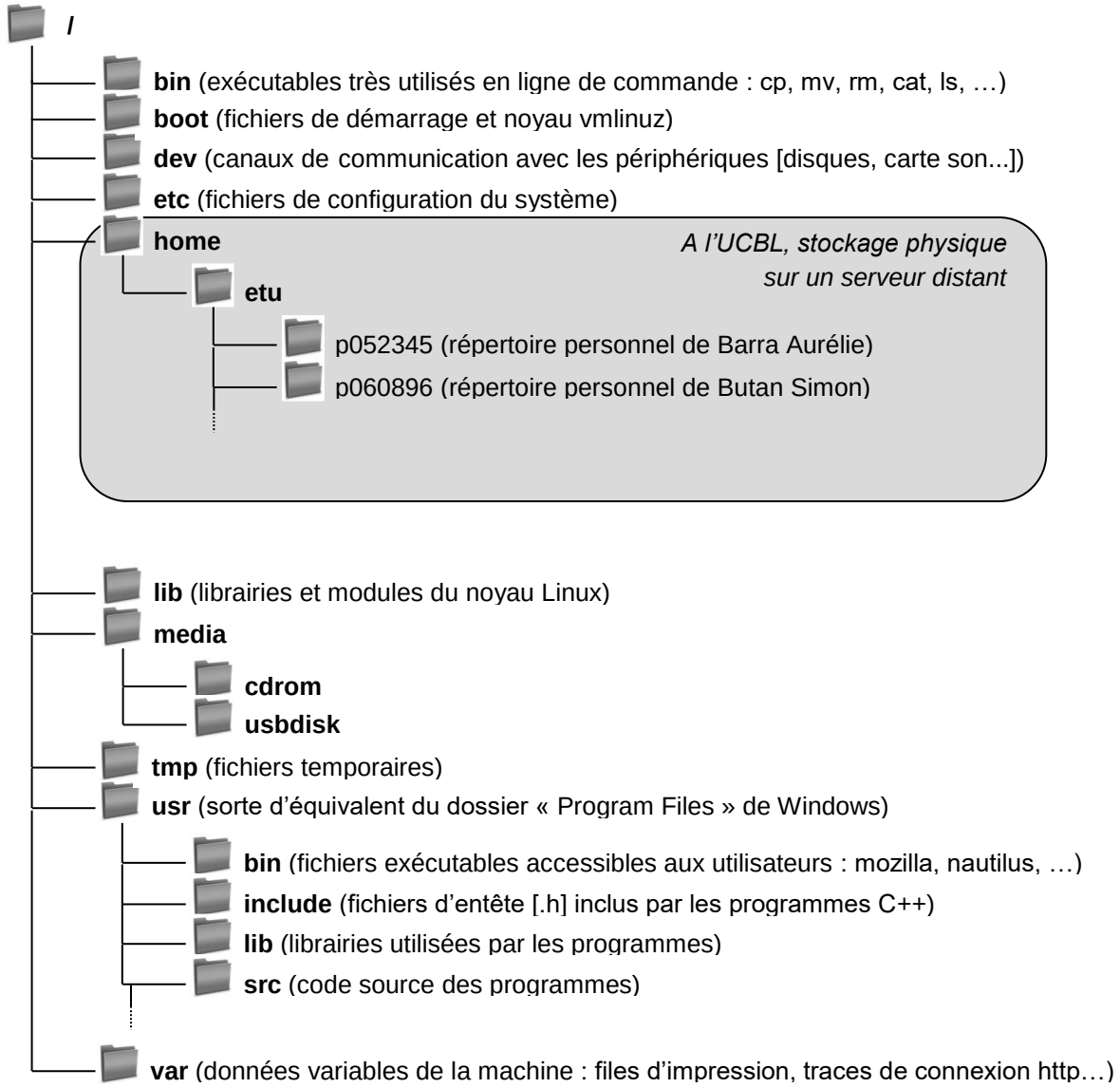
TP1 : De LIFAPI à LIFAPSD…

Exercice 1 : De Windows à Linux

Redémarrez l'ordinateur sous Linux (Ubuntu). Connectez-vous avec les mêmes login / mot de passe que sous Windows. Pour utiliser au mieux son compte Linux, il est nécessaire de connaître quelques notions basiques sur le système de fichiers Linux. La racine du système de fichier (l'équivalent du « C :\ » d'un Windows non partitionné) est le répertoire « **/** ». Voici un schéma des principaux répertoires que contient ce dossier racine.



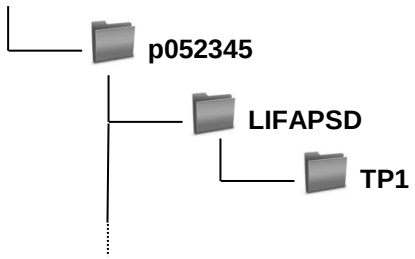
A l'université, les répertoires des utilisateurs ne sont pas stockés physiquement sur les machines des salles de TP, mais sur un serveur auquel les machines de TP accèdent par le réseau (même principe que pour votre lecteur W: sous Windows). Mais cela est transparent pour l'utilisateur, qui accède toujours à son répertoire personnel en allant dans `/home/etu/[n° d'étudiant]`. Cela permet d'avoir accès à ses données même si l'on change de poste de travail (ou d'OS).

Vous pouvez consulter le contenu de votre compte utilisateur de deux façons :

- avec le gestionnaire de fichiers Nautilus (menu Applications / Dossier personnel)

- en ligne de commande :
    - ouvrez un terminal (menu / Emulateur de terminal)
    - vérifiez que vous êtes dans votre répertoire personnel en tapant `pwd` (« print working directory »)
    - demandez le listing du contenu du répertoire en tapant `ls -a`

Dans cette UE, nous allons privilégier l'utilisation de la ligne de commande. Voir l'annexe A pour les commandes Linux de base et des précisions sur la notion de chemin sous Linux. Vous allez créer l'arborescence suivante dans votre répertoire personnel, en n'utilisant que la ligne de commande.



Pour cela, allez dans le terminal, puis :

- vérifiez que vous êtes dans votre répertoire personnel en tapant `pwd`. Si vous n'y êtes pas, retournez-y en tapant `cd` (cela signifie « change directory », et si l'on ne précise pas de répertoire de destination, on va par défaut dans le répertoire personnel)
- créez le répertoire LIFAPSD en tapant la commande suivante (respectez bien l'espace après mkdir, mais n'en mettez pas dans le nom du répertoire) : `mkdir LIFAPSD`
- vérifiez que ce nouveau répertoire apparaît dans le répertoire courant, en tapant `ls`
- allez dans le répertoire créé en tapant `cd LIFAPSD`
- créez le répertoire TP1 en tapant `mkdir TP1`, et déplacez-vous dans ce répertoire

Vérifiez que vous retrouvez bien les dossiers créés en explorant votre dossier personnel en mode graphique.

Exercice 2 : De CodeBlocks aux outils minimaux (éditeur de texte, gcc, terminal)

En LIFAPI, vous avez programmé en C/C++ à l'aide de CodeBlocks, qui est un « environnement de développement intégré » regroupant un éditeur de texte, un compilateur, des outils automatiques de fabrication, et un débogueur. Il en existe d'autres, par exemple Dev-C++. Cependant, programmer dans un environnement plus minimaliste permet d'apprendre à distinguer les éléments essentiels d'un programme et d'une chaîne de compilation. Pour programmer en C++, il faut au minimum : un éditeur de texte, un compilateur en ligne de commande, et un terminal dans lequel on tape les commandes de compilation et d'exécution. C'est ce que nous allons faire dans cette UE. Cela vous permettra, par la suite, de mieux comprendre ce que fait un environnement de développement lorsque vous cliquez sur un bouton « Compiler », par exemple. Cela vous permettra aussi de mieux comprendre les mystérieux fichiers « Makefile » utilisés par ces environnements, car vous allez en faire vous-même (à partir du TP5).

Nous allons utiliser l'éditeur de texte « gedit » (mais vous pouvez aussi utilisez un autre éditeur comme Kate). Pour cela, allez dans le terminal, puis :

- Vérifiez que vous êtes dans le répertoire TP1 en tapant `pwd`. Si vous n'y êtes pas, retournez-y en tapant `cd ~/LIFAPSD/TP1` (~ désigne votre répertoire personnel).
- Lancez gedit en tapant `gedit hello.cpp &`. Comme le fichier hello.cpp n'existe pas encore, gedit le crée pour vous (fichier vide que vous allez pouvoir remplir). Ajouter & en fin de commande vous permet de reprendre la main dans le terminal. Cela va vous permettre en autres de taper les commandes de compilation et d'exécution tout en gardant la fenêtre avec le code ouverte.
- Tapez le code suivant dans gedit (fichier hello.cpp) :

```cpp
#include <iostream>
using namespace std;

int main() {
  cout << "Hello world !" << endl;
  return 0;
}
```

- Sauvegardez vos modifications sans fermer gedit. Retourner dans le terminal (vous aurez peut-être à taper entrée si gedit a envoyé des messages sur la console), et taper `ls` pour vérifier que le nouveau fichier nommé `hello.cpp` est apparu dans le répertoire `TP1`.
- Nous allons maintenant compiler ce code à l'aide du compilateur gcc. Il s'agit du principal compilateur C++ libre. Pour compiler, tapez :

```
g++ -g -Wall -o hello.out hello.cpp
```

- Vérifiez, en tapant `ls`, qu'un nouveau fichier nommé `hello.out` est apparu dans le répertoire `TP1`.
- Tapez `man g++` pour comprendre ce que signifient les différents éléments de la commande de compilation, et complétez le tableau suivant (aide : une fois la documentation affichée, tapez `/mot` pour rechercher un mot, `n` pour passer à l'occurrence suivante, `q` pour sortir).

`g++`	
`-g`	
`-Wall`	
`-o hello.out`	
`hello.cpp`	

`g++`	nom du programme que nous voulons exécuter. Ici, c'est un compilateur qui va traduire notre code C++ en langage machine pour en faire un binaire exécutable.
`-g`	option qui demande à gcc de produire les informations de débugage
`-Wall`	option qui demande à gcc d'afficher tous les « warnings », c'est-à-dire les avertissements indiquant des erreurs possibles dans le code, en plus des erreurs avérées
`-o hello.out`	option qui demande à gcc de nommer l'exécutable « hello.out » (plutôt que « a.out » comme par défaut)
`hello.cpp`	c'est l'argument, il s'agit du fichier source que gcc doit traduire en langage machine

- Exécutez le programme en tapant `./hello.out` dans le terminal.
- Toujours dans le terminal, appuyez plusieurs fois sur la flèche vers le haut. Que se passe-t-il ? A quoi cela peut-il servir ?

Rappel des dernières commandes tapées. Très utile lorsqu'on corrige des erreurs de compilation et que l'on doit donc retaper souvent la commande de compilation.

Exercice 3 : Structures et fonctions globales

Dans cet exercice vous allez définir une structure pour des vecteurs en trois dimensions et des fonctions manipulant ces vecteurs. Dans gedit, ouvrez un nouveau fichier que vous appellerez `Vecteur3D.cpp`. Vous avez à présent deux fichiers ouverts dans gedit : `hello.cpp` et `Vecteur3D.cpp`. Lorsque l'on travaille sur plusieurs fichiers en même temps, il est nettement préférable d'avoir une seule fenêtre gedit et d'utiliser les onglets, plutôt que d'ouvrir plusieurs fenêtres gedit.

a. Dans le fichier `Vecteur3D.cpp`, définisser une structure `Vecteur3D` qui contient trois champs de réels (x, y et z).
b. Puis ajouter les procédures et fonctions suivantes
   - une fonction `Vecteur3DGetNorme` qui retourne la norme d'un vecteur passé en paramètre. Pour rappel la norme (ou taille) d'un vecteur est donnée par la formule : $\sqrt{x^2 + y^2 + z^2}$.
   - une procédure `Vecteur3DNormaliser` qui normalise le vecteur passé en paramètre. Normaliser met à l'échelle le vecteur de telle sorte que sa norme vaut 1.
   - une fonction `Vecteur3DEstNormalise` qui indique si le vecteur passé en paramètre est normalisé.
   - une fonction `Vecteur3DAdd` qui retourne le vecteur somme de deux vecteurs passés en paramètre.
   - une procédure `Vecteur3DAfficher` qui affiche à l'écran le vecteur passé en paramètre sous le format : (x,y,z)
c. Recopier le programme principal suivant en fin de fichier, qui vous permettra de tester vos procédures et fonctions en les commentant/dé-commentant au fur-et-à-mesure.

```
int main () {
   Vecteur3D vecteur1 = {5,2,1};
   Vecteur3D vecteur2 = {0,3,2};

   cout << "vecteur1 non normalise: ";
   Vecteur3DAfficher(vecteur1);
   cout << endl;

   cout << "vecteur2 non normalise: ";
   Vecteur3DAfficher(vecteur2);
   cout << endl;

   cout << "somme: ";
   Vecteur3DAfficher(Vecteur3DAdd(vecteur1,vecteur2));
   cout << endl;

   Vecteur3DNormaliser(vecteur1);
   Vecteur3DNormaliser(vecteur2);

   cout << "vecteur1 normalise: ";
   Vecteur3DAfficher(vecteur1);
   cout << endl;

   cout << "vecteur2 normalise: ";
   Vecteur3DAfficher(vecteur2);
   cout << endl;

   cout << "somme: ";
   Vecteur3D somme = Vecteur3DAdd(vecteur1,vecteur2);
   Vecteur3DAfficher(somme);
   if (Vecteur3DEstNormalise(somme)) cout << " est normalise" << endl;
   else cout << " n'est pas normalise" << endl;
```

```
    return 0;
}
```

d. Compiler et exécuter le programme (nommer l'exécutable `Vecteur3D.out`). Vérifier que vos procédures et fonctions fonctionnent correctement en vérifiant que la trace écran correspond à ce que vous attendez.

```
#include <iostream>
#include <math.h>
using namespace std;

struct Vecteur3D {
    float x,y,z;
};

// Calcule la norme d'un vecteur 3D
float Vecteur3DGetNorme(const Vecteur3D& v) {
    return sqrt(v.x*v.x+v.y*v.y+v.z*v.z);
}

// Normalise le vecteur 3D
void Vecteur3DNormaliser(Vecteur3D& v) {
    float norme = Vecteur3DGetNorme(v);
    if (norme != 0.0) {   // appel possible à !Vecteur3DEstNormalise(v) pour ne pas normaliser
                          // si le vecteur est déjà normalisé
     v.x /= norme; v.y /= norme; v.z /= norme;
    }
}

// Teste si le vecteur est de norme 1
bool Vecteur3DEstNormalise(const Vecteur3D& v) {
    return fabs(Vecteur3DGetNorme(v)-1.0) < 10e-8;
}

// Additionne 2 vecteurs 3D
Vecteur3D Vecteur3DAdd(const Vecteur3D& v1, const Vecteur3D& v2){
    Vecteur3D somme;
    somme.x = v1.x + v2.x;
    somme.y = v1.y + v2.y;
    somme.z = v1.z + v2.z;
    return somme;
}

// Affiche un vecteur 3D sous la forme (x,y,z)
void Vecteur3DAfficher(const Vecteur3D& v) {
    cout << "(" << v.x << "," << v.y << "," << v.z << ")";
}
```

```
Trace écran attendue:

vecteur1 non normalise: (5,2,1)
vecteur2 non normalise: (0,3,2)
somme: (5,5,3)
vecteur1 normalise: (0.912871,0.365148,0.182574)
vecteur2 normalise: (0,0.83205,0.5547)
somme: (0.912871,1.1972,0.737274) n'est pas normalise
```

Exercice 4 : Tableau de structures et paramètre

Toujours dans le fichier `Vecteur3D.cpp`, ajouter les procédures et fonctions suivantes :

a. Une procédure `Vecteur3DRemplirTabVecteurs` qui remplit un tableau déjà alloué en mémoire passé en paramètre avec des vecteurs dont les coordonnées sont des valeurs aléatoires comprises entre -10.0 et 10.0 (avec un chiffre après la virgule). La taille du tableau est aussi passée en paramètre. Consulter l'annexe B pour apprendre comment générer aléatoirement des valeurs.
b. Une procédure `Vecteur3DAfficherTabVecteurs` qui affiche à l'écran l'ensemble des vecteurs contenus dans un tableau de vecteurs passé en paramètre sous le format : vec1 ; vec2 ; ... ; vecn. La taille du tableau est aussi passée en paramètre.
c. Une fonction `Vecteur3DMaxTabVecteurs` qui retourne l'indice du vecteur de plus grande norme dans un tableau de vecteurs passé en paramètre. La taille du tableau est aussi passée en paramètre.

Ecrire le programme principal qui créé un tableau de 10 vecteurs aux valeurs réelles aléatoires, l'affiche, puis affiche le vecteur de plus grande norme. Compiler et exécuter votre programme, et vérifier que vos procédures et fonctions fonctionnent correctement en vérifiant que la trace écran correspond à ce que vous attendez.

```
// Remplit le tableau de vecteurs avec n vecteurs aux valeurs aleatoires [-10,10]
void Vecteur3DRemplirTabVecteurs(Vecteur3D tab[], int taille){
    for (int i=0; i<taille; i++) {
        tab[i].x = ((rand()%201)/10.0)-10.0;
        tab[i].y = ((rand()%201)/10.0)-10.0;
        tab[i].z = ((rand()%201)/10.0)-10.0;
    }
}

// Affiche le tableau de n vecteurs sous la forme vec1 ; vec2 ; ...
void Vecteur3DAfficherTabVecteurs(const Vecteur3D tab[], int taille){
    for (int i=0; i<taille; i++) {
        Vecteur3DAfficher(tab[i]);
        if (i!=taille-1) cout << " ; ";
    }
}

// Retourne l'indice du vecteur de plus grande norme
int Vecteur3DMaxTabVecteurs(const Vecteur3D tab[], int taille){
    if (taille <= 0) return -1;
    float maxNorme = Vecteur3DGetNorme(tab[0]);
    int maxIndice = 0;
    for (int i=1; i<taille; i++) {
        if (Vecteur3DGetNorme(tab[i]) > maxNorme) {
                maxNorme = Vecteur3DGetNorme(tab[i]);
                maxIndice = i;
        }
    }
    return maxIndice;
}

int main() {
  Vecteur3D tab [10];
  srand((unsigned int)time(NULL));
  Vecteur3DRemplirTabVecteurs(tab,10);

  cout << "tab: ";
  Vecteur3DAfficherTabVecteurs(tab,10);
  cout << endl;

  cout << "max tab: ";
  Vecteur3DAfficher(tab[Vecteur3DMaxTabVecteurs(tab,10)]);
  cout << endl;

  return 0;
```

```
}
```

Exercice 5 : Manipulation de tableaux de structures

Toujours dans le fichier `Vecteur3D.cpp`, ajouter les procédures suivantes :

a. Une procédure `Vecteur3DConcatenationTabVecteurs` qui prend trois tableaux de Vecteur3D en paramètre, ainsi que la taille des deux premiers. Cette procédure remplit le troisième tableau déjà alloué en mémoire et de la bonne taille avec les vecteurs contenus dans le premier et le second tableau, dans cet ordre.
b. Une procédure `Vecteur3DInverseTabVecteurs` qui inverse le contenu d'un tableau de Vecteur3D passé en paramètre, avec sa taille.

Ecrire le programme principal qui créé deux tableaux respectivement de 5 et 6 vecteurs aux valeurs réelles aléatoires, les affiche, puis affiche le tableau issu de la concaténation des deux tableaux. Ajouter ensuite les instructions permettant d'inverser le tableau concaténé puis de l'afficher. Compiler et exécuter votre programme, et vérifier que vos procédures fonctionnent correctement en vérifiant que la trace écran correspond à ce que vous attendez.

```
// Concatene les deux premiers tableaux dans le troisième
void Vecteur3DConcatenationTabVecteurs(Vecteur3D tab1 [], int taille1, Vecteur3D tab2 [], int
taille2, Vecteur3D tab3 []) {
    for (int i=0; i<taille1; i++) tab3[i] = tab1[i];
    for (int i=0; i<taille2; i++) tab3[i+taille1] = tab2[i];
}

// Inverse le tableau
void Vecteur3DInverseTabVecteurs(Vecteur3D tab [], int taille) {
    Vecteur3D tmp;
    for (int i=0; i<taille/2; i++) {
        tmp = tab[i];
        tab[i] = tab[taille-i-1];
        tab[taille-i-1] = tmp;
    }
}

int main () {
    Vecteur3D tab1[5]; Vecteur3D tab2[6];
    srand((unsigned int)time(NULL));
    Vecteur3DRemplirTabVecteurs(tab1,5); Vecteur3DRemplirTabVecteurs(tab2,6);
    Vecteur3D tab3[5+6];
    Vecteur3DConcatenationTabVecteurs(tab1,5,tab2,6,tab3);

    cout << "tab1: "; Vecteur3DAfficherTabVecteurs(tab1,5); cout << endl;

    cout << "tab2: "; Vecteur3DAfficherTabVecteurs(tab2,6); cout << endl;

    cout << "tab3: "; Vecteur3DAfficherTabVecteurs(tab3,5+6); cout << endl;

    Vecteur3DInverseTabVecteurs(tab3,5+6);

    cout << "tab3: "; Vecteur3DAfficherTabVecteurs(tab3,5+6); cout << endl;
    return 0 ;
}
```