Université Hassiba Benbouali de Chlef
Faculté des Sciences
Département d'informatique

ASD II
Niveau: Licence 2
2020-2021

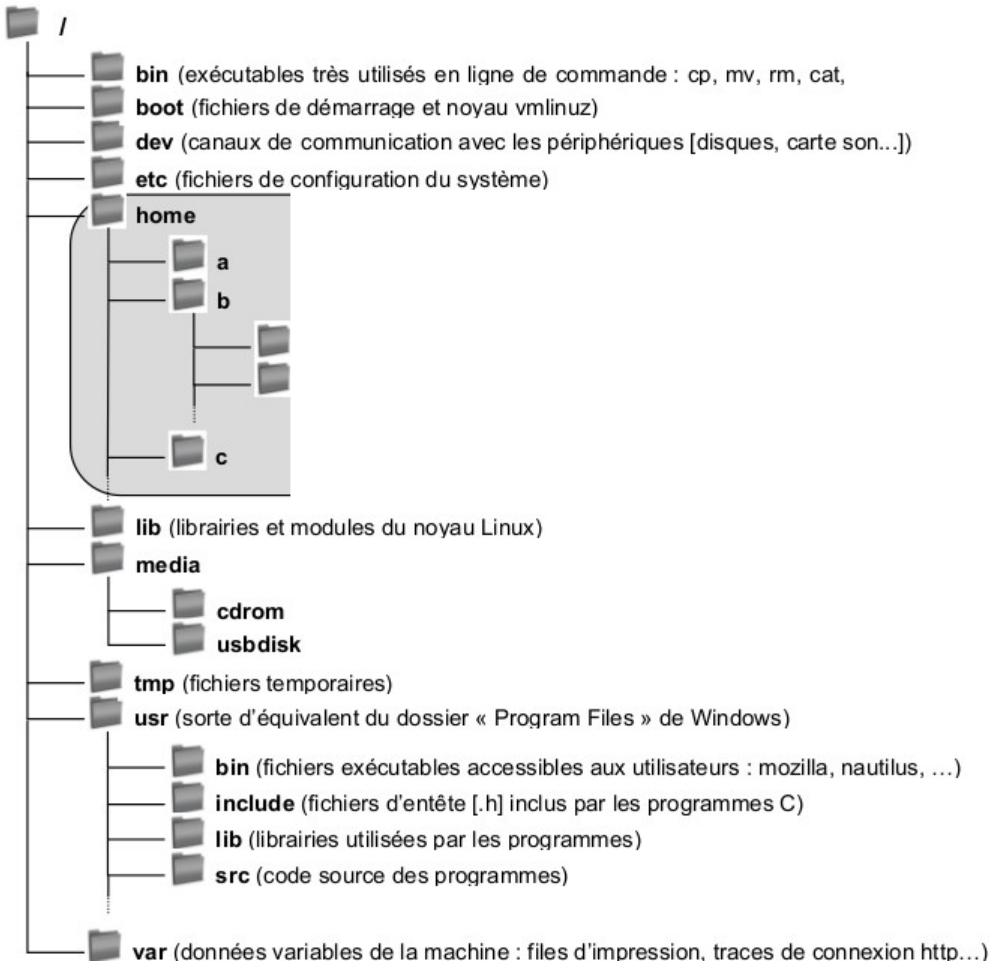
TP1: Première compilation avec gcc sous Linux

Objectifs:

- Découvrir l'environnement Linux.
- Expérimenter avec la ligne de commande.
- Compiler avec la gcc

1. Familiarisation avec le système de fichiers Linux

Redémarrez l'ordinateur sous Linux (Ubuntu), 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.



2. Création d'une arborescence ASD 2 dans votre répertoire personnel

Vous allez créer l'arborescence suivante dans votre répertoire personnel, en n'utilisant que la ligne de commande :

1. Allez dans le terminal, puis vérifiez que vous êtes dans votre répertoire personnel en tapant la commande **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).
2. créez le répertoire **ASD2-TP2018** en tapant la commande suivante (respectez bien l'espace après **mkdir**, mais n'en mettez pas dans le nom du répertoire) : **mkdir ASD2-TP2018.**
3. vérifiez que ce nouveau répertoire apparaît dans le répertoire courant, en tapant **ls**
4. allez dans le répertoire créé en tapant la commande : **cd ASD2-TP2018.**
5. créez le répertoire TP1 en tapant **mkdir TP1**

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

Exercice 1 : Programmer sans environnement de développement

En L1, vous avez programmé en C/C++ à l'aide de **Dev-C++,** 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 **CodeBlocks**. Cependant, il n'est pas indispensable d'utiliser ce genre d'environnement intégré pour programmer : on peut aussi utiliser un éditeur de texte et lancer la compilation et l'exécution en ligne de commande, depuis le terminal. C'est ce que nous allons faire dans ce TP. Cela vous permettra, par la suite, de mieux comprendre ce que fait un environnement de développement lorsque vous cliquez sur « Compiler », par exemple. Cela vous permettra aussi de mieux comprendre les mystérieux fichiers « **Makefile** » utilisés par ces environnements, car vous en aurez fait vous-même (mais nous verrons cela plus tard...).

Nous allons utiliser l'éditeur de texte « **gedit** ». Pour cela, allez dans le terminal, puis:

- Vérifiez que vous êtes dans le répertoire **TP1** en tapant **pwd** .
- Lancez **gedit** en tapant **gedit hello.c &** . Comme le fichier **hello.c** n'existe pas encore, **gedit**
- crée un fichier vide que vous allez pouvoir remplir.
- Tapez le code suivant dans gedit :

```
#include <stdio.h>
int main()
{
printf("Hello world !\n");
return 0;
}
```

Nous allons maintenant compiler ce code à l'aide de gcc. Il s'agit du principal compilateur C libre. Pour compiler, retournez dans le terminal (sans fermer gedit) et tapez :

gcc –g -Wall -ansi -pedantic -o hello.out hello.c

Tapez **man gcc** pour comprendre ce que signifient les différents éléments de cette commande et compléter 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).

gcc	
-g	
-Wall	

-ansi -pedantic	
-o hello.out	
hello.c	

Toujours dans le terminal, appuyez plusieurs fois sur la flèche en haut du clavier. Que se passe-t-il ? A quoi cela peut-il servir ?

Corrigez les erreurs de compilation éventuelles en modifiant votre code source dans gedit, en sauvegardant et en recompilant. Recommencez jusqu'à ne plus avoir aucune erreur. Vous pouvez ensuite exécuter votre programme en tapant dans le terminal : **./hello.out**

Université Hassiba Benbouali de Chlef
Faculté des Sciences
Département d'informatique

ASD II
Niveau: Licence 2
2020-2021

TP2 :

Commencez par créer un répertoire TP1 à l'intérieur de votre répertoire ASD

1. Créez un fichier exo1.c avec gedit.
2. Ecrivez-y l'entête et le code de la procédure suivante :
   *void **remplirTableauAvecEntiersAleatoires**(int * tab, int taille, int valeurMax);*
   Preconditions : tab est un tableau pouvant contenir "taille" entiers.
   Postconditions : tab est rempli avec des nombres entiers aleatoires compris entre 0 (inclus) et ValeurMax (exclue). */
   Aide : en incluant le fichier stdlib.h, on peut utiliser la fonction rand(), qui renvoie un int compris entre 0 et une constante appelée RAND_MAX. Ainsi, la formule **rand()/((double)RAND_MAX + 1)** fournit un double dans [0.0, 1.0[. Ilvous reste à trouver comment, partant de cela, arriver à un int entre 0 et valeurMax.

3. Ecrivez à présent un « main » qui demande à l'utilisateur quelle taille de tableau il souhaite, qui réserve la mémoire pour ce tableau et qui le remplit avec des nombres entiers aléatoires compris entre 0 et 1000000.

4. Ajoutez à votre fichier l'entête et le code d'une procédure de tri par sélection du minimum.

5. Ajoutez à votre fichier l'entête et le code d'une procédure de tri par insertion.

6. Complétez votre « main » pour qu'il demande à l'utilisateur quel algorithme de tri il souhaite utiliser (en tapant par exemple « 0 » pour le tri par sélection, et « 1 » pour le tri par insertion). L'algorithme choisi sera appelé pour trier le tableau de nombres aléatoires.

7. Ajoutez à votre « main » le minutage du temps d'exécution du tri.
   Aide : incluez le fichier time.h et aidez-vous de la page web suivante : http://www.codecogs.com/reference/computing/c/time.h/clock.php (Annexe)

8. Exécutez plusieurs fois votre programme et utilisez un tableur pour tracer les deux courbes du temps d'exécution en fonction de la taille du tableau. Enregistrez votre tableur au format xls.

Annexe: http://www.codecogs.com/reference/computing/c/time.h/clock.php

Interface

```
#include <time.h>
    clock_t   clock (void)
```

Description

The **clock** function determines the amount of processor time used since the invocation of the calling process, measured in *CLOCKS_PER_SEC* of a second.

The code below counts the number of seconds that passed while running some for loop.

Example - Determine processor time used

Workings

```
#include <stdio.h>
#include <time.h>
#include <math.h>
int main()
{
  clock_t start = clock();
  for (long i = 0; i < 100000000; ++i)
    exp(log((double)i));
  clock_t finish = clock();
  printf("It took %d seconds to execute the for loop.\n",
  (finish - start) / CLOCKS_PER_SEC);
  return 0;
}
```

Solution

Output:

```
It took 23 seconds to execute the for loop.
```

Université Hassiba Benbouali de Chlef
Faculté des Sciences
Département d'informatique

ASD II
Niveau: Licence 2
2020-2021

TP3: Liste doublement Chaînée

Exercice 1

Commencez par créer un répertoire TP3 à l'intérieur de votre répertoire ASD2, en utilisant les lignes de commandes vues au TP1 (cd, mkdir, ls...).
L'objectif de ce TP est d'écrire une nouvelle implantation de liste chaînée, différente de celle vue en cours, mais ayant la même interface (mêmes services proposés aux utilisateurs du module). L'implantation que vous allez écrire est celle d'une liste doublement chaînée, dans laquelle:

1. chaque cellule contient un pointeur sur la cellule suivante et un pointeur sur la cellule précédente,
2. la structure Liste contient un pointeur sur la première cellule et un pointeur sur la dernière cellule.

Ainsi, il est possible de parcourir la liste dans les deux sens.

Un autre intérêt de cette implantation est que l'insertion d'un élément en fin de liste (ajoutEnQueue) peut se faire en temps constant, c'est-à-dire en un nombre d'opérations qui ne dépend pas du nombre d'éléments dans la liste. Etait-ce le cas avec l'implantation vue en cours ? Pourquoi ?

Récupérez sur le site de de ASD2 les fichiers Liste.h et Liste.c. Enregistrez les dans votre répertoire Ecrivez-y les #include requis, puis le code de la procédure d'initialisation d'une liste (uniquement cette procédure-ci pour l'instant !). Enregistrez le fichier mais ne le fermez pas, vous y reviendrez plus tard. Créez un nouveau fichier main.c, et écrivez-y un programme principal minimal, qui teste la procédure d'initialisation.

Implantez progressivement les autres fonctions et procédures décrites dans le .h. Attention : vous ne devez PAS écrire toutes les procédures d'un coup. Testez vos sous-programmes au fur et à mesure, en les appelant dans le main. Vérifiez que vous procédures fonctionnent aussi dans les cas particuliers : liste vide, liste ne contenant qu'un seul élément, etc.

Université Hassiba Benbouali de Chlef
Faculté des Sciences
Département d'informatique

ASD II
Niveau: Licence 2
2020-2021

TP4: arbre

1. Récupérez sur le site de ASD2 les fichiers arbre.h et arbre.c. Créez un nouveau fichier main.c et écrivez-y un « main() » vide pour l'instant. Complétez le fichier arbre.c en y ajoutant les définitions des fonctions et procédures suivantes :

- initialiserArbre
- insererElement
- rechercheElement
- hauteur
- testamentArbre

Remarque: Pour certains de ces sous-programmes, vous devrez utiliser une fonction ou procédure auxiliaire travaillant sur un sous-arbre, et prenant donc comme paramètre supplémentaire l'adresse du nœud dans lequel le sous-arbre est enraciné. Ces sous- programmes auxiliaires restent internes au module Arbre : ils n'apparaissent donc pas dans le .h, et sont précédés du mot-clé **static** dans le .c.

Dans le fichier main.c, complétez le main() pour qu'il insère dans un arbre binaire de recherche 255 entiers aléatoires compris entre 1 et 100000, puis qu'il calcule la hauteur maximale de l'arbre et sa profondeur moyenne. Le programme devra bien entendu se terminer proprement, c'est-à-dire en vidant l'arbre. Testez que tout fonctionne bien, en utilisant notamment la procédure d'affichage d'arbre qui vous est fournie. Vérifiez que vous n'obtenez pas le même arbre si vous exécutez deux fois le programme.
Aide : pour le tirage de nombres aléatoires, vous pouvez vous inspirer du petit programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define VALEUR_MAX 100000
int main()
{
int alea1, alea2;
/* Initialisation du générateur, à ne faire qu'une fois
dans le programme. En l'initialisant avec l'heure
à laquelle l'exécution est lancée, on aura des valeurs
différentes à chaque exécution, ce qui est souhaitable ! */
srand((unsigned) time(NULL));
/* Tirage de deux nombres aléatoires compris entre 0 et
VALEUR_MAX incluse */
alea1 = rand()%(VALEUR_MAX + 1);
alea2 = rand()%(VALEUR_MAX + 1);
printf("%d %d\n", alea1, alea2);
return 0;
}
```

3. Complétez le main pour qu'il recherche 100 nombres aléatoires entre 0 et 100000 dans l'arbre (bien évidemment, parmi ces 100 nombres, certains seront effectivement présents dans l'arbre et d'autres non). En plus de la hauteur maximale et de la profondeur moyenne de l'arbre, le main devra afficher le nombre moyen de nœuds visités par opération de recherche.