Algorithmique et Structures de données

L2 2021-2022

Travaux Pratiques 4 - Liste chainée

`gcc mon_fichier.c -std=c11 -Wall -Wextra -o mon_programme`
Les exercices marqués d'une étoile sont optionnels et à faire dans un second temps.

Dans ce tp on travaille avec des listes chainées. On utilisera les définitions de types suivantes.

```
#include<stdlib.h>
typedef struct cel
{
   int valeur;
   struct cel* suivant;
} Cellule;
typedef Cellule* Liste;
```

Une liste vide sera définie par

```
Liste l=NULL;
```

Exercice 1. *Opérations de bases*

Définir les fonctions suivantes :

- `Cellule* allouerCellule(int val)` qui alloue la place pour une cellule d'une liste chainée et lui donne la valeur `val`. La fonction interrompt le processus en cas de problème lors de l'appel de `malloc`.
- `void ajouterEnTete(Liste* l,Cellule* c)` qui ajoute une cellule en tête de liste. La fonction peut être utilisée avec `allouerCellule` ainsi `ajouterEnTete(&liste,allouerCellule(4))` ajoute une cellule contenant 4 en tête de liste.
- `int estVide(Liste l)` qui renvoie 1 si la liste est vide et 0 si la liste contient au moins 1 élément.
- Définir la fonction `Cellule* extraireTete(Liste* l)` qui retire (si elle existe) la première cellule de la liste chainée et renvoie son adresse.
- `void viderListe(Liste* l,int val)` qui vide la liste et desalloue la mémoire occupée.
- `void afficherListe(Liste l)` qui affiche dans l'ordre les entiers contenus dans la liste puis passe à la ligne.

Exercice 2. *En queue*

1. Écrire la fonction `void ajouterEnQueue(Liste* l,Cellule* c)` qui ajoute une cellule en fin de liste (en queue).
2. On veut convertir une suite finie d'entiers contenue dans un tableau en une liste chainée. On a deux choix :
   — Parcourir le tableau des plus petits indices vers les plus grands et utiliser la fonction `ajouterEnQueue`.
   — Parcourir le tableau des plus grands indices vers les plus petits et utiliser la fonction `ajouterEnTete`.
   Pour chacune des deux méthodes, quelle est sa complexité en temps, si on suppose que le tableau est de taille n ? Faut-il en priviliégier une des deux ?

Exercice 3. *Insere à la place i*

1. Écrire une fonction `inserer(Liste* l,Cellule* c,int i)` qui insère une cellule en `i`-ème position de la liste. (La position 0 correspond à la tête de liste).
2. Écrire une fonction `Cellule* extraire(Liste* l,int indice)` qui extrait la cellule placée en `i`-ème position de la liste.

Exercice 4. *Miroir*

Écrire une fonction `void miroir(Liste* l)` qui inverse l'ordre des éléments de la liste l.

Exercice 5. *Liste triée*

1. Écrire une fonction `int estTriee(Liste a)` qui renvoie 1 si la liste est triée dans l'ordre croissant, 0 sinon.
2. Écrire une fonction `void insererTriee(Liste* l,int a)` qui insert le nombre `a` dans la liste qui est supposée être triée (dans l'ordre croissant). La liste doit rester triée.

Exercice 6. **Liste générique*

On souhaite maintenant créer une structure de liste accueillant non pas des entiers mais n'importe quel type. Pour cela on utilisera un pointeur de type `void*`. On déclare donc la structure ainsi :

```
#include<stdlib.h>
typedef struct celg
{
   void* valeur;
   struct celg* suivant;
} CelluleG;
typedef CelluleG* ListeG;
```

Une liste vide sera définie par

```
ListeG l=NULL;
```

1. Implanter les fonctions suivantes : `allouerCelluleG(void* a)`, `ajouterEnTete(ListeG* l,CelluleG* c)`, `estVide(ListeG l)`, `CelluleG* extraireTete(ListeG* l)`.
2. Utiliser la structure précédente pour stocker des vecteurs du plan :

   ```
   typedef struct vec
   {
      float x;
      float y;
   } Vecteur ;
   ```

   On créera une fonction `Vecteur* allouerVecteur(float x, float y)` qui alloue et renvoie un vecteur.
3. Écrire la fonction `Vecteur sommeVecteur(ListeG l)` qui renvoie la somme de tous les vecteurs de la liste `l`.