TD 3 : Subtilités syntaxiques et traitement de bits

Programmation en C (LC4)

Semaine du 20 février 2012

1 Effets de bord

Certaines expressions ne sont pas seulement évaluées en une valeur, mais peuvent avoir un effet sur des variables du programme. Pour utiliser correctement de tels *effets de bord*, il faut savoir quelle est la modification apportée à l'état de la mémoire par une expression donnée.

Question 1 Si `x` et `y` sont des variables contenant les entiers 3 et 5 respectivement, quelle est la valeur de `x`, et celle de `y`, après évaluation des expressions suivantes :

- `x += x++;`
- `x = ++y * x--;`
- `x += y * --x;`
- `y = ++x + --x;`
- `y = ++x + x--;`

Question 2 Considérez la fonction suivante :

```
int beaucoup(int a, int b) {
  while(b-- >= 0) a += --a + b;
  return a;
}
```

Quelle valeur est renvoyée quand on appelle `beaucoup(4,3)` ? Réécrivez cette fonction pour qu'elle fasse le même calcul, mais sans effets de bord.

2 Expressions composées et conditionnelles

Question 3 Considérez la fonction suivante :

```
int mystere(int a, int b) {
  return (((a < 0 ? -a : a) % b) > 0) ? mystere(b,((a < 0 ? -a : a) % b)) : b;
}
```

Quel est l'algorithme implementé par cette fonction ? Réécrivez-la pour que chaque étape de calcul soit lisible directement.

Question 4 Ecrivez une seule instruction qui, étant données trois variables entières `x1`, `x2`, `x3` et `x4`, donne à chacune pour valeur la somme de son ancienne valeur et de la somme des valeurs des variables d'indice inférieur.

3 Opérateurs de traitement de bits et cryptage

On veut utiliser des opérations de bas niveau, traitant directement des bits en mémoire, pour crypter des messages. L'idée est de déplacer les bits de chaque caractère pour que la suite d'entiers représentant le message deviennent une suite d'entiers différents. Le déplacement de bits sera une rotation : on fait tourner les bits de droite à gauche (de manière cyclique), et pour décrypter le message il faudra opérer la rotation dans le sens inverse.

Question 5 Ecrire une fonction `unsigned char lire_bits_g(unsigned char x,int n)` qui renvoie un caractère contenant (à la même position) les n bits les plus à gauche de `x`, et `0` aux autres positions. Faire de même pour lire les bits les plus à droite.

Question 6 Ecrire une fonction `unsigned char ecrire_bits_g(unsigned char x,unsigned char y,int n)` qui renvoie `x`, dans lequel les n bits les plus à gauche ont été remplacés par les n bits les plus à gauche dans `y`. Faire de même pour les bits les plus à droite.

Question 7 Ecrire une fonction `unsigned char rotation_g(char x,int n)` décalant tous les bits de l'entier `x` de n positions vers la gauche et faisant passer chaque bit débordant de gauche à droite (c'est donc une rotation et non un décalage). Par exemple, la suite de bits `10110010`, après rotation gauche de 3 bits, devient `10010101`. Faire de même pour obtenir une rotation vers la droite.

Question 8 Ecrire un programme qui lit un texte depuis l'entrée standard et écrit sur la sortie standard la version cryptée de ce texte (en appliquant `rotation_g(x,n)` pour un certain n choisi, sur chaque caractère `x` de l'entrée).

Un message crypté par ce programme devient donc une suite d'entiers qui ne correspondent pas nécessairement à des lettres (voir la table des caractères ASCII pour avoir une idée des caractères possibles). On peut ensuite transmettre ce message et donner la *clé* correspondante, qui est l'entier n choisi comme nombre de positions dont on décale chaque bit. Le décryptage se fait par rotation dans l'autre sens.

Question 9 Ecrire un second programme, qui lit un message crypté depuis l'entrée standard et écrit sur la sortie standard le texte en clair, décrypté (en utilisant cette fois `rotation_d()`, pour déplacer les bits vers la droite).

On remarque que ce cryptage est plutôt mauvais : il n'y a que 7 valeurs possibles de n pour modifier un caractère. On peut améliorer ceci en n'utilisant pas toujours le même n. La clé est alors un tableau de k entiers, et on crypte ainsi les k premiers caractères avec ces entiers, avant de recommencer avec les k suivants, et ainsi de suite.

4 Champs de bits

La place occupée par un programme en mémoire est parfois un élément critique, que l'on peut maîtriser en utilisant des représentations compacts pour les données. Ici, on manipule des dates, en ne considérant qu'une plage d'un siècle pour les années (comme si on n'avait jamais entendu parler du bug de l'an 2000), en représentant donc l'année 2012 par le nombre 12, par exemple.

Question 10 Définissez un type `date` compact, par un champ de bits à trois champs : un pour l'année, un pour le mois et un pour le jour. Cette structure ne doit pas occuper plus de deux octets en mémoire.

Question 11 Ecrire une fonction `date nouvelle_date(int a,int m,int j)` prenant comme arguments une année, un mois et un jour, et renvoyant la date correspondante. Faire en sorte qu'elle imprime un message lorsque les arguments donnés sont invalides.

Question 12 Ecrire une fonction `void afficher_date(date d)`, qui permette d'afficher une date donnée dans le format `JJ/MM/AA`.

Question 13 Pour comparaison, implémentez les fonctions précédemment définies sans utiliser de champs de bits, mais avec un entier court pour représenter une date, et en manipulant ses composants par des opérations de traitement de bits (c'est beaucoup moins pratique).