

Architecture des ordinateurs et systèmes d'exploitation

Corrigé du TP 2: Boutisme et flottants

Marcel Bosc

Christophe Dehlinger
Benoît Meister

Arnaud Giersch
Nicolas Passat

Mathieu Haefele

1. Boutisme

Écrire un programme C détectant si la machine utilisée est de type *petit* ou *grand boutiste* (*little* ou *big endian*). 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). On pourra se servir du code suivant qui définit un type entier non signé sur 32 bits (`uint32`) et la fonction `make_uint32`. Cette fonction prend quatre octets en paramètres et construit un entier de type `uint32` en stockant les octets dans l'ordre en mémoire.

```
/* type entier non signé sur 32 bits */
typedef unsigned int uint32;

/* construit un uint32 à partir de quatre octets */
uint32 make_uint32 (unsigned char o0, unsigned char o1,
                  unsigned char o2, unsigned char o3)
{
    union { uint32 i; unsigned char o [4]; } v;
    v.o [0] = o0; v.o [1] = o1; v.o [2] = o2; v.o [3] = o3;
    return v.i;
}
```

Correction :

```
#include <stdio.h>
#include "make_uint32.c"

int main (void)
{
    uint32 i;

    i = make_uint32 (0x11, 0x22, 0x33, 0x44);
    printf ("octets: 0x11, 0x22, 0x33, 0x44\n");
    printf ("entier: 0x%08x\n", i);
    if (i == 0x11223344)
        printf ("=> Big Endian\n");
    else if (i == 0x44332211)
        printf ("=> Little Endian\n");
    else
        printf ("=> Unknown Endianness\n");

    return 0;
}
```

2. Flottants

Écrire un programme C décortiquant un 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).

On pourra se servir de la fonction suivante qui retourne le nième bit ($0 \leq n \leq 31$) d'un float.

```
/* type entier non signé sur 32 bits */
typedef unsigned int uint32;

/* retourne le n-ième (0 <= n <= 31) bit d'un float */
int nieme_bit ( float f, int n)
{
    union {uint32 i; float f;} v;
    v.f = f;
    return (v.i >> n) & 0x1;
}
```

On pourra trouver des informations sur codage des flottants suivant la norme IEEE 754 aux URL suivantes :

- <http://perso.club-internet.fr/ariffart/boutils/boutils05.html>
- <http://research.microsoft.com/~hollasch/cgindex/coding/ieeefloat.html>

Correction :

```
#include <stdio.h>
#include "nieme_bit.c"

/* définit une NaN */
union {uint32 i; float f;} unan = {0x7fc00000UL};
#define NAN (unan.f)

int main (void)
{
    float f, mantisse, ma;
    int i, signe, exposant;

    /* on demande un float */
    printf ("f? ");
    if (scanf ("%f", &f) != 1)
        f = NAN;
    printf ("printf(f) : %g\n", f);

    /* extraction du signe , ... */
    signe = nieme_bit (f, 31);
    printf ("signe ....: %d (%s)\n", signe, signe? "-": "+");

    /* ..., de l'exposant , ... */
    printf ("exposant .: ");
    exposant = 0;
    for (i = 30; i >= 23; i--) {
        printf ("%d", nieme_bit (f, i));
        exposant = 2 * exposant + nieme_bit (f, i);
    }
    /* l'exposant est biaisé de 127 */
    exposant = exposant - 127;
    printf (" (%d)\n", exposant);

    /* ... et de la mantisse */
    printf ("mantisse .: ");
    /* si l'exposant vaut -127 (que des 0), la mantisse est dénormalisée */
    if (exposant == -127) {
        mantisse = 0.0;
        ma = 2.0;
    } else {
        mantisse = 1.0;
    }
}
```

```

    ma = 1.0;
}
for ( i = 22; i >= 0; i--){
    printf ( "%d", nieme_bit ( f , i ));
    ma = ma / 2.0;
    if ( nieme_bit ( f , i ))
        mantisse = mantisse + ma;
}
printf ( " (%g)\n", mantisse);

/* affichage du nombre décortiqué */
printf ( "=> ");
if ( exposant == 128) {
    /* exposant à 128 (que des 1): cas particulier ( inf ou nan)*/
    if ( mantisse == 1.0)
        printf ( "%sinf\n", signe? "-": "+");
    else
        printf ( "nan\n");
} else {
    printf ( "%s%g",
            signe? "-": "+", mantisse);
    if ( mantisse != 0.0 || exposant != -127)
        printf ( " x 2^%d", exposant);
    printf ( "\n");
}

return 0;
}

```