Automne 2011 **Examen final - LO01**

2 H - Sans documents (sauf diagrammes de C) ni calculette

N'oubliez pas de commenter vos programmes.

Vous aurez besoin de 4 copies : une par exercice

1. Traduction Google (5 points)

On veut aider un linguiste à traduire des fichiers contenant du texte d'une langue à une autre en traduisant partiellement ces fichiers. Le but est de remplacer automatiquement les mots dont on connaît déjà la traduction au travers de fichiers contenant des collections de mots et leur traduction.

Pour cela, on dispose de **fichiers structurés** contenant des valeurs du type :

```
typedef struct traduction {
    char langueA[50];
    char langueB[50];
};
```

Chaque enregistrement contient un mot dans une langue A et le même dans une langue B sous forme de chaînes de caractères contenant au plus 49 caractères.

1.1 - : Ecrire une fonction d'entête *int recherche(char* dico, char* motA, char* motB)*; qui recherche dans un dictionnaire (dont le nom est fourni dans la chaîne de caractères représentée par la variable dico) la traduction d'un mot (dont le nom est fourni dans la chaîne de caractères représentée par la variable motA). Si ce mot existe dans le dictionnaire, sa traduction est recopiée dans la chaîne de caractères représentée par la variable motB et la fonction renvoie 1. Si ce mot n'existe pas dans le dictionnaire, motB est affectée avec la chaîne vide et la fonction renvoie 0.

1.2 - : Ecrire une fonction d'entête *int traduire(char* dico, char* mon_texte, char* traduction);* qui traduit un **fichier texte** (dont le nom est fourni dans la chaîne de caractères représentée par la variable mon_texte) en un **fichier texte** dont le nom est fourni dans la chaîne de caractères représentée par la variable traduction. On considèrera que le fichier mon_texte est prétraité et contient une suite de mots séparés par des espaces. Le fichier traduction contiendra les même mots que dans le fichier mon_texte mais en les remplaçant par leur traduction (si elle existe) trouvée dans le fichier dont le nom est fourni dans la chaîne de caractères représentée par la variable dico. Un mot dont la traduction n'existe pas dans dico est recopié tel quel dans le fichier traduction.

Vous pouvez utilisez les deux fonctions de la bibliothèque standard string.h du C :

- strcpy(str1,str2) qui recopie la chaîne de caractères str2 dans la chaîne de caractères str1.
- strcmp(str1,str2) qui renvoie 0 uniquement si les chaînes de caractères str1 et str2 sont identiques.

---

2. Gestion d'une cave à vin (5 points)

On utilise la structure suivante pour définir un vin :

```
#define MAX 100
Typedef struct {
        int nbBouteilles ;      /*nombre de bouteilles identiques */
        char nom[MAX];          /*nom du vin */
        char type ;             /*Blanc, Rouge ou Rosé */
        int annee;              /*année du vin */
} unVin ;
```

On voudrait gérer une cave à vin par l'intermédiaire d'un fichier.

2.1 – Ecrire une fonction *initMaCave(char *nomDeLaCave),* qui permet de créer le fichier correspondant au nom de la cave et demande à l'utilisateur de rentrer chacun des vins dans le fichier.

2.2 – Ecrire un fonction *selection(char *nomDeLaCave , int annee, char type)* qui affichera tous les vins du type et de l'année passés en paramètre. Attention lorsque le nombre de bouteilles est égale à 0, on ne l'affichera pas.

2.3 – Ecrire une fonction *int enleveUneBouteille(char *nomDeLaCave, char *nomDuVin)* qui permet de soustraire dans la cave "nomDeLaCave" une bouteille de vin "nomDuVin". Cette fontion retournera 1 si l'opération s'est bien passée (nom du vin correct et nombre de bouteilles supérieur à 1), 0 sinon. Attention dans cette fonction, on devra réécrire l'enregistrement dans le fichier au même endroit.

3. Calcul de distance parcourue (5 points)

On voudrait écrire une fonction qui calcule une approximation de la distance parcourue par un véhicule à partir des positions GPS recueillies lors d'un trajet. Les positions GPS sont définies dans une structure de type *unePoseGPS* et stockées dans un tableau.

Afin de calculer simplement la distance parcourue, il est nécessaire de transformer ces positions en coordonnées cartésiennes dans un système local. Le repère de référence pour la France s'appelle le Lambert 93. Il suffit ensuite d'additionner les distances euclidiennes entre chaque position en coordonnées cartésiennes.

Soient les types suivants

```
typedef struct{
    float longitude, lattitude ; /* en degré */
        float elevation ;              /* en mètres*/
    char refLong ;                     /* 'E' ou 'O' */
        char refLat  ;                 /* 'S' ou 'N' */
        }unePoseGPS ;

typedef struct{
    float x, y, z ; /* en mètres */
        }unePoseCart ;
```

3.1 - Ecrire la fonction *distance()* qui calcule la distance euclidienne entre 2 positions données en coordonnées cartésiennes.

Soit la fonction supposée connue :
*void changeCoord(unePoseGPS pgps, unePoseCart *pl93)* qui transforme une position GPS en une position cartésienne en Lambert93.

3.2 - Ecrire la fonction *float distanceParcourue(unePoseGPS tabP [], int nbP)* qui calcule et retourne une approximation de la distance parcourue à partir des nbP positions GPS contenues dans le tableau tabP.

Soit la fonction supposée connue :
*void saisiePoseGPS(unePoseGPS *pgps)* qui saisit une position GPS et la stocke dans la mémoire pointée par pgps.

3.3 - Ecrire la fonction *saisieParcoursGPS()* qui remplit un tableau de positions GPS (MAX = 10000) à l'aide de la fonction *saisiePoseGPS()*et retourne le nombre de positions que contient ce tableau.

3.4 Ecrire le programme principal qui saisit un parcours de véhicule et affiche la distance parcourue.

Rappel : la fonction *sqrt()*définie dans *math.h* permet de calculer la racine carrée d'un réel.

Remarque : une attention particulière sera portée à la définition de l'entête des fonctions des questions 3.1 et 3.3 et à l'utilisation correcte des fonctions supposées connues.