Architecture des ordinateurs

TP 1: Représentation des nombres entiers et flottants

1. Représentation binaire d'un entier positif

   Durant cet exercice vous utiliserez le fichier *basemain.c* pour tester vos fonctions. Complétez les fonctions du fichier *bases.c* :
   - Complétez la fonction **base2** afin de donner la représentation binaire sur 8 bits d'un entier compris entre 0 et 255. Pour cela, vous pourrez utiliser la division entière et les modulos. Soit a et b deux entiers Le calcul a/b est appelé *division entière* de a par b et son résultat est la partie entière inférieure de la valeur réelle $\frac{a}{b}$. En C, a modulo b (le reste de la division entière de a par b) s'écrit $a\%b$.
   - Complétez la fonction **base8** afin de donner la représentation en base 8 (octal) sur 8 bits d'un entier compris entre 0 et 255.
   - Complétez la fonction **base16** afin de donner la représentation en base 16 (hexadecimal) sur 8 bits d'un entier compris entre 0 et 255. Utilisez la fonction **d2c** qui donne, à partir d'un nombre entre 0 et 36, le code ASCII du caractère le représentant : (10→A, 11→B, ..., 35→Z).
   - Complétez la fonction **basen** afin de donner la représentation en base n sur 8 bits d'un entier compris entre 0 et 255. De même, utilisez la fonction **d2c**.

   Fichier : basemain.c

```c
#include "bases.c"

int main(int argc, char** argv)
{
    int entier;
    int base;

    printf("Entrer un nombre compris entre 0 et 255 : ");
    scanf("%d",&entier);

    printf("Base 2 : ");
    base2(entier);
    printf("\n");

    printf("Base 8 : ");
    base8(entier);
    printf("\n");

    printf("Base 16 : ");
    base16(entier);
    printf("\n");

    printf("Entrer une base comprise entre 1 et 36 : ");
    scanf("%d",&base);
    printf("Base %d : ",base);
    basen(entier, base);
    printf("\n");

    return 0;
}
```

Fichier : bases.c

```c
#include <stdio.h>      // EntrÃl'e / Sortie standard
#include <stdint.h>     // Entiers standard


int d2c (int n)
{
    if(0<=n && n<10)
        return '0'+n;
    else    if (n < 36)
        return 'A'+ (n-10);
    else return '?';
}


void base2(uint8_t entier)
{

}

void base8(uint8_t entier)
{

}

void base16(uint8_t entier)
{

}

void basen(uint8_t entier, uint8_t base)
{

}
```

2. Valeur maximale

   Ecrire un programme qui donne la valeur maximale d'un entier. Pour cela, on part d'un entier de valeur 1 et on le multiplie par 2 jusqu'à atteindre un dépassement de capacité, c'est-à-dire jusqu'à revenir à 0. Si pour cela, le nombre a été multiplié n fois, la valeur maximale est $2^n - 1$. Tester ce programme avec un entier signé (type `int` en C) puis avec un entier non signé (type `unsigned int`). Vous utiliserez la fonction **puiss** suivante (Pourquoi utilise t'on un retour de fonction en **long** ?) :

   Fichier : limit.c

```c
#include <stdio.h>

/* Calcul x^n : */
long puiss(int x,int n) { return n==0?1:puiss(x,n-1)*x; }

int main(int argc, char** argv)
{

}
```

3. Boutisme

   Rappel : on parle de petit boutiste ou little endian (resp. grand boutiste ou big endian) lorsque les octets constituant un mot sont stockés en mémoire du plus faible au plus fort (resp. du plus fort au plus faible).

   Écrire un programme C détectant si la machine utilisée est de type *petit* ou *grand boutiste* (*little* ou *big endian*). Utilisez le code suivant qui définit un type entier non signé sur 32 bits (`uint32_t`) et la fonction

`make_uint32_t`. Cette fonction prend quatre octets en paramètres et construit un entier de type `uint32_t` en stockant les octets dans l'ordre en mémoire. Complétez le corps des fonctions suivantes :

- Complétez la fonction **print_LE** afin de donner la représentation little endian d'une zone mémoire pointée par **ptr** de **size** octets.
- Complétez la fonction **print_BE** afin de donner la représentation big endian d'une zone mémoire pointée par **ptr** de **size** octets.
- Complétez la fonction **endian** renvoyant la bonne fonction d'affichage selon l'architecture de la machine.
- Note : Pour les fonctions **print_LE** et **print_BE**, utilisez la fonction **base2** définie dans la partie précédente.

Fichier : boutisme.c

```c
#include "bases.c"
#include <stdlib.h>


/* Pointeur de fonction d'affichage d'un entier d'une certaine taille */
typedef void (*endian_print_t)(void* ptr, size_t size);

/* construit un uint32 Ãă partir de quatre octets */
uint32_t make_uint32_t (uint8_t o0, uint8_t o1, uint8_t o2, uint8_t o3)
{
    union {uint32_t i; uint8_t o[4];} v;
    v.o[0] = o0; v.o[1] = o1; v.o[2] = o2; v.o[3] = o3;
    return v.i;
}

/* Affichage pour petit boutisme/little endian : */
void print_LE(void *ptr, size_t size)
{

}

/* Affichage pour grand boutisme/big endian : */
void print_BE(void *ptr, size_t size)
{

}

/* Choix de la bonne fonction d'affichage : */
endian_print_t endian()
{

}

/* Main : */
int main(int argc, char** argv)
{
    endian_print_t fun=endian();

    uint8_t a = 255;
    printf(" a = %d ", a);
    fun(&a, sizeof(a));

    uint32_t b = 0XF0F0F0F0;
    printf(" b = %X ", b);
    fun(&b, sizeof(b));

    float c = 333.333;
    printf(" c = %f ", c);
```

```
    fun(&c, sizeof(c));

    float d = -333.333;
    printf(" d = %f ", d);
    fun(&d, sizeof(d));

    return 0;
}
```

4. Flottants

   Écrire un programme C décortiquant une nombre flottant de type `float`. Ces nombres sont codés sur 32 bits suivant la norme IEEE 754 (single : 1 bit de signe, 8 pour l'exposant et 23 pour la mantisse). Utilisez la fonction suivante qui retourne le nième bit ($0 \leq n \leq 31$) d'un `float`.

   Fichier : flottant.c

```
#include <stdio.h>
#include <stdint.h>

/* retourne le n-iÃÍme (0 <= n <= 31) bit d'un float */
int nieme_bit(float f, int n)
{
    union {uint32_t i; float f;} v;
    v.f = f;
    return (v.i >> n) & 0x1;
}
```