Architecture des ordinateurs : types de données et représentation en mémoire

Instructions

Réalisation des exercices :
- Les programmes doivent être réalisés en C.
- Privilégier un style clair et lisible
- Nommer les fonctions et les variables avec des noms appropriés
- Commentez votre code

Remise des exercices :
- Rendre une archive .tar.gz (ou zip) contenant d'un répertoire nommé TP2.
- Nommer l'archive par vos NOM, PRENOM et NUMERO (sur le modèle NOM-PRENOM-NUMERO.tar.gz).
- Le répertoire TP2 de l'archive contient les programmes réalisés pendant votre TP
- Chaque fichier doit être nommé avec le numéro de l'exercice (ex : `"exo1_1"`).
- Évitez les sous-dossiers
- Placez toutes les réponses aux questions dans un fichier "reponses.txt". Avant chaque réponse, faites un copier/coller des commandes saisies en console pour compiler vos programmes et les lancer ainsi que des résultats obtenus.

Déposer l'archive sur : https ://moodle.univ-paris8.fr/moodle/course/view.php ?id=1745 code 'ARCHI'

1 Problème de débordement

En C, le type `char` est représenté par un nombre limité de bits. Si l'on tente de représenter une valeur qui dépasse la capacité définie par le nombre de bits de la représentation, on risque un débordement. Vous allez utiliser la fonction suivante pour constater le problème de débordement :

```
void overflow_char() {
    char i = 0;
    while (i < (char) (i+1)) {
        i++;
    }
    printf("plus grand char positif : %d\n", i);
    printf("plus grand char négatif : %d\n", (char) (i+1));
}
```

La fonction `printf` permet d'afficher les valeurs en décimal avec `%d` (utilisé aussi pour afficher les `int` en décimal).

Dans un fichier texte `reponses.txt` placez la réponse à la question suivante : Que fait cette fonction, comment va-t-elle permettre de constater le débordement ? Expliquez en détails.

1.1 Test du type char

La fonction C précédente est prête à être utilisée dans le fichier `exo1_1.c`. Récupérez ce fichier sur moodle, compilez-le et lancez le programme. Dans le fichier texte `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console (pour compiler et lancer le programme) et du résultat obtenu. Répondez ensuite aux questions :

- A votre avis, pourquoi je dois préciser que `(i+1)` est un char en plaçant `(char)` (un "cast explicite") devant ?
- Quelle est la valeur maximale représentée par un char ?
- Quelle est la valeur minimale représentée par un char ?
- Puisqu'un char signé est représenté en complément à 2, d'après la valeur maximale obtenue, sur combien de bits est représenté un char ?

1.2 Test du type int

Faites une copie du premier programme sous le nom `exo1_2.c` et modifiez la fonction pour faire le même test avec un `int` (attention, le temps de calcul sera plus long). Dans le fichier `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu. Répondez ensuite aux questions :

- Quelle est la valeur maximale représentée par un int ?
- Quelle est la valeur minimale représentée par un int ?
- Puisqu'un int signé est représenté en complément à 2, sur combien d'octets (groupes de 8 bits) est représenté un int ? (vous pouvez vous aider de la calculatrice en mode programmeur pour le déterminer)

1.3 La fonction sizeof()

Vérifiez les résultats obtenus précédemment en affichant les valeurs retournées par la fonction sizeof() du langage C. Cette fonction affiche le nombre d'octets d'un type de donnée, il faut multiplier par 8 pour avoir le nombre de bits (un octet contient 8 bits).

exemple d'utilisation :

```
printf("taille d'un int : %lu bits\n", sizeof(int) * 8);
```

Ici, `%lu` indique que la taille retournée par sizeof() est un entier long (**l**ong), non signé (**u**nsigned), c'est à dire un entier de grande taille et positif.

Créez un autre fichier C nommé `exo1_3.c`. Dans le main de votre programme utilisez plusieurs fois sizeof et affichez les tailles des types suivants avec printf : char, int, float, double, short, long, long double, unsigned int, unsigned char, unsigned short, unsigned long.

Dans le fichier "reponses.txt", faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu. Répondez ensuite aux questions (une recherche sur internet peut être nécessaire si vous ne connaissez pas la réponse) :

- Le résultat obtenu correspond-il à ce que vous aviez calculé ?
- Quels sont les types prévus pour stocker des nombres entiers ?
- Quels sont les types prévus pour stocker des nombres flottants (à virgule) ?
- Quels sont les types prévus pour stocker des nombres non signés ?
- Quels sont les types prévus pour stocker des nombres signés ?

2 Représentation des données en mémoire

La fonction suivante reçoit un pointeur sur une variable et sa taille en octets (paquets de 8 bits) et elle affiche une représentation des bits codant le contenu de cette variable.

```
void mem2bin(void * p, unsigned int size) {
    char mask;
    int i, j;
    for (i = 0; i < size; i++) {
        for (j = 7; j >= 0; j--) {
            mask = 1 << j;
            if (*(char*)(p + i) & mask) printf("1");
            else  printf("0");
        }
        printf(" ");
    }
    printf("\n");
}
```

Voici une explication du fonctionnement de cette fonction :

1. La boucle "`for (i = 0; i < size; i++)`" va examiner chaque octet tour à tour.

2. Pour chaque octet, la boucle "`for (j = 7; j >= 0; j-)`" permet d'examiner chaque bit.
3. Dans une variable `mask` on place un bit à 1 et avec l'opérateur de décalage `«`, on décale le bit pour le placer à la position voulue. Pour lire le bit le plus à gauche dans l'octet courant, il faudra décaler de 7 positions. Pour lire le bit le plus à droite, il faudra décaler de 0 positions. C'est pour cette raison que j prend des valeurs de 7 à 0.
4. L'opérateur `&` utilisé dans l'expression `(p + i) & mask` est un ET logique binaire. Il permet d'isoler le bit qui est à la position indiquée par le masque. Si le résultat est supérieur à 0 cela signifie que le bit isolé était à 1 et le test if est donc concluant.

2.1 Ordre des octets

La fonction C précédente est prête à être utilisée dans le fichier `exo2.c`. Récupérez ce fichier sur moodle. Vous constaterez que la fonction est lancée avec une variable de type `int` valant 252117761. Le `&` collé au nom de la variable, permet de récupérer son adresse (un pointeur). La fonction `sizeof()` permet de récupérer la taille en octets de la variable. Compilez et lancez le programme.

Dans le fichier `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu. Répondez ensuite aux questions :

- A l'aide d'une calculatrice en mode programmeur, convertissez 252117761 en binaire. Comparez le résultat avec ce que votre programme a affiché. Que remarquez vous ?
- Faites une recherche avec le mot "Endianness" sur internet. Grâce à ces informations comment expliquez vous la différence entre le résultat obtenu par votre programme et la représentation binaire du nombre indiquée par la calculatrice ?

2.2 Affichage des octets de gauche à droite

Faites une copie de la fonction précédente (en lui donnant un autre nom) et modifiez la boucle "for (i = 0 ; i < size ; i++)" pour examiner l'octet le plus à droite en premier (il suffit de faire une boucle inverse de `size-1` à `0`). Appelez cette nouvelle fonction dans le main (en conservant l'appel à la fonction précédente pour comparer leur résultat). Si votre fonction est réussie, elle doit maintenant afficher le même résultat que la calculatrice.

Dans le fichier `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu.

2.3 Afficher la représentation d'un nombre signé

Calculez à la main la représentation en complément à 2 du nombre -211 sur 4 octets. Mettez les détails de votre calcul dans le fichier `reponses.txt`.

Toujours dans le fichier `exo2.c`, dans le main de votre programme, ajoutez un nouevl appel à votre fonction `mem2bin` modifiée pour vérifier votre calcul.

Dans le fichier `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu.

2.4 Afficher la représentation d'un nombre flottant

Toujours dans le fichier `exo2.c`, dans le main de votre programme, créez un flottant initialisé à -6.843750 (exercice de conversion donné en cours). Affichez sa représentation binaire avec `mem2bin`. Vérifiez que cela correspond à ce que vous aviez calculé

Dans le fichier `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu.

3 Placer la représentation d'un nombre en mémoire

Nous allons maintenant utiliser une fonction qui fait l'inverse : elle prend une chaine de caractère représentant un nombre en binaire (`char bin[]`) et un pointeur sur une zone mémoire (`*p`) ainsi que la taille des données à écrire (`unsigned int size`). Elle va ensuite modifier les bits correspondant en mémoire pour y placer la représentation du nombre. Quand la fonction se termine, le pointeur `p`, pointe donc vers la représentation du nombre en binaire .

```
void bin2mem(char bin[], void * p, unsigned int size){
    int i, j;
    char mask;
```

```
    for (i = 0; i < size; i++) {
        for (j = 0; j < 8; j++) {
            mask = 1 << (7 - j);
            if (bin[i*8 + j] == '1') {
                *(char*)(p + size-1 - i) |= mask; // 1
            } else {
                *(char*)(p + size-1 - i) &= ~mask; // 0
            }
        }
    }
}
```

Le fichier C `exo3.c` contient cette fonction prête à être utilisée. La variable `repr` contient la représentation du nombre sous forme de chaine de caractère. Une variable de type `int` nommée resultat est prête à accueillir les données. L'appel à la fonction `bin2mem` permet de placer la représentation binaire dans la variable résultat. L'appel à printf permet d'afficher le résultat. Compilez et lancez le programme pour le tester.

3.1 Test sur des entiers signés

Dans le main de du programme `exo3.c`, créez une nouvelle variable contenant la représentation du nombre 211 en binaire (calculé précédemment) et une nouvelles variable pour accueillir le résultat. Appelez la fonction `bin2mem` avec ces deux variables. Affichez ensuite avec `printf` le contenu de la variable résultat (avec %d).

Faites un troisième appel à la fonction avec -211 représenté en binaire en complément à 2 et affichez une fois de plus le résultat avec `printf`. Compilez et lancez le programme.

Dans le fichier texte "reponses.txt", faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu. Vérifiez que les valeurs affichées par le programme sont les bonnes.

Ajoutez un appel à la fonction dans votre programme pour répondre à la question suivante :

- Quel est la valeur en décimal du nombre entier représenté par `"11000111111101101000010010000001"` en complément à 2 sur 4 octets ?

3.2 Test sur des flottants en simple précision

Dans le main de votre programme ajoutez maintenant une variable **de type float**. Appelez la fonction `bin2mem` avec la représentation binaire 01010000100110111000000000000000 dans la norme IEEE754 en simple précision (exercice de conversion déjà réalisé). Affichez le contenu de la variable après cet appel et vérifiez que cela correspond à ce que vous aviez calculé.

Ensuite, indiquez ou calculez (à la main) les représentations des valeurs binaires spéciales +Inf, -Inf et NaN, du nombre 0 et un nombre dénormalisé de votre choix avec la norme IEEE754 en simple précision. Indiquez les détails dans le fichier `reponses.txt`.

Dans votre programme ajoutez plusieurs appels à la fonction `bin2mem` (pour chaque représentation) pour placer successivement chaque représentation dans un float que vous afficherez ensuite avec printf (avec `%e` pour la représentation scientifique). Vérifiez que vous obtenez bien les valeurs prévues.

Dans le fichier `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu.

3.3 Comprendre qu'une représentation est une convention...

Dans le fichier `exo3_3.c` il y a les éléments suivants :

- la déclaration une variable de type `char` initialisée à `'A'`.
- un appel à printf pour afficher cette variable successivement avec `%c` (caractère), `%d` (decimal) et `%x` (hexadécimal).
- la déclaration un tableau de 4 caractères, un entier et un flottant.
- trois appels à la fonction `bin2mem` pour mettre la représentation `"00000000010000110100001001000001"` dans chacune des variables.
- l'affichage de la chaîne avec `%s`, de l'entier avec `%d` (decimal) puis `%x` (hexadécimal) et du flottant avec `%f` (flottant) et `%e` (notation scientifique).

Compilez et lancez le programme pour le tester.

Dans le fichier `reponses.txt`, faites un copier/coller de toutes les commandes saisies en console et du résultat obtenu. Examinez ce résultat et expliquez le succinctement. Répondez alors aux questions suivantes :

- Si on trouve les bits `01101101` dans la mémoire d'un ordinateur que représentent ils ?
- Si on trouve les bits `01000100010000110100001001000001` dans la mémoire d'un ordinateur que représentent ils ?