Programmation en C

Examen du 10/01/2009 - 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

Exercice 1

Appelons *bisymétrique* toute matrice dont le contenu ne change pas si on la représente en inversant la droite et la gauche, ou en inversant le haut et le bas. Par exemple, les deux matrices suivantes sont bisymétriques :

$$\begin{array}{|cccc|} \hline 1 & 2 & 2 & 1 \\ 3 & 6 & 6 & 3 \\ 7 & 4 & 4 & 7 \\ 3 & 6 & 6 & 3 \\ 1 & 2 & 2 & 1 \\ \hline \end{array} \qquad \begin{array}{|ccccc|} \hline 1 & 2 & 5 & 2 & 1 \\ 3 & 6 & 9 & 6 & 3 \\ 7 & 4 & 8 & 4 & 7 \\ 3 & 6 & 9 & 6 & 3 \\ 1 & 2 & 5 & 2 & 1 \\ \hline \end{array}$$

Ecrire `int bisymetrique(int m[H][L])`, où `H` et `L` sont deux constantes, et où `m` est supposée de taille `H` × `L`. Cette fonction doit renvoyer `1` si `m` est bisymétrique, et `0` sinon. Elle doit renvoyer sa réponse le plus rapidement possible, en particulier en évitant au maximum de comparer deux cases déjà comparées.

Exercice 2

Soit `t` un tableau dont les valeurs sont comprises entre `0` et `N - 1`, où `N` est une constante quelconque. Appelons *matrice de succession* de `t` la matrice `m` de taille `N` × `N`, tel que `m[i][j]` contienne le nombre de fois où l'entier `i` est immédiatement suivi de l'entier `j` dans `t`. Par exemple, si `N` vaut `3` et si `t` contient `{0, 0, 0, 2, 1, 2, 2, 0, 0}`, la matrice de succession de `t` est :

$$\begin{array}{|ccc|} \hline 3 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ \hline \end{array}$$

Ecrire `void successions(int n, int *t, int m[N][N])`. Cette fonction suppose que `t` est l'adresse d'un tableau à `n` éléments, de valeurs comprises entre `0` et `N - 1`. La matrice `m` sera supposée allouée de taille `N` × `N`, mais de contenu quelconque au moment de l'appel. En fin d'appel, `m` devra être égale à la matrice de succession de `t`. Le contenu de `t` ne devra être lu qu'une seule fois.

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`.

Exercice 3

Appelons *carré* tout texte dont les lignes sont de même longueur, et, si chaque ligne contient `n` caractères, contenant exactement `n` lignes - autrement dit, ce texte a la forme d'un carré de taille $\mathtt{n} \times \mathtt{n}$. Par exemple, `"abc\ndef\nghi\n"` est carré :

```
abc
def
ghi
```

Ecrire `int carre(char *s)` renvoyant `1` si le texte stocké à l'adresse `s` est carré, et `0` sinon. Ce texte ne devra être lu qu'au plus une seule fois.

Exercice 4

On numérote les lignes d'un texte à partir de `0`, et chaque caractère de chaque ligne à partir de `0`. Ecrire les fonctions suivantes :

1. `char *ligne(int k, char *s)`

   Si le texte stocké à l'adresse `s` contient bien une ligne n^o `k`, cette fonction doit renvoyer l'adresse d'une zone mémoire (allouée par `malloc`) contenant une copie conforme de cette ligne - sans le `'\n'` qui suit cette ligne, mais terminée par un `'\0`. Sinon, cette fonction doit renvoyer un pointeur nul.

2. `char *colonne(int k, char *s)` Cette fonction doit renvoyer l'adresse d'une zone mémoire (allouée par `malloc`) contenant dans l'ordre, et pour chaque ligne : son caractère n^o `k` s'il existe, un espace sinon. Cette suite de caractères sera terminée par un `'\0`.

3. `char *effacer_ligne(int k, char *s)`

   Si le texte stocké à l'adresse `s` contient bien une ligne n^o `k`, cette fonction doit renvoyer l'adresse d'une zone mémoire (allouée par `malloc`) contenant une copie du texte sans sa ligne n^o `k`. Cette copie sera terminée par un `'\0`. Sinon, cette fonction doit renvoyer un pointeur nul.

4. `char *effacer_colonne(int k, char *s)`

   Cette fonction doit renvoyer l'adresse d'une zone mémoire (allouée par `malloc`) contenant une copie du texte dans laquelle chaque ligne sera privée de son caractère n^o `k` s'il existe, et recopiée telle quelle sinon. Cette copie du texte sera terminée par un `'\0`.

Dans chaque cas, la zone mémoire allouée doit être exactement de la taille nécessaire (il faut donc calculer cette taille). Chaque fonction doit minimiser le nombre de relectures du texte et/ou d'une ligne.

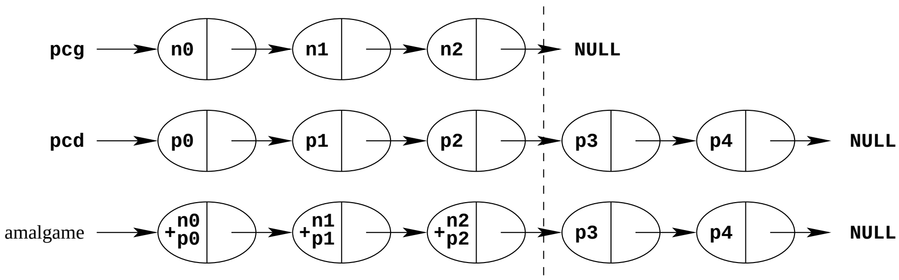
Partie III - Listes

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;
};
```

Exercice 5

Soient `pcg`, `pcd` deux listes. Soit `k` le nombre de clefs de la plus courte. Appelons *amalgame de* `pcg` *et* `pcd` la liste constituée : de `k` clefs, chacune égale à la somme des clefs de `pcg` et `pcd` de même rang ; puis, de toutes les clefs de la plus longue liste après ses `k` premières :



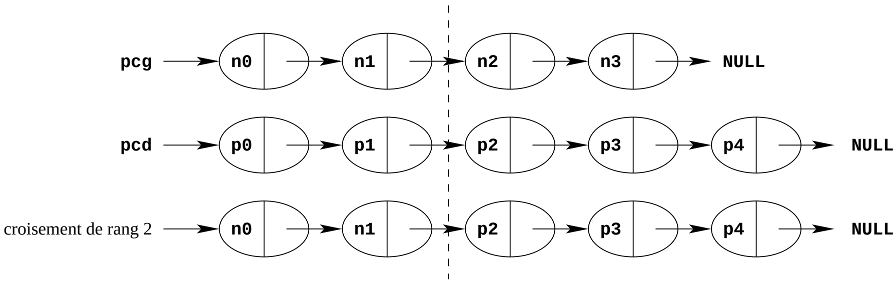
Ecrire *par récurrence* (ni boucles, ni variables statiques) une fonction

```
struct cell *amalgame(struct cell *pcg, struct cell *pcd)
```

Cette fonction doit renvoyer une nouvelle liste (dont les cellules seront allouées par `malloc`) dont la suite des clefs est celle de l'amalgame de `pcg` et `pcd`.

Exercice 6

Soient `pcg`, `pcd` deux listes. Etant donné un entier `k` positif, appelons *croisement de rang* `k` *de* `pcg` *et* `pcd` la liste constituée des `k` premières clefs de `pcg` (ou de toutes les clefs de `pcg` si `pcg` contient moins de `k` clefs), puis des clefs de `pcd` après les `k` premières (ou d'aucune autre clef si `pcd` contient moins de `k` clefs) :



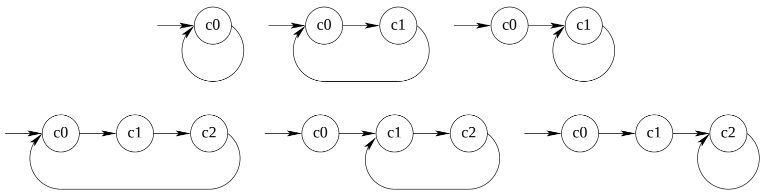
Ecrire *par récurrence* (ni boucles, ni variables statiques) une fonction

```
struct cell *croisement(int k, struct cell *pcg, struct cell *pcd)
```

Cette fonction doit renvoyer une nouvelle liste (dont les cellules seront allouées par `malloc`) dont la suite des clefs est celle du croisement de rang `k` de `pcg` et `pcd`.

Exercice 7

Soit `pc` une liste chaînée consistant en une cellule c_0 chaînée à une cellule c_1, chaînée à une cellule c_2, etc. On dira que cette liste est *bouclante* s'il existe i, j, avec $i \leq j$, tels que c_j soit chaînée à c_i.



Ecrire, par itération, une fonction `int bouclante(struct cell *pc)` renvoyant `1` si la liste `pc` est bouclante, et `0` sinon. Votre fonction doit bien sûr toujours terminer, même si la liste est bouclante. Elle ne doit pas modifier la structure de la liste, ni son contenu. Noter qu'il est possible que toutes les clefs de la liste soient égales.

Casse-tête final

Ce casse-tête ne doit être traité que si vous avez traité tout ce qui précède. Etonnement, il est possible d'écrire une fonction effectuant le traitement demandé dans l'exercice 7 en temps *linéaire* (et non quadratique). Après tout, la seule vraie contrainte est que la liste soit dans le même état en début et en fin d'appel... Comment faire ? Sauriez-vous écrire cette fonction ?