Programmation en C

Examen du 30/05/2007 - 3h

Les notes de cours et de TD sont autorisées. La correction tiendra aussi compte de la qualité de rédaction, de l'effacité algorithmique et du respect des contraintes de l'énoncé.

1 Listes chaînées

On considère la représentation usuelle des listes chaînées d'entiers, à l'aide du type de structure :

```
struct cell {
  int clef;
  struct cell *suiv;
}
```

Comme d'habitude, une liste chaînée non vide est représentée par un pointeur vers sa première cellule, la liste vide par un pointeur nul.

Ecrire les fonctions suivantes. Chacune devra obligatoirement être écrite par récurrence, sans fonctions auxiliaires, et sans relire la liste. Noter que chacune fait à peine quelques lignes.

1. `int parite_nombre(struct cell *pc)`

   renvoyant `1` si `pc` contient un nombre pair de clefs, et `0` sinon.

2. `int parite_clefs(struct cell *pc)`

   renvoyant `1` si toutes les clefs de `pc` sont paires, `0` sinon.

3. `int parite_alterne(struct cell *pc)`

   renvoyant `1` si les clefs des cellules de numéro pair de `pc` sont toutes paires et si les clefs des cellules de numéro impair sont toutes impaires, `0` sinon. On numérote les cellules d'une liste à partir de `0`.

2 Justification d'un texte

Etant donnée un texte réparti sur plusieurs ligne, sa *justification gauche* est le texte obtenu en supprimant les espaces apparaissant au début de chaque ligne. Sa *justification droite* est le texte obtenu :

1. en supprimant les espaces apparaissant à la fin de chaque ligne, puis,
2. en ajoutant au début de chaque ligne qui n'est pas la plus grande le nombre d'espaces nécessaires pour chaque ligne ait la même longeur.

texte initial

```
    Nous sommes de
 l'étoffe dont sont faits
     les rêves, et notre
   petite vie est
entourée de sommeil
```

justification gauche

```
Nous sommes de
l'étoffe dont sont faits
les rêves, et notre
petite vie est
entourée de sommeil
```

justification droite

```
          Nous sommes de
l'étoffe dont sont faits
     les rêves, et notre
          petite vie est
     entourée de sommeil
```

Rappelons que dans une chaîne de caractères, chaque retour à la ligne est indiqué par une occurrence de `'\n'`.

Exercice 1.1

Ecrire une fonction `void justifier_gauche(char *s)` transformant la chaîne `s` en sa justification gauche. Le traitement devra se faire dans `s`, et en une seule lecture de `s` (pas de copie dans un tableau, pas de `strlen`). Attention aux cas limites (chaîne vide, ligne vide, etc.)

Exercice 1.2

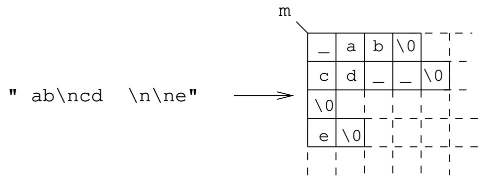
Le but de cet exercice est d'implémenter le calcul de la justification droite d'une chaîne. La constante `MAX` ci-dessous est supposée suffisamment grande pour effectuer chaque traitement demandé.

Vous devrez dans cet exercice respecter la contrainte suivante : les variables locales des fonctions des étapes 1 à 6 ne peuvent être que de type `int`. Vous ne pouvez donc dans ces fonctions ni déclarer de nouveaux tableaux, ni faire d'allocation mémoire.

Etape 1

Ecrire `int transferer(char *s, char m[MAX][MAX])`. Si `s` contient `n` lignes numérotées de `0` à `n-1`, cette fonction doit, pour chaque `i`, recopier la ligne n^o `i` de `s` sur la ligne n^o `i` de `m`, à partir de la colonne `0`. Le dernier caractère de chaque ligne devra être suivi d'un `'\0'` dans `m`.

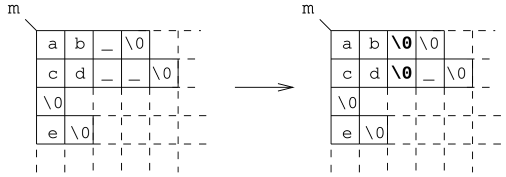
Le transfert doit se faire en une seule lecture de `s`. Cette fonction doit renvoyer le nombre total de lignes transférées.



Dans les étapes qui suivent, la matrice `m` passée à chaque fonction est toujours supposée contenir un texte stocké avec les mêmes conventions : chaque ligne stockée commence à la colonne `0` et se termine par un `'\0'`.

Etape 2

Ecrire `void tronquer(int h, char m[MAX][MAX])`. Cette fonction suppose que le texte stocké dans `m` contient `h` lignes. Elle doit supprimer les espaces à la fin de chaque ligne non vide, c'est-à-dire remplacer par `'\0'` le caractère qui suit le dernier caractère de cette ligne différent d'un espace.

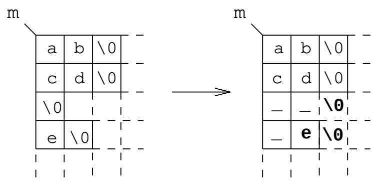


Etape 3

Ecrire `int largeur(int h, char m[MAX][MAX])`. Cette fonction suppose que le texte stocké dans `m` contient `h` lignes. Elle doit renvoyer la longueur de sa plus grande ligne.

Etape 4

Ecrire `void decaler(int h, int l, char m[MAX][MAX])` Cette fonction suppose que le texte stocké dans `m` contient `h` lignes, chaque ligne contenant au plus `l` caractères. Elle doit ajouter au début de chaque ligne le nombre d'espaces nécessaires pour qu'en fin de traitement, chaque ligne contienne exactement `l` caractères.



Etape 5

Ecrire `void recomposer(int h, char m[MAX][MAX], char *t)`. Cette fonction suppose que le texte stocké dans `m` contient `h` lignes. Le pointeur `t` est supposé pointer vers un espace mémoire suffisamment grand. Elle doit recomposer dans `t` le texte stocké dans `m`, avec tous les retours à la ligne nécessaires.

Etape 6

Ecrire `int taille(int h, char m[MAX][MAX])`. Cette fonction suppose que le texte stocké dans `m` contient `h` lignes. Elle doit renvoyer la taille totale du texte stocké en comptant un retour-chariot après chaque ligne, y compris la dernière.

Etape finale

A partir des fonctions précédentes, écrire :

```
char *justification_droite(char *s)
```

Cette fonction doit déclarer une matrice de stockage de taille `MAX`×`MAX`, transférer la chaîne `s` dans celle-ci, puis transformer le texte stocké en sa justification droite. Elle doit ensuite calculer la taille exacte du texte résultant, allouer par `malloc` un espace mémoire de cette taille dans lequel elle transférera ce texte. Elle doit enfin renvoyer l'adresse de l'espace alloué.