Algorithmique et Structures de données 1

L2 2021-2022

Travaux Pratiques 4

Site du cours : `https://defelice.up8.site/algo-struct.html`
Les exercices marqués de (@) sont à faire dans un second temps.
Un fichier écrit en langage C se termine conventionnellement par `.c`.
Une commande de compilation est `gcc fichier_source1.c fichier_source2.c fichier_source3.c`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall` pour demander au compilateur d'afficher plus de Warnings
- `-Wextra` pour demander au compilateur d'afficher plus de Warnings
- `-std=c11` pour compiler selon la norme C11

Exemple : `gcc -Wall fichier1.c -o monprogramme`

Dans ce TP on travaillera sur des listes chainées d'entiers à travers la structure suivante. **Respecter les noms des types et des champs**

```
typedef struct s_cellule_t
{
   int val; // valeur
   struct s_cellule_t* suiv; // suiv
} cellule_t;

// la liste vide sera  NULL
```

Exercice 1. *Opérations de base*

Écrire les fonctions de bases suivantes et testez-les.

- `cellule_t* consL(int val,cellule_t* queue)` Qui construit une nouvelle liste en ajoutant une nouvelle cellule en tête de `queue`. Renvoie l'adresse de la cellule de tête.
- `void libererL(cellule_t* liste)` qui libère la mémoire reservée à une liste.
- `void afficherL(cellule_t* liste)` qui imprime les éléments de la liste et passe à la ligne.

Exercice 2. *Opération de calcul*

1. Écrire une fonction `int longueurL(cellule_t* li)` qui renvoie la longueur de la liste `li`.
2. Écrire une fonction `sommeL` qui fait la somme des éléments positifs de la liste.
3. Écrire une fonction `estTrieeL` qui renvoie 1 si les éléments de la liste sont dans l'ordre croissant.

Exercice 3. *Opération de modification*

Écrire les fonctions suivantes en C. Ces fonctions travaillent sur un pointeur qui pointe sur la tête de la liste.

- Écrire une fonction `void ajouteEnTeteL(int val,cellule_t** li)` qui ajoute l'élément `val` en tete de liste.
- Écrire une fonction `void fusionL(cellule_t** liC,cellule_t** liS)` qui fusionne les deux listes `liC` et `liS`. Apres appel, `liS` doit être vide et `liC` doit contenir les élements de `liS` à la fin de ses propres éléments. La fonction doit utiliser un nombre d'opérations d'ordre $O(n)$ où n est la longueur de la liste `liC`.

Exercice 4. *Extraire et inserer*

- Écrire une fonction `void insererL(int val,cellule_t** li, int i)` qui insert l'élément `val` à l'emplacement `i` (i=0 en tete de liste, i=1 juste apres la première valeur de la liste, i=2 ...)

— Écrire une fonction `cellule_t* extraireL(cellule_t** li, int i)` qui extrait la cellule `i` (i=0 la première cellule i=1 la seconde cellule) de la liste et renvoie son adresse.

Exercice 5. *Inserer dans l'ordre*

Écrire une fonction `void insererTriee(int val,cellule_t** li)` qui insère l'entier `val` dans la liste. Avant et après l'insertion la liste doit être et rester triée dans l'ordre croissant.

Exercice 6. *Miroir*

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