I2010 : langage C (17)

Exercice récapitulatif

Imaginons un capteur de température installé devant l'entrée de l'Institut Paul Lambin. Ce capteur mesure la température extérieure sur commande et écrit le résultat de ses mesures dans un fichier binaire.

Nous vous demandons d'écrire un programme qui « analyse » un de ces fichiers.

Structures C

Une mesure est représentée en C par le type `Temperature` :

```
typedef struct {
   int id;          // identifiant
   double temp;     // relevé de température
   char* message;   // message d'alerte
} Temperature ;
```

Un fichier de mesures est structuré en mémoire RAM à l'aide du type `TempArray` qui modélise un tableau de `Temperature` de longueur *length*.

```
typedef struct {
   Temperature* tab;  // tableau de mesures
   int tailleLog;     // taille logique de tab
   int taillePhys;    // taille physique de tab
} TempArray;
```

Ces deux structures sont définies dans le fichier *temperature.h*.

Fonctions à construire

Le fichier *temperature.h* contient les signatures des fonctions ci-dessous que nous vous demandons d'implémenter dans le fichier *temperature.c*. Lorsque vous construirez ces méthodes, vous ne devez pas vérifier leurs paramètres, vous devez supposer qu'ils sont valides.

- **`TempArray readTemperatures ();`**

  Cette fonction renvoie un tableau de mesures qu'elle a lues à l'entrée standard, en suivant le format suivant : l'identifiant et la température séparés par un caractère *whitespace* sur la première ligne et le message d'alerte (éventuellement vide) sur la seconde ligne.

- **`bool writeTemperature (FILE* f, Temperature temp);`**

  Cette fonction est un *sérialiseur* : elle écrit une mesure dans un fichier binaire. Pour ce faire, elle prend en paramètre un fichier binaire ouvert en écriture et une structure `Temperature`. Elle écrit cette mesure dans le fichier de la manière suivante : écriture de l'id, écriture de la température, ensuite écriture du nombre n de caractères du message d'alerte (sans le caractère '\0'), suivi de l'écriture des n caractères de ce message.

- **`bool readTemperature (FILE* f, Temperature* temp);`**

  Cette fonction est un *désérialiseur* : elle lit une mesure qui a été écrite dans un fichier binaire à l'aide de la fonction `writeTemperature`. Pour ce faire, elle prend en paramètre un fichier binaire ouvert en lecture et un pointeur vers une structure `Temperature`. Elle lit une mesure dans le fichier de la manière suivante : lecture de l'id, lecture de la température, ensuite lecture du nombre n de caractères du message d'alerte, suivi de la lecture des n caractères de ce message.

- **`bool writeFile (char* path, TempArray t);`**

  Cette fonction prend en paramètre le chemin d'un fichier *path* et un tableau de mesures t. Elle sauve les mesures du tableau t dans le fichier. Si le fichier *path* n'existe pas, elle le crée ; sinon elle ajoute les nouvelles données à la suite de celles qui s'y trouvent déjà.

- **`bool readFile (char* path, TempArray* t);`**

  Cette fonction prend en paramètre le chemin d'un fichier de mesures *path* et un pointeur t vers un tableau de mesures. Elle initialise le tableau pointé par t de telle manière que t contienne toutes les mesures du fichier de températures dans le même ordre.

- **`void print (TempArray t);`**

  Cette fonction prend en paramètre un tableau de mesures. Elle affiche sur *stdout* les mesures contenues dans le tableau en respectant le format ci-dessous :

```
NBR OF MEASURES: 5
 2 ==>   2
 3 ==>   5   [unreliable: significant variance]
 4 ==>  -3
 7 ==>  -5
10 ==>  15
```

- **`int nbrPos (TempArray t);`**

  Cette fonction prend en paramètre un tableau de mesures. Elle renvoie le nombre de températures plus grandes ou égales à 0. Par exemple, elle renvoie 3 avec le tableau d'exemple ci-dessus.

- **`void filter (TempArray t1, TempArray* t2);`**

  Cette fonction prend en paramètre un tableau de mesures t1 et un pointeur t2 vers un tableau de mesures. Elle initialise le tableau référencé par t2 de telle manière que t2 contienne toutes les mesures de températures positives de t1 dans le même ordre et rien que celles-là. Par exemple, si on applique cette fonction à partir des mesures de notre exemple et qu'on imprime le paramètre t2 résultant à l'aide de la fonction « *print* », on devrait obtenir le résultat suivant :

```
NBR OF MEASURES: 3
 2 ==>   2
 3 ==>   5   [unreliable: significant variance]
10 ==>  15
```

- **`void sort_temperatures (TempArray* t);`**

  Cette fonction prend en paramètre un pointeur vers un tableau de mesures t. Elle trie les mesures de t par ordre croissant. Pour ce faire, définissez une fonction de comparaison de deux mesures (comparaison des températures et, en cas d'égalité, des identifiants) et passez-la en paramètre à la fonction standard *qsort*.

Fonction supplémentaire

De plus, définissez la signature et implémentez la fonction `freeMem` qui libère la mémoire d'une variable `TempArray`, met *tailleLog* et *taillePhys* à 0 et *tab* à NULL. Ajoutez la signature de cette fonction au fichier *temperature.h*.

Autres fichiers

Ecrivez un fichier *measures.c* qui contiendra un programme pour créer le fichier binaire de mesures saisies à l'entrée standard et un fichier *analyzer.c* qui contiendra un programme vous permettant de tester vos fonctions sur les mesures lues dans le fichier binaire créé par *measures.c*.

Pour faciliter vos tests, créez un fichier text de mesures que vous redirigerez vers l'entrée standard de votre programme (ex : `./measures < mesures.txt`).

Makefile

Pour finir, nous vous demandons d'écrire un fichier *makefile* qui vous permettra de générer les deux fichiers exécutables `measures` et `analyzer`.

Notez que, bien que nous ne les ayons pas fournies, il est fortement conseillé d'ajouter de nouvelles fonctions au module *temperature* afin d'enrichir celui-ci, mais aussi de faciliter l'implémentation et d'améliorer la lisibilité de votre code.