Examen d'informatique
L2.1 janvier 2010

2 heures

Notes de cours et de TD autorisées

1. **Complexité**
   Un tableau d'entiers de taille N représente une permutation de l'ensemble $\{1,\ldots,N\}$ si et seulement si chaque entier de $\{1,\ldots,N\}$ apparaît une et une seule fois dans le tableau.

   (a) (1 point) Écrire une fonction `int EstPermutation(int tab[],int n)` qui reçoit un tableau d'au moins n entiers et renvoie 1 si les n premières cases du tableau représentent une permutation de l'ensemble $\{1,\ldots,n\}$ et 0 sinon. Quelle est sa complexité ?

   (b) (4 points) Écrire une fonction `void Permutation(int n)` qui affiche toutes les permutations de l'ensemble $\{1,\ldots,n\}$.
   On pourra utiliser une fonction récursive `void Place(int val,int n,int T[])` effectuant le placement successif de l'entier `val` à toutes les positions libres possibles du tableau T (parmi les n premières cases) avant de d'effectuer récursivement le placement des entiers suivants.
   Quelle est la complexité de la fonction `Permutation`?

2. **Traitement de fichiers textes**
   Dans la suite, on appelle `Mot` toute suite de moins de `MAXLETTRES` caractères ne contenant pas de séparateurs (*espace, tabulation, passage à la ligne)*, et `texte` un fichier contenant une suite de `Mot` séparés par des espaces. On appelle lexique un ensemble de couples (Mot, nombre d'apparitions). Le but de l'exercice est d'associer à chaque mot d'un texte son nombre d'apparitions dans le texte, et donc de creer le dictionnaire du texte.
   Pour permettre un accès rapide aux mots, on utilise un *hachage ouvert*: les mots sont mémorisés, avec leur nombre d'apparition, dans les cellules d'une des listes d'un tableau de listes. L'indice de la liste est obtenu grâce à une fonction de hachage. Les mots d'une liste sont triées par nombre d'apparitions croissant. En utilisant le type :

```
typedef struct celmot{
 char *Mot; /*adresse de la chaine de caracteres*/
 int nombre;/* nombre d'occurences*/
 struct celmot *suivant;
} CelluleMot, *ListeMot;
```

   le lexique est un tableau formé de N `ListeMot` (`#define N 1000`), l'indice sera donc calculé modulo N.

   (a) (1 point) Écrire la fonction `int Hachage(char M[])` qui renvoie la somme

   $$\sum_{i=0}^{M[i]='\backslash 0'} (i+1)M[i] \pmod N$$

   (*fonction pas très efficace*).

   (b) (2 points) Écrire la fonction `ListeMot Allouecellule(char M[])` qui crée une nouvelle cellule. Elle utilise la place mémoire minimale pour copier M et le nombre d'occurence est égal à 1.

   (c) (2 points) Écrire une fonction `int AjouteListe(ListeMot *L, char M[])` qui ajoute le mot M à la liste triée (*L) : si M était déjà présent dans la liste on augmente son nombre d'occurences, sinon on insère une nouvelle cellule. La fonction renvoie 0 en cas d'échec, le nouveau nombre d'apparitions de M sinon.

(d) (1 point) Écrire une fonction `void InitLexique(ListeMot lexiq[])` qui place la liste vide dans chaque case du tableau de listes.

(e) (1 point) Écrire une fonction `int AjouteLexique(ListeMot lexiq[], char M[])` qui ajoute le mot M au tableau s'il n'était pas présent et augmente son nombre d'occurences sinon. La fonction renvoie 0 en cas d'echec, le nouveau nombre d'occurence de M sinon.

(f) (2 points) Écrire une fonction `int ConstruitLexique (FILE *in, ListeMot lexiq[])` qui ajoute au tableau l'ensemble des mots contenus dans le fichier manipulé grâce à `in`. La fonction renvoie le nombre de mots différents contenus dans la liste.

(g) (1 point) Écrire une fonction `int NombreMot(ListeMot lexiq[])` qui renvoie le nombre total de mots contenus dans le texte dont `lexiq` est le lexique.

(h) (1 point) Écrire une fonction `int NombreApparition(ListeMot lexiq[], char *M)` qui affiche le nombre d'apparition du mot M dans le texte dont `Lexiq` est le lexique.

(i) (2 points) Écrire une fonction `int LePlusFrequent(ListeMot lexiq[])` qui renvoie l'adresse de la cellule qui contient le mot M qui apparait le plus souvent dans le texte dont `lexiq` est le lexique.

(j) (2 points) Écrire une fonction `void AfficheLexique(FILE *out, ListeMot lexiq[])` qui place dans le fichier manipulé grâce à `out` la suite des mots présents dans le lexique et leur nombre d'apparitions. Chaque couple (mot, nombre) est écrit sur une ligne. L'ordre des mots est libre.

(k) (4 points) Écrire le bloc `main` permettant la construction du lexique d'un texte et l'écriture des couples (mots, nombre d'apparitions) dans un fichier. Les noms du fichier où lire le texte ainsi que celui du fichier où écrire les couples seront transmis comme arguments de la ligne de commande.
Exemple:
si l'exécutable a pour nom `lexique`, la commande
`$> lexique /home/zipstein/examen motdeexam`
placera dans le fichier `motdeexam` du répertoire courant la suite des mots du fichier `/home/zipstein/examen`.