Programmation en C

Examen du 23/01/2010 - 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

Appelons *triangulaire supérieure stricte* toute matrice d'entiers carrée :

- dont les valeurs strictement au dessus de la diagonale sont toutes nulles.
- dont toutes les autres valeurs sont non nulles.

Par exemple, la matrice suivante est triangulaire supérieure stricte :

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 5 & 6 & 5 & 0 \\ 3 & 4 & 5 & 42 \end{pmatrix}$$

Ecrire `int ts_str(int m[N][N])` où `N` est une constante quelconque définie en début de programme, renvoyant `1` si `m` est une matrice triangulaire supérieure stricte et `0` sinon. Le contenu de la matricene devra être lu qu'au plus une fois, et la fonction devra renvoyer sa réponse le plus rapidement possible.

Partie II - Chaînes et itérations

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` - tenez en compte dans la recherche de vos solutions.

Exercice

Question 1

Ecrire une fonction

`void afficher_page(int k, char *s)`

Cette fonction doit afficher à l'écran le texte stocké à l'adresse `s` sur une largeur maximale de `k + 1` colonnes - les lignes trop longues seront coupées par autant de retours à la ligne que nécessaire. Par exemple, si `k` vaut `1` et si le texte est `"abcde\nfg\nh\n"` le texte affiché sera:

```
ab
cd
e
fg
h
```

Noter que les seuls retours à la ligne ajoutés sont après `"ab"` et `"cd"` : tous les autres font partie du texte. Le texte ne devra être lu qu'une seule fois.

Question 2

Ecrire à présent une fonction

```
char *mettre_en_page(int k, char *s)
```

Cette fonction doit allouer (par `malloc`) et construire une chaîne dont le texte sera celui stocké à l'adresse `s` avec tous les retours à la ligne ajoutés par la fonction précédente. La fonction renverra l'adresse de la zone-mémoire allouée.

Par exemple, si `k` vaut `1` et si le texte est à nouveau `"abcde\nfg\nh\n"`, le texte produit sera `"ab\ncd\ne\nfg\nh\n"`.

Le texte stocké à l'adresse `s` ne devra être lu que deux fois en tout : une fois pour déterminer la taille d'allocation, une fois pour la construction du nouveau texte.

Question 3

Un peu plus difficile, et sur un principe un peu différent, écrire une fonction

```
char *aligner_retours(int k, char *s)
```

Cette fonction doit allouer et construire une chaîne dont le texte sera : le texte stocké à l'adresse `s`, dans lequel toute ligne contenant moins de `k + 1` caractères avant son `'\n'` sera complétée par des espaces, de manière à ce qu'elle contienne exactement `k + 1` caractères avant `'\n'`. La fonction renverra l'adresse de la zone-mémoire allouée.

Par exemple, si `k` vaut `1` et si le texte est `"abc\nde\nf\n"`, le texte produit sera `"abc\nde\nf \n"` - les lignes `"abc"` et `"de"` sont suffisamment longues, la ligne `"f"` est complétée par un seul espace.

Là encore, et pour les mêmes raisons, le texte stocké à l'adresse `s` ne devra être lu que deux fois en tout. Vérifiez soigneusement vos indices et vos bornes.

Partie III - Listes et récurrence

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;
};
```

Par convention, la *position* d'une cellule dans une liste est le nombre de cellules qui la précèdent, e.g. la première cellule d'une liste est à la position zéro.

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, `free`. N'oubliez pas de gérer les cas limites (liste vide, éventuellement liste singleton).

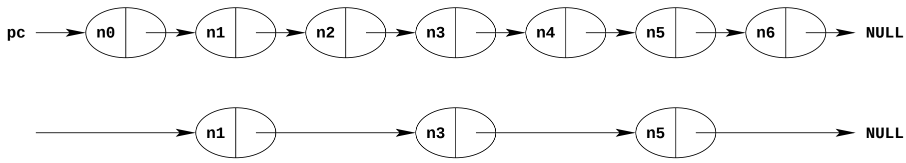
Exercice

Question 1

Ecrire par récurrence une fonction

```
struct cell *clefs_impaires(struct cell *pc)
```

Cette fonction doit supprimer une partie des cellules de `pc`, en ne gardant que celles de positions impaires.



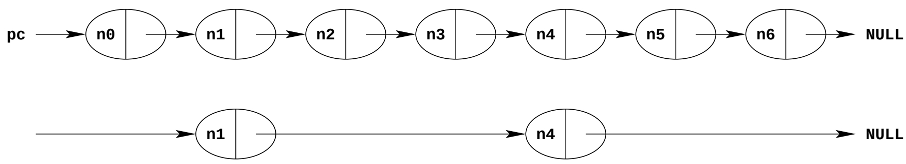
L'espace mémoire des cellules supprimées devra être libéré par `free`. La fonction devra renvoyer l'adresse de la première cellule de la liste résultante.

Question 2

Ecrire par récurrence une fonction

```
struct cell *sous_liste(int k, int n, struct cell *pc)
```

Cette fonction doit supprimer une partie des cellules de `pc`, en ne gardant qu'une cellule sur `n` à partir de la cellule de position `k`. Voici par exemple le résultat de `sous_liste(1,3,pc)` (on garde une cellule sur 3 à partir de la position 1) :



L'espace mémoire des cellules supprimées devra être libéré par `free`. La fonction devra renvoyer l'adresse de la première cellule de la liste résultante.

Exercice

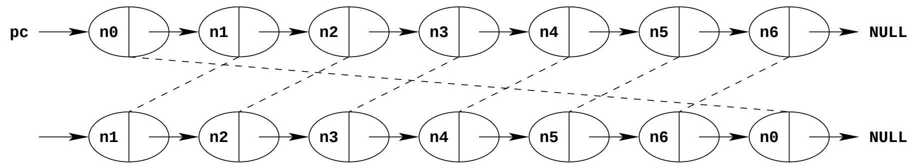
Cet exercice est un peu plus difficile. Faites attention à l'ordre des opérations, et ne perdez pas de lien vers l'une des cellules.

Question 1

Ecrire par récurrence une fonction

```
struct cell *rotation_gauche(struct cell *pc)
```

En ne modifiant que son chaînage, cette fonction doit effectuer une rotation gauche de `pc` : la première cellule de `pc` doit être déplacée en fin de liste.



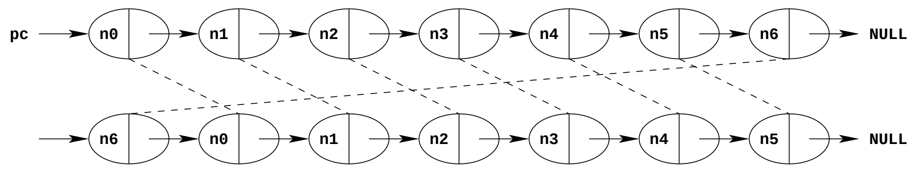
Cette fonction ne devra allouer ou libérer aucune cellule, ni modifier de clefs. Elle devra renvoyer l'adresse de la première cellule de la liste résultante.

Question 2

Ecrire par récurrence une fonction

```
struct cell *rotation_droite(struct cell *pc)
```

En ne modifiant que son chaînage, cette fonction doit effectuer une rotation droite de `pc` : la dernière cellule de `pc` doit être déplacée en tête de liste.



Cette fonction ne devra allouer ou libérer aucune cellule, ni modifier de clefs. Elle devra renvoyer l'adresse de la première cellule de la liste résultante.