Programmation en C

Examen du 5/01/2012 - 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.

I - Tableaux

Les fonctions qui suivent supposent que `t` est l'adresse d'un tableau contenant exactement `n` éléments.

Exercice 1

Ecrire une fonction `void annuler_reps(int *t, int n)`. Cette fonction doit remplacer par `0` le contenu de chaque case du tableau d'adresse `t` contenant une certaine valeur `v`, mais située plus à droite qu'une autre case contenant `v`. Si par exemple le tableau est `{3,0,2,2,3,1,0,2,3}`, le tableau résultant sera `{3,0,2,0,0,1,0,0,0}`.

Exercice 2

Ecrire une fonction `int max_reps(int *t, int n)`. Cette fonction suppose que le tableau d'adresse `t` est non vide, trié par ordre croissant, et qu'il peut contenir des répétitions. Elle doit renvoyer la plus grande des valeurs apparaissant le plus grand nombre de fois dans le tableau. Le contenu du tableau ne devra être lu qu'une seule fois.

Si par exemple le tableau est `{0,1,1,1,2,2,2,3,3}` – où `1`, `2` sont les deux valeurs apparaissant le plus grand nombre de fois – la valeur renvoyée sera `2`.

II - Chaînes de caractères

Les fonctions qui suivent supposent que `s` est l'adresse d'une chaîne de caractères.

Exercice 3

Un chaîne de caractères sera dite *bi-voyellique* si elle contient deux voyelles distinctes (parmi `'a'`, `'e'`, `'i'`, `'o'`, `'u'`, `'y'`) telle que chaque voyelle du texte soit égal à la première ou à la seconde de ces voyelles. Par exemple `"xxoooaxzaaaxzzoaox"` est bi-voyellique – les deux voyelles sont `'a'` et `'o'`.

Ecrire `int bi_voyellique(char *s)` renvoyant `1` si la chaîne d'adresse `s` et bi-voyellique, et `0` sinon. On supposera la chaîne écrite sans majuscules ni accents. La fonction doit renvoyer sa réponse le plus rapidement possible. La chaîne ne devra être parcourue qu'au plus d'une fois.

Exercice 4

Rappelons que lorsqu'une chaîne u est de la forme vw, la chaîne v est dite *préfixe* de u. Par exemple, les chaînes `""`, `"a"`, `"ab"` et `"abc"` sont toutes prefixes de la chaîne `"abc"`.

D'autre part, si l'on considère un ensemble de chaînes donné, ces chaînes ont toujours un plus long préfixe commun. Par exemple, `"ab"` est le plus long préfixe commun aux chaînes `"abcd"`, `"abdef"`, `"abdx"`. La chaîne vide `""` est le plus long préfixe commun aux chaînes `"abc"` et `"def"`.

On suppose à présent que le texte d'adresse `s` est découpé en lignes : chaque ligne se termine par un retour à la ligne `'\n'`, y compris la dernière, immédiatement avant le marqueur de fin de chaîne `'\0'`.

Ecrire une fonction `int prefixe_commun(char *s)` renvoyant la longueur du plus grand préfixe commun à chacune des lignes du textes. Attention à la prolifération de variables inutiles.

III - 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;
};
```

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, `malloc` et/ou `free`. Attention aux cas limites.

Exercice 5

Appelons *compression* d'une liste `pc` la liste contenant une seule cellule dont la clef est égale à la somme des clef de `pc`. Par exemple, si la suite des clefs de `pc` est $(0, 2, 1, 3)$, son déploiement aura pour unique clef 6. Par convention, la compression d'une liste vide est la liste d'unique clef 0.

Ecrire une fonction `struct cell *compression(struct cell *pc)`. Cette fonction doit renvoyer l'adresse d'une nouvelle liste formant la compression de `pc`, tout en libérant intégralement l'espace mémoire alloué pour les cellules de `pc`.

Exercice 6

Etant donnée une liste `pc` dont toutes les clefs sont supérieures ou égales à `0`, appelons *déploiement* de `pc` la liste dont toutes les clefs sont égales à `1`, et dont la longueur est la somme des clefs de `pc`. Par exemple, si la suite des clefs de `pc` est $(0, 2, 1, 2)$, son déploiement aura pour suite de clefs $(1, 1, 1, 1, 1)$.

Ecrire une fonction `struct cell *deployer(struct cell *pc)`. En supposant que toutes les clefs de `pc` sont positives ou nulles, cette fonction doit renvoyer l'adresse d'une nouvelle liste formant le déploiement de `pc`, tout en libérant intégralement l'espace mémoire alloué pour les cellules de `pc`. Elle peut modifier temporairement les clefs de `pc`, pourvu que cette dernière contrainte soit respectée.

Casse-tête final

Exercice 7

Etant donné un texte découpé en lignes et un entier n, on dit que ce texte forme une *échelle de rang* n si chaque ligne autre que la première peut être obtenue à partir de la ligne précédente en modifiant exactement n caractères de celle-ci. Par exemple, le texte suivant (dû à Lewis Carroll) forme une échelle de rang 1 :

```
head
heal
teal
tell
tall
tail
```

Notez que toutes les lignes d'une échelle sont de même longueur, et qu'un texte dont toutes les lignes sont égales forme une échelle de rang 0.

On suppose le texte d'adresse `s` découpé en lignes comme dans l'exercice 4. Ecrire une fonction `int degre_echelle(char *s)` renvoyant `-1` si le texte d'adresse `s` n'est pas une échelle, et sinon, renvoyant son degré. Prenez le temps de bien analyser le problème posé.