par un programme spécial appelé **compilateur** qui traduit le texte du programme en instructions-machine, exécutables par l'ordinateur. Le compilateur produit un nouveau fichier, par exemple `monprogramme.exe`. L'exécution de ce fichier réalise le traitement décrit dans le texte du programme. Toute modification du programme nécessite donc une recompilation suivie d'une exécution pour en mesurer les effets.

Les travaux pratiques sont réalisés sous le système $Linux$.

Vous utiliserez le compilateur **g++** développé par GNU et accessible en ligne de commande depuis un terminal. L' alias `gpp` fixe certaines options de compilation.

3 Un premier programme

3.1 Saisir

Dans un éditeur de texte, saisissez le programme suivant dans un nouveau document (ne pas saisir les numéros de ligne) :

```
1  /* table.cpp - auteur : C. Enguehard
2  Role : affiche la table de multiplication choisie par l'utilisateur.
3  */
4
5  #include <iostream>
6
7  using namespace std;
8
9  //-----------------------------------------
10 // fonction principale
11 int main ()
12 {
13   int nombre;
```

```
14    int table;
15
16    // saisie de la table
17    cout << "Quelle table de multiplication afficher ? ";
18    cin >> table;
19
20    // affichage de la table
21    for (nombre = 1; nombre <= 10; nombre++) {
22      cout << nombre << " x " << table << " = " << nombre * table << endl;
23    } // for
24    return(0);
25  }
```

les lignes 1-3 de ce programme sont un bloc de commentaire. La tradition veut que le nom du fichier contenant le programme soit noté sur la première ligne avec le nom du ou des auteurs. Ensuite est indiqué le rôle du programme.

Les lignes 5 et 7 constituent le préambule du programme. La ligne 5 indique que le programme utilisera les fonctionnalités d'entrées/sorties définies dans le fichier `iostream.h` de la bibliothèque standard C++. La ligne 7 indique que le programme utilisera sans préfixe les fonctionnalités de l'espace de nom (*namespace*) `std` ; cet espace de nom inclut la plupart des fonctionnalités de la bibliothèque standard C++ ; sans cette ligne il faudrait préfixer chacune de leur utilisation par `std::` (par exemple `std::cout` et `std::endl`).

Les lignes 9, 10, 16 et 20 et 23 comportent des commentaires d'une seule ligne ; ils peuvent occuper toute la ligne (ligne 9, 10, 16 et 20) ou bien suivre une instruction (ligne 23).

Les lignes 11 à 25 constituent le programme à proprement parler. `int` main() est la déclaration d'une fonction appelée `main` retournant un entier et n'ayant aucun paramètre ; c'est le nom consacré de la fonction principale d'un programme, celle qui est appelée automatiquement lorsque le programme est exécuté.

Les accolades '`{`' et '`}`' représentent le début et la fin d'une séquence d'instructions, appelé *bloc d'instructions* en C++. Il est courant de faire suivre l'accolade fermante d'un commentaire indiquant le nom du bloc qu'elle conclut.

Les lignes 13 et 14 constituent la partie déclaration de ce programme. Y sont déclarées deux variables de nom `nombre` et `table` et de type `int`, l'un des types représentant les entiers en C++. Ces deux variables auraient pu être déclarées ensemble : `int nombre, table;`

Les lignes 16 à 24 représentent le corps de notre programme. Celui-ci va afficher la table de multiplication choisie par l'utilisateur au moyen d'une répétitive `for` correspondant au "pour" algorithmique. Les lignes 17 et 22 sont des instructions d'affichage, la ligne 18 est une saisie dont le résultat est stocké dans la variable table.

La ligne 24 conclue notre programme. Comme un programme en C++ est une fonction retournant un entier, il faut retourner une valeur entière en fin de programme. La valeur de retour d'un programme est un code représentant l'erreur survenue lors de son exécution, 0 si aucune erreur ne s'est produite. La dernière instruction d'un programme sera donc toujours `return(0);`

Notez comment toute instruction se termine par un '`;`' en C++, y compris les déclarations et même certaines instructions du préambule.

Exercice :

Recopiez ce programme dans un fichier nommé "table.cpp" au moyen d'un éditeur de texte et enregistrez ce fichier dans votre répertoire "Informatique2/tp1".

3.2 Compiler

Ouvrez un terminal et changez le répertoire courant pour qu'il soit celui contenant votre fichier "table.cpp". Tapez alors la commande suivante :

```
gpp table.cpp -o table.exe
```

Apparemment il ne s'est rien passé ... pourtant si vous tapez la commande `ls` vous pourrez constater que votre répertoire contient à présent un fichier "table.exe". Il s'agit du résultat de la compilation de "table.cpp" ! Notez que vous pouvez donner le nom que vous voulez au fichier exécutable, même sans l'extension ".exe". Si vous omettez le nom (et l'option `-o` qui permet de le préciser) le fichier exécutable se nommera "a.out" par défaut. Il fonctionnera tout aussi bien que "table.exe".

Ouvrez le fichier exécutable avec un éditeur de texte. Il ne ressemble pas à votre programme, il s'agit de sa traduction dans le langage de bas niveau de la machine.

Retournez au texte de votre programme. Modifiez quelque chose, par exemple enlevez un '`;`'. Dans le terminal compilez à nouveau votre programme. Rien ne va plus : le compilateur émet une erreur ! Apprenez à lire les messages d'erreur du compilateur, ils vous informent sur les erreurs de programmation que vous avez commises. Les messages d'erreur ont toujours la même structure : d'abord le compilateur vous indique le fichier et la fonction en cause ("table.cpp" et fonction `int main()` ici) ; ensuite, un message indique la ligne où l'erreur s'est produite et une indication du type d'erreur, par exemple (en français) "j'attendais un ' ;' et j'ai trouvé autre chose". Attention, l'erreur se situe parfois à une ligne qui précède celle indiquée.

Le compilateur est actuellement configuré pour s'arrêter à la première erreur détectée, aussi vous devrez corriger vos erreurs une par une et recompiler votre programme à chaque fois. Aucun programme exécutable n'est produit tant que le programme contient des erreurs. Vous devrez donc corriger toutes vos erreurs avant de pouvoir tester votre programme.

Ajoutez maintenant l'instruction suivante à la ligne 15 de votre programme : `int j; // variable inutilisee`. Recompilez votre programme. Le compilateur grogne à nouveau, mais il ne s'agit pas d'une erreur cette fois : c'est un avertissement (*warning*) afin d'attirer votre attention sur une source potentielle d'erreur qui pourrait provoquer une exécution erronée de votre programme, ici le fait qu'une variable est déclarée mais pas utilisée. Les avertissements n'empèchent pas la compilation et le fichier exécutable est malgré tout créé. Vous êtes tout de même invité à considérer tout avertissement comme une erreur en puissance et à corriger votre programme en conséquence.

3.3 Exécuter

Une fois le programme corrigé et compilé avec sucès, sans erreur ni avertissement, il ne vous reste plus qu'à enfin tester votre programme. Dans un terminal, rendez vous dans le répertoire contenant le fichier exécutable "table.exe" et tapez la commande :

```
./table.exe
```

Vous devez jouer le rôle de l'utilisateur et saisir un entier suivi d'un retour à la ligne. Ensuite vous verrez s'afficher la table de multiplicatiuon de l'entier que vous avez choisi. Changez '10' en '20' dans le programme (ligne 21) et ré-exécutez. Y a-t-il un changement ? Avez-vous bien pensé à sauver le fichier modifié et à le recompiler ? N'oubliez pas : après chaque modification, sauvegarde et compilation ; en cas d'erreur (ou d'avertissement), correction, sauvegarde et compilation.

4 Exercice

Reprenez les algorithmes traités en TD. Pour transcrire vos algorithmes en programme, vous vous référerez au **mémento Algorithmique-C++** et au document présentant les **types C++** disponibles sur le site Madoc du module.

À la demande de votre enseignant, vous devrez rendre, sur Madoc, le programme correspondant à l'un des exercices de la feuille.

Veillez à bien rendre le fichier source du programme (fichier d'extension `.cpp`) que vous aurez pris soin de nommer `GRP_NOM1+NOM2_NTP-NEX.cpp` où GRP est votre numéro de groupe, $NOM1$ et $NOM2$ les noms des membres du binôme, NTP le numéro de la feuille de TDTP, et NEX le numéro de l'exercice. Ne pas oublier d'indiquer les noms des auteurs du programme au début de celui-ci et de toujours bien commenter vos programmes, quitte, s'ils ne compilent pas ou ne fonctionnent pas, à expliquer dans des commentaires les erreurs rencontrées et vos tentatives pour les lever : l'absence de remise vaudra toujours 0 alors que la remise d'un travail incomplet/incorrect mais bien commenté vous rapportera toujours des points.