Programmation en C

Examen du 10/01/2008 - 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.

Partie I - Matrices

Etant donnée une suite finie d'entiers, sa *rotation droite* est obtenue en plaçant en tête son dernier élément : par exemple, la rotation droite de $(0, 1, 2, 3)$ est $(3, 0, 1, 2)$.

Question 1

Une matrice carrée est un *enroulement droit* si chaque ligne après la première est la rotation droite de la ligne précédente. Par exemple, la matrice suivante est un enroulement droit :

1	3	0	5
5	1	3	0
0	5	1	3
3	0	5	1

Ecrire une fonction `int enroulement(int m[TAILLE][TAILLE])` renvoyant `1` si `m` est un enroulement droit, `0` sinon. On supposera que `TAILLE` est une constante entière quelconque, définie en début de programme par un `#define`.

Question 2

Ecrire `void enrouler(int t[TAILLE], int m[TAILLE][TAILLE])`. En supposant que `t` contient `TAILLE` éléments, cette fonction doit remplir la matrice `m` de manière à ce qu'après exécution, `m` soit un enroulement droit dont la première ligne contient la suite des éléments de `t`. Elle ne doit écrire que dans `m`.

Partie II - Chaînes

Dans chacune des fonctions demandées ci-dessous, on supposera que chaque ligne des textes considérés y compris la dernière se termine par un retour à la ligne, *i.e.* par un `\n`.

Question 1

Un texte est dit *pyramidal* si chaque ligne autre que la première est au moins aussi longue que la ligne précédente. Par exemple,`"a\nb\ncdef\nghijkl\n"` est un texte pyramidal :

```
a
b
cdef
ghijkl
```

Ecrire `int pyramidal(char *s)` renvoyant `1` si `s` contient un texte pyramidal, et `0` sinon. Cette fonction ne peut en utiliser aucune autre, et ne devra lire le contenu de `s` qu'une seule fois.

Question 2

Un texte est dit *incrémental* si chaque ligne autre que la dernière est préfixe de la ligne suivante. Par exemple, `"a\nabc\nabcdef\nabcdefghi\n"` est un texte incrémental :

```
a
abc
abcdef
abcdefghi
```

Ecrire `int incremental(char *s)` renvoyant `1` si `s` contient un texte incrémental, et `0` sinon. Cette fonction ne peut en utiliser aucune autre, et ne devra relire qu'au plus une fois chaque ligne de `s`.

Question 3

Appelons *tableau des longueurs* d'un texte le tableau contenant une case de plus que le nombre de lignes du texte, contenant dans sa case n^o i la longueur de la ligne du texte n^0 i, et contenant `-1` dans sa toute dernière case. La numération des lignes commence à `0`, et on ne tient pas compte dans la longueur d'une ligne du caractère `\n`. Par exemple, le tableau des longueurs de `"a\naa\n\naa\n"` est `{1,2,0,2,-1}`.

Ecrire `int *tableau_longueurs(char *s)`. Cette fonction devra allouer puis remplir le tableau des longueurs de `s`, et renvoyer un pointeur vers son premier élément. La chaîne `s` ne pourra être lue que deux fois en tout.

Question 4

Appelons *tableau des positions* d'un texte le tableau contenant une case de plus que le nombre de lignes du texte, contenant dans sa case n^o i l'indice dans le texte du début de la ligne n^0 i, et contenant `-1` dans sa toute dernière case.

Ecrire `int *tableau_positions(int *tl)`. A partir de la seule donnée du tableau des longueurs `tl` d'un texte, cette fonction devra allouer puis remplir le tableau des positions du même texte, et renvoyer un pointeur vers son 1er élément.

Question 5

Un peu plus dfficile... appelons *symétrique* d'un texte le texte obtenu en inversant lignes et colonnes, et complétant par des espaces de manière à donner au texte résultant une forme rectangulaire :

’a’	’b’	’c’	’\n’	
’d’	’\n’			
’e’	’f’	’g’	’h’	’\n’
’i’	’j’	’\n’		

$\longrightarrow$

’a’	’d’	’e’	’i’	’\n’
’b’	’ ’	’f’	’j’	’\n’
’c’	’ ’	’g’	’ ’	’\n’
’ ’	’ ’	’h’	’ ’	’\n’

Noter que le texte obtenu a un nombre de lignes égal au nombre de caractères de la plus grande ligne du texte initial, et que chaque ligne est, sans tenir compte du `\n`, de longeur égale au nombre de lignes du texte initial. Ecrire une fonction `char *symetrique(char *s)` allouant et construisant le symétrique de `s`, et renvoyant un pointeur vers son 1^{er} élément. Vous pouvez pour cette question vous servir des deux fonctions précédentes, et éventuellement écrire des fonctions auxiliaires.

Partie III - Listes

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;
};
```

Le *rang* d'une clef dans une liste est le nombre d'élements qui la précèdent : la première clef d'une liste est de rang `0`, la suivante de rang `1`, etc.

Toutes les fonctions qui suivent doivent obligatoirement être écrites par récurrence pure : elles ne peuvent contenir ni `for`, ni `while`, ni variables statiques.

Question 1

Une liste est dite *duplicante* si elle est de longueur paire, et si chaque clef de rang pair est égale à la clef qui la suit. Par exemple, la liste contenant successivement les clefs `1, 1, 3, 3, 0, 0` est duplicante.

Ecrire `int duplicante(struct cell *pc)` renvoyant `1` si la liste `pc` est duplicante, `0` sinon.

Question 2

Ecrire `struct cell *suite(int i, struct cell *pc)`. Cette fonction suppose que `pc` contient au moins `i + 1` cellules. Elle doit renvoyer un pointeur vers la cellule de rang `i` de `pc`.

Question 3

Ecrire `int clef_prefixe(int k, struct cell *pc)`. Cette fonction doit renvoyer `1` si et seulement si `pc` commence par `k` clefs toutes égales entre elles, et `0` dans tous les autres cas (en particulier si la liste contient moins de `k` clefs).

Question 4

Une liste est dite *réplicante d'ordre* `n` si la suite de ses clefs est de la forme :

$$\overbrace{\mathtt{c}_0, \ldots, \mathtt{c}_0}^{\mathtt{n}\text{ fois}}, \ \ldots, \ \overbrace{\mathtt{c}_p, \ldots, \mathtt{c}_p}^{\mathtt{n}\text{ fois}},$$

Autrement dit, elle contient `n` fois une certaine clef, suivi de `n` fois une certaine clef, suivi de `n` fois une certaine clef, etc. Par convention, la liste vide est réplicante d'ordre `n`. A partir des fonctions `suite` et `clef_prefixe` des questions précédentes, écrire par récurrence `int replicante(int n, struct cell *pc)` renvoyant `1` si `pc` est réplicante d'ordre `n`, et `0` sinon.

Casse-tête final

Ce casse-tête est en bonus, et ne doit être abordé que si vous avez traité tout ce qui précède. Un explorateur dispose de la carte d'un labyrinthe dont les pièces sont numérotées de `0` à `N`. La pièce n^o`0` est l'entrée du labyrinthe. La carte se présente comme une matrice `m` de taille `N`×`N` contenant des `0` et des `1` : la case `m[i][j]` est à `1` s'il existe un passage allant de la pièce n^o `i` à la pièce n^o `j` - la forme des passages fait que l'inverse n'est pas garanti, à moins que que la case `m[j][i]` ne soit elle aussi à `1`. Sachant qu'il dispose d'un ordinateur portable et d'un compilateur C, comment doit-il programmer - quitte à surcharger la représentation de sa carte - le calcul de l'ensemble des numéros de pièces qu'il peut atteindre à partir de l'entrée, en étant sûr de pouvoir revenir à son point de départ ?