

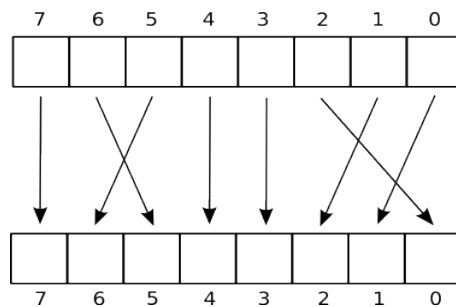


Examen n°2 de Programmation en C - IR1 2007-2008

Polycopiés de cours et notes de TD autorisés Durée: 2 heures

Exercice 1: bougez vos bits (6 points)

On veut disposer d'une fonction capable de déplacer les bits d'un octet suivant un ordre défini par une permutation (rappelez-vous, un concept vu en Maths 2) représentée par un tableau de 8 entiers compris entre 0 (bit de poids faible) et 7 (bit de poids fort). Par exemple, si l'on veut échanger les bits 5 et 6 et décaler circulairement les bits 0, 1 et 2, on utilisera le tableau suivant: {1,2,0,3,4,6,5,7}. En effet, le bit 0 se retrouvera en position 1, le bit 1 en position 2, le bit 2 en position 0, etc, comme on peut le voir sur le schéma suivant:



- 1) Écrire une fonction `est_permutation` prenant un tableau d'entiers supposé être de taille 8, et retournant 1 s'il contient tous les nombres compris entre 0 et 7; 0 sinon.
- 2) Écrire la fonction `void placer_un_bit(unsigned char src, unsigned char *dest, int pos_src, int pos_dest)`. Elle doit prendre le bit n° `pos_src` de `src` et donner la même valeur au bit n° `pos_dest` de `*dest`.
Exemple: pour `src=01100100`, `*dest=11101001`, `pos_src=6`, `pos_dest=2`, on doit obtenir au final `*dest=11101101`.
- 3) Écrire une fonction `deplacer_bits` prenant par adresse un octet non signé et un tableau de 8 entiers, qui effectue le déplacement des bits suivant l'ordre indiqué par le tableau d'entiers. Vous devrez naturellement gérer tous les cas d'erreurs.

Exercice 2: 73 6F 72 74 69 65 en hexadécimal (7 points)

On souhaite disposer d'un programme capable d'afficher le contenu d'un fichier en hexadécimal.

- 1) Écrire une fonction `print_hexa_octet` prenant un octet non signé et affichant sa valeur en hexadécimal sur deux caractères sur la sortie standard. Rappel: un octet est composé de 2 fois 4 bits, chaque groupe de 4 bits correspondant à un chiffre en hexadécimal. Exemple:

pour la valeur 10, on doit afficher **0A**. Bien entendu, l'usage de `printf("%02X", ...)` est interdit. Vous n'avez droit qu'à `printf("%c", ...)`.

- 2) Écrire une fonction `print_hexa_fichier` prenant un `FILE*` ainsi qu'un entier `N`, et affichant sur la sortie standard le contenu du fichier en hexadécimal, à raison de `N` valeurs hexadécimales par ligne, séparées par un espace. Par exemple, si l'on invoque la fonction avec un fichier contenant toutes les valeurs croissantes de 0 à 30 avec `N=7`, on doit obtenir le résultat suivant:

```
00 01 02 03 04 05 06
07 08 09 0A 0B 0C 0D
0E 0F 10 11 12 13 14
15 16 17 18 19 1A 1B
1C 1D 1E
```

- 3) On souhaite maintenant écrire le `main` du programme. Le comportement attendu est le suivant:
 1. Le programme prend en argument le nom du fichier à traiter. S'il n'y a pas d'argument, la lecture se fera sur l'entrée standard. S'il y a plusieurs arguments, seul le premier sera pris en compte.
 2. Le programme admet l'option `-N XXX` où `XXX` est la valeur du `N` utilisé dans la question précédente. Si l'option n'est pas utilisée, la valeur par défaut de `N` doit être 8. Naturellement, vous traiterez les cas d'erreur et vous utiliserez `getopt`.
- 4) Écrire une seconde version de la fonction `print_hexa_fichier` qui, après les valeurs en hexadécimal d'une ligne, affiche les caractères dont le code est supérieur ou égal à 32 (soit 20 en hexadécimal). Les autres seront symbolisés par des points. Par exemple, pour le fichier contenant:

```
Cet examen est super!
:)
```

on devra obtenir l'affichage suivant pour `N=8`:

```
43 65 74 20 65 78 61 6D Cet exam
65 6E 20 65 73 74 20 73 en est s
75 70 65 72 21 0A 0A 3A uper!...:
29 0A                               ).
```

Exercice 3: on fait des trucs avec des chaînes, mais on se protège (4 points)

On veut manipuler des chaînes de caractères qui évitent les problèmes de débordement en réallouant de l'espace quand c'est nécessaire. Pour cela, on utilisera le type suivant:

```
typedef struct {
    char* content; /* le buffer contenant les caractères */
    int size;      /* le nombre de caractères qui sont réellement
                   dans le buffer */
    int capacity; /* la taille maximum du buffer */
} String;
```

On ne mettra pas de `'\0'` dans le buffer.

- 1) Écrire une fonction **init_String** prenant un entier et construisant un objet **String** de la capacité correspondante.
- 2) Écrire la fonction **free_String** libérant toute la mémoire associée à un objet **String***.
- 3) Écrire la fonction **my_strcat** prenant un **String* leopard** et un **char* dassaut**, qui concatène **dassaut** à **leopard** en réallouant si nécessaire.

Exercice 4: pétanque mathématique (4 points)

- 1) On veut écrire une fonction **petanque** qui prenne un nombre réel double précision **cochonnet**, ainsi qu'un nombre variable de pointeurs vers des fonctions du même type que **cos**, c'est-à-dire prenant et retournant un réel double précision. Le dernier pointeur sera à **NULL**. La fonction **petanque** devra afficher sur une ligne les résultats des différentes fonctions appliquées au premier argument.

Exemple: **petanque(2.0, cos, sin, sqrt, NULL)** doit afficher ceci:

-0.416147 0.909297 1.414214

- 2) Modifier la fonction précédente pour qu'elle dise, en plus, quel pointeur de fonction donne le résultat le plus proche de **cochonnet**: **petanque2(2.0, cos, sin, sqrt, NULL)**

-0.416147 0.909297 1.414214

Le plus près est le pointeur n°2 (car 1.414214 est le nombre le plus proche de 2.0)