Programmation en C

Examen du 16/12/2011 - 3h

Bien qu'inutiles, les notes de cours sont autorisées, pas les livres. Dans une certaine mesure, la correction tiendra compte de la lisibilité du code produit et de son efficacité en termes de temps d'exécution et de quantité de mémoire utilisée. Chaque question peut être traitée indépendamment. Ne perdez pas de temps à écrire des `main` pour les fonctions demandées, ils ne seront pas pris en compte.

I - Tableaux

Les fonctions qui suivent supposent que `t` est l'adresse d'un tableau contenant exactement `n` éléments.

Exercice 1

Ecrire une fonction `int occs_non_finales(int *t, int n, int v)`. Cette fonction doit lire une seule fois le contenu du tableau d'adresse `t`, et renvoyer le nombre de fois où `v` apparaît dans les cases autres que la dernière. Si par exemple `v` vaut `1`, et si le tableau est `{0,1,2,1,1,0,1}`, la fonction renverra `3`.

Ecrire une seconde fonction `int *succ_occs(int *t, int n, int v)`. En utilisant la fonction precédente, cette fonction doit allouer, remplir et renvoyer l'adresse d'un tableau contenant, dans l'ordre, pour chaque case du tableau contenant `v` et qui n'est pas la dernière case du tableau, la valeur de la case suivante. Si par exemple `v` vaut `1` et le tableau est `{0,1,2,1,1,0,1}`, le tableau renvoyé sera `{2,1,0}`

Exercice 2

Un tableau sera dit *bivalué* s'il contient deux valeurs distinctes telles que chaque élément du tableau soit égal à la première ou à la seconde de ces valeurs. Par exemple, le tableau `{6,6,3,3,6,6,3,6,3}` est bivalué - les deux valeurs sont `6` et `3`.

Ecrire une fonction `int bivalue(int *t, int n)` renvoyant `1` si le tableau d'adresse `t` est bivalué, et `0` sinon. Le contenu du tableau ne devra être lu qu'au plus une fois.

II - Chaînes de caractères

Exercice 3

Une chaîne s sera dite *de période* p si elle est de la forme $m \dots m$, où m est un mot de longeur p. Noter que dans ce cas, m est nécessairement un préfixe de s.

> Par exemple, `"abababab"` est de période 2 (en prenant m égal à `"ab"`, la chaîne est bien égale à $mmmm$), mais aussi de période 4 (mm, en prenant m égal à `"abab"`), et de période 8 (en prenant pour m la chaîne elle-même).

Plus généralement, une chaîne s est toujours de période l, où l est la longueur de s. Elle admet aussi toujours une *plus petite période* – égale à 2 dans l'exemple précédent.

1. Ecrire une fonction `int periodique(char *s, int ls, int p)`. Cette fonction suppose que `s` est l'adresse d'une chaîne de caractères de longueur `ls`. Elle doit renvoyer `1` si cette chaîne est de période `p`, et `0` sinon.

   La fonction devra renvoyer sa réponse le plus rapidement possible. Elle peut être écrite par récurrence. Attention aux cas limites (chaîne vide et/ou période nulle, etc).

2. A partir de la fonction précédente, écrire `int periode(char *s)`, renvoyant la plus petite période de la chaîne d'adresse `s`. Cette fonction peut (et même, raisonnablement, doit) appeler la précédente. Vous pouvez aussi y invoquer la fonction prédéfinie `int strlen(char *s)`, renvoyant la longueur d'une chaîne.

Exercice 4

Un peu plus difficile... un *acrostiche palindromique* est un texte découpé en lignes, tel que la suite des premières lettres de chaque ligne forme un palindrome, c'est-à-dire un mot ayant la même suite de lettres lorsqu'on le lit de gauche à droite ou de droite à gauche. Par exemple, le texte suivant est un acrostiche palindromique :

```
ce texte est un exemple d'acrostiche
palindromique
avec pour suite de
premières lettres
cpapc, qui forme bien un palindrome.
```

Ecrire une fonction `int acropal(char *s)`, renvoyant `1` si le texte d'adresse `s` est un acrostiche palindromique et `0` sinon. On fera sur ce texte les hypothèses suivantes : chaque ligne du texte est suivi d'un retour à la ligne `'\n'`, y compris la toute dernière ligne; le texte contient au moins une ligne, et aucune ligne vide.

Cette fonction peut utiliser `strlen`, mais devra rester efficace dans son traitement. Elle ne doit pas déclarer ou allouer de nouveau tableau, mais simplement examiner le contenu du texte. Attention aux cas limites : le texte peut être réduit à une seule ligne, auquel cas il vérifie trivialement la condition ci-dessus. Le nombre de lignes peut d'autre part être pair ou impair.

III - Listes chaînées

On considère le type usuel permettant de représenter en C des listes chaînées d'entiers, avec les conventions habituelles :

```
struct cell {
  int clef;
  struct cell *suiv;
};
```

Les fonctions ci-dessous doivent être écrites par récurrence pure : pas de boucles, pas de variables statiques. Chaque fonction ne peut appeler qu'elle même et, si nécessaire, `malloc` et/ou `free`. Attention aux cas limites.

Exercice 5

Ecrire une fonction

```
void inserer_succ*(struct cell *pc, int n)
```

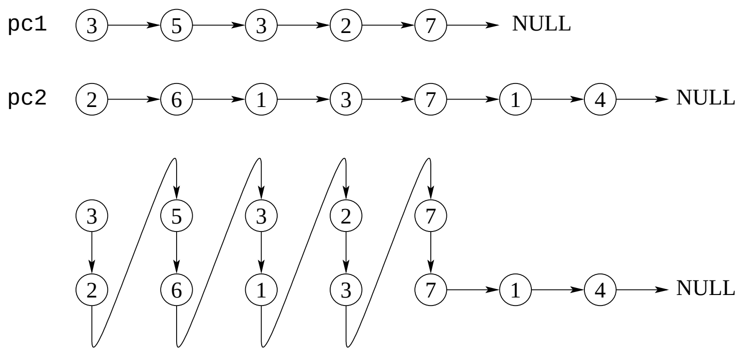
Cette fonction doit insérer, après chaque cellule de la liste `pc`, une nouvelle cellule de clef `n`. Si par exemple la liste a pour suite de clefs $(2, 3, 2, 1)$ et si `n` vaut `5`, la liste contiendra la suite de clefs $(2, 5, 3, 5, 2, 5, 1, 5)$ après l'appel. Seules les cellules insérées devront être alloués : les cellules de `pc` seront toutes conservées, et chaînées à ces nouvelles cellules.

Exercice 6

Etant données deux listes `pc1`, `pc2`, la liste formée par *alternance* de `pc1`, `pc2` est la liste formée de :

- la première cellule de `pc1` si elle existe, suivi de...
- la première cellule de `pc2` si elle existe, suivi de...
- la seconde cellule de `pc1` si elle existe, suivi de...
- la seconde cellule de `pc2` si elle existe, suivi de...
- la troisième cellule de `pc1` si elle existe, suivi de...
- la troisième cellule de `pc2` si elle existe, suivi de...
- etc., etc.

Voici un exemple de construction de la liste formée de l'alternance de deux listes :



Ecrire une fonction

```
struct cell alternance*(struct cell *pc1, struct cell *pc2)
```

construisant la liste formée de l'aternance de `pc1`, `pc2`. Cette fonction ne doit ni allouer ou libérer de cellules, ni modifier de clefs, mais seulement réorganiser le chaînage des listes `pc1`, `pc2`. Elle doit renvoyer un pointeur vers la première cellule de la liste résultante.