Programmation en C

Examen du 22/12/2006 - 3h

Bien qu'inutiles, les notes de cours sont autorisées, pas les livres. Inutile d'écrire des `main` pour les fonctions demandées. Le casse-tête final (4) est en bonus.

1. Matrices

On dit qu'une matrice d'entiers est un *masquage* si elle ne contient que des `0` et des `1`.

- Un masquage est dit *supérieur* si ses `1` sont tassés vers le haut : aucun `0` ne se trouve au dessus d'un `1`.
- Un masquage est dit *gauche* si ses `1` sont tassés vers la gauche : aucun `0` ne se trouve à gauche d'un `1`.

Par exemple, la matrice ci-dessous est un masquage à la fois supérieur et gauche. Ses `1` sont tassés à la fois vers la gauche et vers le haut.

	0	1	2	3	4
0	1	1	1	1	0
1	1	1	1	1	0
2	1	1	0	0	0
3	1	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Dans les questions qui suivent, `HAUTEUR` et `LARGEUR` sont deux constantes entières déclarées en début de programme.

Exercice 1.1

Noter qu'à partir d'un masquage quelconque, on peut construire un masquage gauche en tassant tous ses `1` vers la gauche : il suffit pour chaque ligne de compter son nombre de `1`, puis de réécrire cette ligne en ramenant tous les `1` en début de ligne. Ecrire :

```
void tasser_gauche (int m[HAUTEUR][LARGEUR])
```

Cette fonction suppose que `m` est un masquage. Elle doit modifier le contenu de `m` de manière a construire un masquage gauche, suivant la méthode ci-dessus.

Exercice 1.2

Ecrire :

```
int sup_gauche (int m[HAUTEUR][LARGEUR])
```

renvoyant `1` si la matrice `m` est un masquage supérieur gauche, et `0` sinon. Cette fonction devra aussi gérer le cas où `m` n'est pas même un masquage. La matrice ne devra être parcourue qu'une seule fois.

2. Abrégé d'abrégé de littérature potentielle

Dans les exercices qui suivent, on ne considère que des chaînes dont toutes les lettres sont comprises entre `'a'` et `'z'` (pas de majuscules ni d'accents), mais pouvant contenir d'autres symboles servant à séparer les mots (espaces, points, etc.) Tous les symboles qui ne sont pas des lettres seront considérés comme des caractères de séparation, et uniformément appelés *séparateurs*.

Exercice 2.0 - Préliminaire

La fonction suivante est utilisable dans toute la suite de l'énoncé. Ecrire :

```
int lettre (char c)
```

renvoyant `1` si `c` est une lettre, `0` si c'est un séparateur.

Exercice 2.1 - Pangramme

On dit qu'une chaîne de caractères est un *pangramme* si elle mentionne une et une seule fois chacune des 26 lettres comprises entre `'a'` et `'z'`. Par exemple, `"cwm, fjord-bank glyphs vext quiz !"` est un pangramme (anglais). Ecrire :

```
int pangramme (char *s)
```

renvoyant `1` si `s` est un pangramme et `0` sinon, en respectant la contrainte suivante : la chaîne `s` ne devra être lue qu'une seule fois (pas de `strlen`, pas de copie de `s`, etc).

Exercice 2.2 - Texte palindromique

On dit qu'une chaîne de caractères est *palindromique* si, sans tenir compte de ses séparateurs, elle est égale à la chaîne obtenue en inversant l'ordre de ses lettres. Par exemple, `"jelenovi pivo nelej !..."` est une chaîne palindromique (tchèque). Ecrire :

```
int palindromique (char *s)
```

renvoyant `1` si `s` est palindromique et `0` sinon. Vous pouvez utiliser `strlen`.

Exercice 2.3 - Chaîne-valise

Etant données deux chaînes `s` et `t` et une position `pos`, on dit que `s` *se superpose à* `t` *en* `pos` si la suite des caractères de `s` à partir de `pos` est un préfixe de la suite des caractères de `t`. Par exemple, `"superpose"` se superpose à `"posement"` à la position `5` :

0	1	2	3	4	5	6	7	8	9				
s	u	p	e	r	p	o	s	e	\0				
					p	o	s	e	m	e	n	t	\0

Noter que pour toutes chaînes `s` et `t`, il existe toujours *au moins une* position de superposition valide, celle où `pos` est placé sur le marqueur de fin `'\0'` de `s` : dans ce cas, la suite *vide* de caractères de `s` à partir de `pos` se superpose au préfixe *vide* de `t`. Par exemple `"abc"` se superpose à `"def"` à la position `3`.

Etant données deux chaînes `s` et `t`, on appelle *chaîne-valise formée à partir de* `s` *et* `t` la chaîne contenant :

- tous les caractères de `s` jusqu'à `(pos - 1)` inclus, où `pos` est la $1^{\text{ère}}$ position en laquelle `s` se superpose à `t`,
- puis, tous les caractères de `t`.

Par exemple, la chaîne-valise formée à partir de `"superpose"` et `"posement"` est `"superposement"`. Celle formée à partir de `"abc"` et `"def"` est `"abcdef"`. Ecrire :

1. `int superpose (int pos, char *s, char *t)`

   renvoyant `1` si `s` se superpose à `t` en `pos`, et `0` sinon.

2. `char *allouer_chaine (int n)`

   cette fonction doit allouer dynamiquement une zone mémoire suffisamment grande pour stocker une chaîne contenant `n` caractères, et renvoyer un pointeur vers la zone allouée.

3. `char *chaine_valise (char *s, char *t)`

   à l'aide des deux fonctions précédentes, cette fonction doit allouer puis construire la chaîne-valise formée à partir de `s` et `t`, et renvoyer un pointeur vers la zone mémoire allouée pour cette chaîne.

3. 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;
};
```

Exercice 3.1

Ecrire :

`int clefs_egales (struct cell *pc)`

renvoyant `1` si toutes les clefs de `pc` sont égales et `0` sinon. Noter que toutes les clefs d'une liste vide ou contenant une seule clef sont égales, puisqu'elle ne contient pas deux clefs différentes. Cette fonction peut être écrite par récurrence.

Exercice 3.2

Une suite est *monotone* si elle est croissante (chaque élément inférieur ou égal au suivant) ou décroissante (chaque élément supérieur ou égal au suivant). En particulier, une suite est monotone si tous ses éléments sont égaux. Ecrire :

`int monotone (struct cell *pc)~`

renvoyant `1` si la suite des clefs de `*pc` est monotone, et `0` sinon. Cette fonction est plus facile à écrire sans récurrence. La liste `pc` ne devra être parcourue qu'une seule fois.

4. Casse-tête final

Cet exercice est plutôt difficile, et ne doit être traité que si vous avez résolu ce qui précède. Une chaîne de caractères est dite *anacyclique* si, sans tenir compte des caractères qui ne sont pas des lettres (espacements, ponctuations, etc...), elle est égale à la chaîne obtenue en inversant l'ordre de ses mots, sans inverser l'ordre des lettres de chaque mot. Par exemple,

```
"souffrir sans amour, l'oublies-tu parfois ?...  parfois,
tu oublies l'amour sans souffrir !"
```

est une chaîne anacyclique. Ecrire

```
int anacyclique (char *s)
```

renvoyant `1` si `s` est anacyclique et `0` sinon. On supposera que toutes les lettres de mots de `s` sont comprises entre `'a'` et `'z'`, tous les autres symboles étant considérés comme des séparateurs. Vous pouvez utiliser `strlen` et la fonction `lettre` de l'exercice 2.0. Il est conseillé de bien analyser le problème, et d'écrire une ou plusieurs fonctions auxilaires.