Programmation en C

Examen du 14/06/2011 - 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.

Partie I - Chaînes

Les textes considérés dans cette partie seront supposées écrits sans majuscules ni accents - les caractères qui ne sont pas entre `'a'` et `'z'` seront tous considérés comme des séparateurs de mots. Il peut y avoir plus d'un séparateur entre deux mots, et éventuellement un ou plusieurs séparateurs avant le premier mot ou après le dernier.

Exercice 1

Un *abécédaire* est un texte de 26 mots dans lequel la suite des premières lettres de chaque mot forme exactement la suite des lettres de l'alphabet. Par exemple, le texte suivant est un abécédaire (les accents et majuscules ont été supprimés) :

> *a brader : cinq danseuses en froufrou grassouillettes, huit ingenues joueuses kleptomanes le matin, neuf (ou plus) quadragenaires rabougries, six travailleuses, une valeureuse walkyrie, x yuppies zelees.*

Ecrire une fonction `int est_abc(char *s)` renvoyant `1` si la chaîne `s` est un abécédaire, et `0` sinon. Cette chaîne ne devra être lu qu'au plus une fois.

Exercice 2

Une *belle absente*, de base m, est un texte t contenant autant de lignes que le nombre de lettres du mot m. Chaque ligne de t peut contenir des lettres qui ne sont pas dans m, mais : la première ligne contient toutes les lettres de m sauf la première, la seconde ligne contient toutes les lettres de m sauf la seconde, etc.

Le poème suivant, dû à Georges Perec, est une belle absente de mot de base "oulipo" ('u', 'l', 'i', 'p' apparaissent dans la 1ère ligne, mais pas 'o', etc) :

> *champ defait jusqu'a la ligne breve,*
> *j'ai desire vingt-cinq fleches de plomb*
> *jusqu'au front borne de ma page chetive.*
> *je ne demande qu'au hasard cette fable en prose vague,*
> *vestige du charme deja bien flou qui*
> *defie ce champ jusqu'a la ligne breve.*

Ecrire une fonction `int absente(char *m, char *t)` renvoyant `1` si `t` est une belle absente de base `m`, et `0` sinon. On supposera que chaque ligne du texte, y compris la dernière, se finit par un unique `'\n'`. Le texte ne devra être lu qu'au plus une fois. Essayez de rester méthodique et efficace, et faites attention au cas (comme ci-dessus) où une lettre apparaît plusieurs fois dans `m`.

Partie II - Listes chaînées 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;
};
```

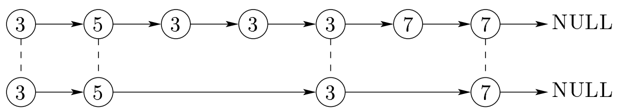
Les fonctions ci-dessous indiquées comme devant être recursives doivent être écrites par récurrence pure : pas de boucles, pas de variables statiques. Chaque fonction ne peut appeler qu'elle même et, si nécessaire, `malloc` ou `free`. Attention aux cas limites.

Exercice 3

Ecrire une fonction récursive

```
struct cell *non_iter(struct cell *pc)
```

Cette fonction doit extraire de `pc` une sous-liste de plus grande longueur possible, et ne contenant pas deux clef successives égales. Si par exemple `pc` a pour suite de clefs (3,5,3,3,3,7,7), la liste résultante aura pour suites de clefs (3,5,3,7).



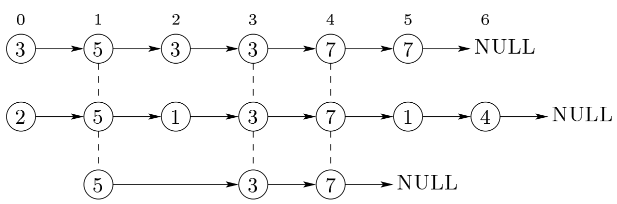
Cette fonction ne doit pas créer de nouvelle cellules ni modifier de clefs, mais simplement réorganiser le chaînage des cellules de `pc`. Elle devra libérer tout l'espace mémoire alloué pour les cellules non conservées, et renvoyer l'adresse de la première cellule de la liste résultante.

Exercice 4

Ecrire une fonction récursive

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

A partir de `pc1`, `pc2`, cette fonction doit construire une nouvelle liste formée de la suite de toutes les clefs à la même position dans les deux listes. Si par exemple `pc1` a pour suite de clefs (3,5,3,3,7,7) et `pc2` a pour suite de clefs (2,5,1,3,7,1,4), la liste résultante aura pour suite de clefs (5,3,7).



Cette fonction ne doit pas modifier `pc1` ou `pc2`. Elle doit renvoyer l'adresse de la première cellule de la liste construite.

Exercice 5

On souhaite écrire une fonction permettant étant donnée une liste, de supprimer de cette liste toutes les cellules contenant une clef strictement supérieures à la première lorsque cette liste est non vide. Cette fonction sera de la forme :

```
void liste_inf(struct cell *pc)
```

La façon la plus simple de procéder est d'écrire pour cette fonction une seconde fonction auxiliaire `aux_inf` qui sera, elle, récursive et qui sera appelée par `list_inf`. Choisissez le type et les arguments de `liste_inf` et écrivez ces deux fonctions. Aucune ne devra créer de nouvelles cellules ni modifier de clefs. Tout l'espace mémoire alloué pour les cellules non conservées devra être libéré.

Casse-tête final

Il se trouve que la fonction `liste_inf` ci-dessus *peut* être écrite de manière récursive, sans aucune fonction auxiliaire (sauf `free`), sans boucles, sans variables statiques, etc. - c'est assez difficile, mais pas impossible. Comment ?