NF16 – TP4 : Les arbres binaires de recherche

Le but de ce TP est de se familiariser avec les arbres binaires de recherche (ABR) et les différentes opérations nécessaires pour les manipuler. Nous souhaitons utiliser un arbre binaire de recherche pour indexer les fiches médicales des patients d'un médecin.

Enoncé :

Un médecin souhaite mettre en place un système pour stocker les fiches médicales de ses patients. L'indexation de ses fiches se fera sur la base des noms des patients (on suppose que les noms des patients sont uniques).
Pour ce faire, nous allons utiliser un ABR. Les fiches contiendront des informations sur le patient (nom, prénom, le nombre de consultations et une liste des consultations du patient) et seront triés selon le nom du patient suivant un ordre alphabétique croissant.

NB : Les noms des patients doivent être sauvegardés uniquement en minuscule ou uniquement en majuscule.

A. Structures de données :

Définir les structures de données suivantes :

- Le type **`Consultation`** associé à une structure qui comporte les champs :
  - **date** de type char* (sous le format « JJ-MM-AAAA »).
  - **motif** de type char* qui contient le motif de la consultation.
  - **niveauUrg** de type entier qui contient le niveau d'urgence de la consultation
  - **suivant,** un pointeur sur la consultation suivante.

- Le type **`Patient`** associé à une structure qui comporte les champs :
  - **nom** de type char* qui contient le nom du patient.
  - **prenom** de type char* qui contient le prénom du patient.
  - **ListeConsult**, un pointeur de type **`Consultation.`**
  - **nbrconsult** de type entier qui représente le nombre de consultations du patient.
  - **fils_gauche** un pointeur de type **`Patient.`**
  - **fils_droit** un pointeur de type **`Patient.`**

- Le type **`Parbre`** qui est un pointeur vers le type **`Patient`**

B. Fonctions à implémenter :

1. Ecrire une fonction qui permet de créer une fiche médicale d'un patient.

Le prototype de la fonction est comme suit :

```
Patient* CreerPatient(char * nm , char * pr);
```

2. Ecrire une fonction qui permet d'ajouter la fiche d'un patient à un arbre :

```
void inserer_patient(Parbre * abr, char * nm, char * pr);
```

3. Ecrire une fonction qui permet de chercher la fiche d'un patient dans un ABR :

```
Patient * rechercher_patient(Parbre * abr, char* nm);
```

4. Ecrire une fonction qui affiche la fiche médicale d'un patient (avec la liste de ses consultation) :

```
void afficher_fiche(Parbre * abr, char* nm);
```

5. Ecrire une fonction qui permet l'affichage préfixe de la liste des patients :

```
void afficher_patients(Parbre * abr);
```

6. Ecrire une fonction qui crée une nouvelle consultation :

```
Consultation * CreerConsult(char * date, char* motif, int nivu);
```

7. Ecrire une fonction qui ajoute une consultation à la fiche médicale d'un patient :

```
void ajouter_consultation(Parbre * abr, char * nm, char * date, char* motif, int
                                    nivu);
```

8. Ecrire une fonction qui supprime la fiche médicale d'un patient de l'ABR:

```
void supprimer_patient(Parbre * abr, char* nm);
```

NB : il faut prendre en considération le cas où il n'y a plus de patients dans l'ABR.

9. Pour des raisons de sécurité, le médecin souhaiterait pouvoir périodiquement copier et mettre à jour la liste de ses patients dans un autre ABR. Ecrire une fonction qui permet de réaliser cette tâche :

```
void maj(Parbre * abr, Parbre * abr2);
```

C. Interface:

Utiliser les fonctions précédentes pour écrire un programme permettant de manipuler les matrices creuses. Le programme propose un menu qui contient les fonctionnalités suivantes :

1. Ajouter un patient
2. Ajouter une consultation à un patient
3. Afficher une fiche médicale
4. Afficher la liste des patients
5. Supprimer un patient
6. Copier ou mettre à jour la liste des patients
7. Quitter

Consignes générales:

- Sources

À la fin du programme, les blocs de mémoire dynamiquement alloués doivent être proprement libérés.

L'organisation MINIMALE du projet est la suivante :

- Fichier d'en-tête tp4.h, contenant la déclaration des structures/fonctions de base,
- Fichier source tp4.c, contenant la définition de chaque fonction,
- Fichier source main.c, contenant le programme principal.

- Rapport

Votre rapport de quatre pages maximum contiendra :

- La liste des structures et des fonctions supplémentaires que vous avez choisi d'implémenter et les raisons de ces choix.
- Un exposé succinct de la complexité de chacune des fonctions implémentées.

Votre rapport et vos trois fichiers feront l'objet d'une remise de devoir sur **Moodle** dans l'espace qui sera ouvert à cet effet (un seul rendu de devoir par binôme).