PR6 – Programmation réseaux
TP n° 2 : Programmation C (révisions)

Partie 1 : Rappels de compilation

Le programme suivant est sauvegardé dans le fichier `hello.c` .

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

Pour créer un fichier exécutable, le plus simple consiste en la commande : `$ gcc hello.c`
Cette commande effectue l'ensemble des phases de la compilation et crée le fichier exécutable `a.out` . Avec l'option `-o` nous pouvons spécifier le nom du fichier exécutable :
`$ gcc -o hello hello.c`
Ici le programme exécutable est `hello` .
Une autre possibilité, utile lors de compilation séparée, est de passer par l'intermédiaire de fichiers objets (`*.o`) :

```
$ gcc -c hello.c
$ gcc -o hello hello.o
```

L'option `-Wall` permet d'afficher tous les messages de warnings. Prenez en compte ces messages afin d'avoir de bonnes habitudes de programmation en C.

```
$ gcc  -Wall -o hello hello.c
```

N'oubliez pas que vous pouvez utiliser `gdb` [1] pour debugger vos programmes et `valgrind` [2] pour tester si vos programes ont des fuites de mémoire.

Partie 2 : Lecture de fichiers et entrée standard

Exercice 1 :

Écrire un programme qui stocke dans un tableau les entiers lus sur l'entrée standard jusqu'à lire la chaîne de caractères `quit`. Attention, on ne sait pas à priori combien d'entiers vont être lus.

1. `https://openclassrooms.com/courses/deboguer-son-programme-avec-gdb`
2. `https://openclassrooms.com/courses/debuguer-facilement-avec-valgrind`

Exercice 2 :

1. Écrire une fonction qui prend en argument un descripteur de fichier ouvert en lecture, et qui vérifie que ce fichier est composé uniquement de chiffres (les caractères `'0'` à `'9'` et pas de caractère de ponctuation).
2. Écrire un programme qui prend 3 arguments `fic`, `alire`, `pivot` sur la ligne de commande, vérifie que `fic` est composé uniquement de chiffres et si c'est le cas, stocke dans une chaîne de caractères `ch` les caractères lus dans le fichier de nom `fic`, en lisant par bloc de `alire` caractères, tant que les derniers caractères lus forment un nombre inférieur au nombre `pivot`. A la fin les derniers caractères lus, leur nombre et la chaîne `ch` sont affichés.

   Par exemple si le contenu du fichier de nom `fic` est `00005000230008700654007` et que l'exécutable du programme se nomme `lire`, alors :

   ```
   $lire fic 5 899
   lus=3, derniers caracteres lus:007
   00005000230008700654007

   $lire fic 5 66
   lus=5, derniers caracteres lus:00087
   000050002300087
   ```

3. Modifier le programme pour qu'il puisse lire dans un fichier ou à partir de `stdin`. Pour lire à partir de `stdin`, on utilise l'option `-I` sur la ligne de commande à la place du nom de fichier.
4. Créer le script bash suivant pour obtenir une séquence de chiffres pseudo-aléatoires en concaténant des valeurs de retour de la fonction interne `RANDOM`, qui retourne un entier dans [0,32767]. Ne pas oublier de le rendre exécutable.

   ```
   #!/bin/bash
   R="";
   for i in {1..5}; do R+=$RANDOM; done;
   echo $R
   ```

5. Appelez votre programme `lire` avec l'option `-I` en redirigeant `stdin` avec un tuyau (pipe) depuis la sortie du script ci-dessus.

Partie 3 : Deviner le mot

Exercice 3 :

1. Ecrire une fonction `int devine(char *mot)` qui prend en paramètre un mot et qui lit un à un les symboles tapés sur la console. Dès que le dernier symbole lu est égal au prochain symbole non-marqué du `mot`, affichez celui-ci sur une nouvelle ligne. La fonction quitte dès que le mot entier est marqué ou lorsque l'utilsateur tape un point (`"."`). Si tout le mot a été marqué, retourner le nombre total de symboles tapés, sinon retourner 0. Par exemple, Si le `mot=="hello"`, et l'utilisateur tape `111h1ell222o`, la fonction retourne 12. Si l'utilisateur tape `111h1el.` la fonction retourne 0.

2. Écrire un programme `devine` qui prend un mot en paramètre sur la ligne de commande et appelle la fonction `devine(mot)`. Si la fonction retourne 0 afficher `Perdu!`, sinon afficher `Gagné en ... coups`.
3. (Optionnel) Par défaut lorsqu'un utilisateur tape un symbole au clavier, le symbole tapé est affiché (echo). Pour modifier ce comportement utiliser la commande `system("stty -echo")` qui permet de désactiver l'affichage des caractères tapés au clavier. Pour le remettre dans son état initial la commande est `system("stty echo")`. (Le comportement peut dépendre du type de terminal que vous utilisez, donc à utiliser avec précaution.)

   Modifier votre programme pour que le terminal affiche uniquement les symboles lus qui font partie du mot donné en entrée.

Partie 4 : Big-endian et Little-endian

Dans une architecture Big-endian, les octets sont conventionnellement numérotés de la gauche vers la droite. Dans le mode Big-endian les octets de poids fort sont placés en tête et occupent donc des emplacements mémoire avec des adresses plus petites. Dans une architecture Little-endian, c'est le contraire. Voici un exemple pour l'entier 1 sauvegardé en mémoire dans les architecture Big-endian et Little-endian.

	Adresses			
	A	A+1	A+2	A+2
Little-endian	0x01	0x00	0x00	0x00
Big-endian	0x00	0x00	0x00	0x01

Le protocole IP définit un standard, le network byte order (soit ordre des octets du réseau). Dans ce protocole, les informations binaires sont en général codées en paquets, et envoyées sur le réseau, l'octet de poids le plus fort en premier, c'est-à-dire selon le mode Big-endian et cela quel que soit l'endianness naturel du processeur hôte. Les fonctions suivantes qui permettent de convertir la représentation d'un entier sur la machine hôte en la représentation standard utilisée sur le réseau et vice-versa :

```
uint32_t htonl(uint32_t hostlong);    uint32_t ntohl(uint32_t netlong);
uint16_t htons(uint16_t hostshort);   uint16_t ntohs(uint16_t netshort);
```

Exercice 4 :

Écrire un programme qui teste si votre machine fonctionne en Big-endian ou Little-endian. Une solution est de vérifier l'octet de poids faible d'un entier donné. Testez votre programme sur `nivose` et `lucien`.

Exercice 5 :

Écrire un programme qui lit un entier (en 32 bit) et qui affiche sa représentation hexadécimale en Big-endian et Little-endian. Pour cela, il faut d'abord tester si la machine que vous utilisez fonctionne en Big-endian ou Little-endian. Testez votre programme sur `nivose` et `lucien`.

Exercice 6 :

Écrire un programme qui lit un entier (en 32 bit) et qui affiche sa représentation hexadécimale sur la machine hôte et sur le standard utilisé sur le réseau. Pour cela, vous pouvez utiliser les fonctions `htonl`, `htons`, `ntohl` et `ntohs`. Testez votre programme sur `nivose` et `lucien`. Quelle différence remarquez-vous ?

Partie 5 : Rappels sur l'arithmétique des pointeurs

Exercice 7 :

Soit `P` un pointeur qui 'pointe' sur un tableau `A` :

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions :

1. `P`
2. `A`
3. `&A`
4. `&P`
5. `*P+2`
6. `*(P+2)`
7. `&P+1`
8. `P+1`
9. `&A[4]-3`
10. `A+3`
11. `P+(*P-10)`
12. `*(P+*(P+8)-A[7])`

Si on rajoute l'instruction `++P;` à la fin du code précédent, quelle adresse est fournie par `P` ? Si on rajoute l'instruction `++A;`, qu'est-ce qu'il se passe ?