CS-119(a) – ICC-C Série 11

2024-05-07

Pendant cette série nous allons implémenter une liste chaînée d'entiers. Comme dans le cours, nous allons utiliser les deux types suivants :

```
typedef struct _cell
{
    int contenu;
    struct _cell *next;
} cell_t;

typedef struct _list
{
    cell_t *head;
    cell_t *last;
} list_t;
```

Pour tester notre implémentation, rien de mieux que d'afficher la liste. Pour ce faire, vous pouvez utiliser cette fonction :

```
void afficher_liste(const list_t *plist)
{
    printf("[");
    for (cell_t *iterator = plist->head;
         iterator != NULL;
         iterator = iterator->next) {
        if (iterator == plist->head)
            printf("head(%d)", iterator->contenu);
        else if (iterator == plist->last)
            printf("->last(%d)", iterator->contenu);
        else
            printf("->(%d)", iterator->contenu);
    }
    printf("]\n");
}
```

Nous pouvons aussi définir la liste vide comme une constante globale :

```
const list_t liste_vide = {NULL, NULL};
```

Exo1 Insertion en tête de liste

Écrivez une fonction avec la signature suivante :

```
void insert_head(list_t *plist, int valeur);
```

qui prend une liste et une valeur, et la met en tête de la liste. Il faut bien mettre à jour les pointeurs `head` et `last`.

Pour tester votre code, insérez les chiffres de `1` à `5` dans la liste et ensuite affichez la liste. Vous devriez voir la sortie suivante (pourquoi ?) :

```
[head(5)->(4)->(3)->(2)->last(1)]
```

Exo2 Effacer la tête de la liste

Écrivez une fonction

```
int delete_head(list_t *plist)
```

qui efface l'élément se trouvant en tête de la liste et qui retourne 1 si l'opération a réussi, ou 0 si la liste était vide. Souvenez-vous de libérer la mémoire occupée par la cellule qu'on enlève.

En appliquant cette opération à la liste définie à l'exercice précédent vous devriez obtenir la liste

```
[head(4)->(3)->(2)->last(1)]
```

Pile Bravo, vous venez d'implémenter une pile ! La fonction `insert_head` effectue l'opération "push", et la fonction "pop" pourrait être implémentée comme

```
int pop(list_t *plist, int *valeur)
{
    if (plist->head != NULL)
    {
        *valeur = plist->head->contenu;
    }
    return delete_head(plist);
}
```

Exo3 Insertion après un élément

Écrivez une fonction

```
int insert_after(list_t *plist, cell_t *where, int valeur);
```

qui prend une liste et un pointeur vers un élément de la liste et insère un nouvel élément juste après celui-ci. Attention au cas spécial quand `where` est égal au dernier élément de la liste `plist->last`.

En utilisant la fonction que vous venez d'écrire, insérez un `7` en deuxième position et un `10` en dernière position pour obtenir

```
[head(4)->(7)->(3)->(2)->(1)->last(10)]
```

Exo4 Effacer après un élément

Écrivez une fonction

```
int delete_after(list_t *plist, cell_t *where)
```

qui prend une liste et un pointeur vers un élément de la liste et qui efface l'élément se trouvant juste après cet élément. La fonction doit retourner 1 si l'opération a réussi, ou 0 si la liste est vide ou si le pointeur est `NULL`. Souvenez-vous de libérer la mémoire occupée par la cellule qu'on enlève.

Effacez ainsi le troisième élément de notre liste. Vous devriez obtenir la liste

```
[head(4)->(7)->(2)->(1)->last(10)]
```

Exo5 Chercher

Écrivez une fonction

```
cell_t *find_first(const list_t *plist, int valeur);
```

qui retourne un pointeur vers le premier élément de la liste contenant la valeur donnée. Si aucun tel élément n'existe, la fonction retournera `NULL`.

Vérifiez que l'élément `-1` n'est pas dans la liste. Ensuite cherchez l'élément `1` et modifiez-le pour y mettre la valeur `13`.

La liste devrait maintenant être

```
[head(4)->(7)->(2)->(13)->last(10)]
```

Exo6 Nettoyage

Écrivez une fonction

```
void delete_list(list_t *plist);
```

qui efface toute la liste en libérant la mémoire de chaque élément. Effacez votre liste et affichez-la. Vous devriez voir

```
[]
```