Programmation Avancée

L2 2022-2023

Travaux Pratiques 3

Liste chaînée (LC)

Chaque fonction codée doit faire l'objet d'un ou plusieurs tests dans le main. Veiller à bien respecter les noms de fonctions et de types, et l'ordre des paramètres.

Reprendre le type `node_t` vu en cours en utilisant le type `int` pour `element` :

```c
typedef int element_t;
typedef struct node node_t;

struct node{
    element_t elem; /* it could be called data */
    struct node* next;
};

node_t* ll_add(node_t* queue, element_t elem){
    node_t* l = malloc(sizeof *l);
    l->elem = elem; /* (*l).elem = l->elem is of type element_t */
    l->next = queue; /* (*l).next = l->next is of type node_t* */
    return l;
};

int main(void){
    node_t* l = NULL;
    l = ll_add(l, 25);
    l = ll_add(l, 12);
    l = ll_add(l, -3);

    return 0;
}
```

Exercice 1. *Listes chaines*

Coder les fonctions suivantes :

1. `int ll_empty(node_t* l);` teste si la liste `l` est vide.
2. `void ll_print(node_t* l);` affiche la liste `l` à l'écran (exemple : "-3, 12, 25").
3. `void ll_print_inverse(node_t* l);` affiche la liste `l` à l'inverse à l'écran (exemple : "25, 12, -3").
4. `element_t ll_length(node_t* l);` renvoie le longeur de `l` (version recursive et version itérative).
5. `element_t ll_ithElement(int i, node_t* l);` Renvoie l'élément dans la position `i` de `l`.
6. `element_t ll_maxList(node_t* l);` Renvoie le maximum de `l`.
7. `int ll_contains_iterative(element_t e, node_t* l);` Renvoie 1 si l'élément `e` est dans `l` et 0 sinon (version recursive et version itérative).
8. `int ll_is_increasing(node_t* l);` Teste si `l` est croissante.
9. `node_t* ll_make_range(int a, int b);` Renvoie la liste des éléments de l'intervalle [a..b].
10. `int ll_sum(node_t* l);` Renvoie la somme des éléments de `l`.
11. `node_t* ll_delete_all_by_key(element_t e, node_t* l)` Supprimer tous les éléments qui sont égaux à `e`.

Coder également les fonctions permettant de manipuler le type `node_t*` comme une pile :

12. `element_t ll_top(node_t* s);` Renvoie la valeur du sommet de la pile `s` (sans dépiler).

13. `element_t ll_pop(node_t** s);` Dépile et renvoie la valeur du sommet de la pile `s`.
14. `void ll_push(element_t e, node_t** s);` Empile `e` au sommet de la pile `s`.
15. `void ll_make_empty(node_t** s);` Vide la pile `s`.

Liste doublement chaînée (LDC)

Avantages de la liste doublement chaînée (LDC) par rapport à la liste singulièrement chaînée :

— Une LDC peut être parcourue dans les deux sens, en avant et en arrière.
— On peut rapidement insérer un nouveau nœud avant un nœud donné.
— Dans une liste singulièrement chaînée, pour supprimer un nœud, un pointeur vers le nœud précédent est nécessaire. Pour obtenir ce nœud précédent, il faut parfois parcourir la liste. Dans LDC, nous pouvons obtenir le noeud précédent en utilisant le pointeur précédent.

Désavantages de LDC par rapport à la LC :

— Chaque nœud de LDC nécessite un espace supplémentaire pour un pointeur précédent.
— Toutes les opérations nécessitent le maintien d'un pointeur précédent supplémentaire.

Reprendre le type `nodeDC_t` vu en cours en utilisant le type `int` pour `element` :

```
typedef struct dl_node dl_node_t;

struct dl_node{
  element_t elem;
  struct dl_node* prev;
  struct dl_node* next;
};

typedef struct {
    dl_node_t* first;
    dl_node_t* last;
} dl_head_t;
```

Exercice 2. *Listes doublement chaînées et files*

1. `dl_head_t convert(node_t* l);` Copie le contenu de la liste `l` dans une nouvelle (`dl_head_t (convert(NULL)` permet de créer une liste doublement chaînée vide).
2. `int dl_empty(dl_head_t f);` Teste si la file `f` est vide.
3. `element_t dl_top(dl_head_t f);` Renvoie la valeur du début de la file `f` (sans la retirer).
4. `element_t dl_pop(dl_head_t *f);` Retire et renvoie la valeur du début de la file `f`.
5. `void dl_append(element_t e, dl_head_t *f);` Ajoute `e` à la fin de la file `f`.