TP n°2

Programmation C (révisions)

Partie 1 : compilation et fonctions

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

Ce programme est sauvegardé dans le fichier `hello.c`. 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. Il est recommandé de prendre en compte ces messages afin d'avoir les bonnes habitudes de programmation en C.

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

Exercice 1

Écrire un programme qui saisit 2 entiers et affiche successivement la somme, la différence, le produit, les quotients de la division réelle et euclidienne et le reste de la division euclidienne de ces 2 entiers.

Exercice 2

Écrire deux programmes retournant la factorielle, calculée respectivement de façon itérative et récursive, d'un nombre entier entré par l'utilisateur.

Partie 2 : fonction, allocation de mémoire et pointeur

Exercice 3

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

Exercice 4

- Écrire une fonction `int toInt(char* s)` qui convertit la chaîne de caractères ascii `s` en un entier. On considère que `s` représente un nombre entier positif.
- Ensuite écrire un programme demandant à l'utilisateur de saisir au clavier un entier positif d'au plus 6 chiffres et qui le traduira en appelant la fonction `toInt`. Ce programme devra utiliser un pointeur vers `char` et une allocation dynamique pour stocker la chaîne de caractères.

Partie 3 : Big-endian et little-endian

Dans une architecture big-endian, les octets sont conventionnellement numérotés de la gauche vers a 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.

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.

Exercice 5

Écrire un programme qui teste si votre machine fonction en Big-endian ou Little-endian. Une solution est de vérifier l'octet de poids faible d'un entier donné.

Exercice 6

Écrire un programme qui lit un entier (en 32 bit) et qui affiche sa représentation hexadécimale en big endian et little endian. Pour cela utilisez une des fonctions vue en cours.