info803 : informatique

TP 2 : opérations bit à bit et représentation des nombres flottants

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22
téléphone : 04 79 75 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/

Le compte-rendu de TP sera contenu en commentaire dans les sources de votre programme. Vous devrez donc me rendre un unique fichier appelé "`nom.c`*". Votre fichier devra être le plus lisible possible (indentation intelligente de votre code, présentation lisible de vos commentaires).*

Exercice 1 : multiplication de matrices

Programmez au mieux l'algorithme vu en cours qui permet de minimiser le nombre de multiplications scalaires pour le calcul du produit d'une chaîne de matrices.

Spécifiez le problème et les limites de votre algorithme. Commencez par une version assez simple avant de le complexifier pour obtenir un algorithme le plus général possible.

Exercice 1-bis : faire de la monnaie

Si vous préferrez, vous pouvez remplacer le problème de la multiplication des matrices par le problème de la monnaie...

Exercice 2 : opérations bit à bit

En plus des opérations booléennes usuelles, C permet de manipuler les entiers (et autres types associés) comme suites de bits. Les opérations disponibles sont

- le "ou" bit à bit : `|`
- le "et" bit à bit : `&`
- le "ou exclusif" bit à bit : `^`
- le décalage à droite : `>>`
- le décalage à gauche : `<<`

Je vous renvoie au chapitre 7.2 du polycopié de Bernard Cassagne pour les détails.

Pour la suite, on définit le type `octet` comme un synonyme pour le type `char`. (Un `char` est codé sur 8 bit.)

```
typedef unsigned char octet ;
```

(On utilise `unsigned` pour être sûr que les bits rajoutés par un décalage à droite sont bien des `0` ; voir la remarque à ce sujet dans le poly de Bernard Cassagne.)

Question 1 : écrivez une fonction qui permet d'afficher un octet. Servez-vous en pour afficher le code ASCII binaire de quelques caractères.

Question 2 : un entier de type `int` est codé sur 4 octets. Pour vérifier ça, il suffit d'appeler la fonction `sizeof()` sur un objet de type `int` ; ou directement sur le type `int`. À cause de la sémantique des pointeurs, un pointeur vers un entier peut donc être vu comme un tableau de 4 octets.

```
int n ;
octet *N ;
...
N = (octet*) n ;
...
N[0]...
```

Utilisez ceci pour afficher, dans l'ordre, les octets composants un nombre entier.
Est-ce que les bits de poids forts sont au début ou à la fin du nombre ?

Question 3 : écrivez une fonction qui affiche la représentation d'un entier comme suite de bits, avec les bits de poids fort à gauche. Est-ce que cette fonction fonctionnera sur tous les ordinateurs ?

Question 4 : écrivez un petit programme qui vous permette de modifier la valeur d'un entier en changeant ses bits un par un. Votre fonction declarera un tableau de 4 octets, que vous pourrez modifier comme bon vous semble, puis afficher comme entier signé ou non signé.
Comment sont représentés les entiers, signés ou non en C ?

Exercice 3 : représentation des flottant, norme IEEE7754

Question 1 : écrivez une fonction pour afficher la représentation d'un flottant.

Pour convertir un flottant en tableau de bits, le plus simple est de faire une copie de mémoire bas niveau :

```
#include <strings.h>
...
float x ;
octet X[sizeof(x)] ;
...
memcpy(X,&x,sizeof(x)) ;    /* copie x dans X */
...
memcpy(&x,X,sizeof(x)) ;    /* copie X dans x */
```

Question 2 : adaptez votre programme de l'exercice 2 pour explorer la représentation des flottants.

Question 3 : essayez de créer les nombres suivants
- -0
- 0.1, pouvez-vous le représenter de manière exacte ?
- quelques nombres dénormalisés
- le plus petit nombre strictement positif représentable
- le plus grand nombre strictement positif représentable
- $+\infty$ ($\mathtt{0\,11\cdots11\,00\cdots00}$)
- $-\infty$
- plusieurs "Not a Number"
- $+\infty * -\infty$
- $1/0$
- ∞/∞
- $\infty * 0$
- $0/0$
- $\sqrt{-1}$, $\sqrt{-2}$
- ...

Comparez ces nombres avec les opérations usuelles (inférieur, supérieur, égalité).

Question 4 : adaptez votre programme pour qu'il fonctionne avec des flottants "double precision". (Type `double` en C.)
Combien de bits forment la mantisse ? Combien pour l'exposant ?