Exercices: Série 10 - Listes chaînées - ICC-C 2025-2026

1. Introduction

Dans cette série, vous allez implémenter tous les détails d'une liste chaînée content des entiers. Nous utiliserons les structures suivantes, que nous avons vues en cours :

```
typedef struct cell {
  int value;
  struct cell *next;
} cell_t;

typedef struct intlist {
  cell_t *head;
  cell_t *last;
} intlist_t;
```

On va devoir tester les différentes fonctions manipulant les listes. Pour faire ça sans trop de difficultés, définissez les fonctions suivantes :

```
void assert_list_empty(const intlist_t *list) {
  assert(list->head == NULL);
  assert(list->last == NULL);
}

void assert_list_contains(const intlist_t *list, size_t size,
    const int expected_values[]) {
  if (size == 0) {
    assert_list_empty(list);
  } else {
    cell_t *current = list->head;
    for (size_t i = 0; i < size - 1; i++) {
      assert(current->value == expected_values[i]);
      current = current->next;
      assert(current != NULL);
    }
    assert(current->value == expected_values[size - 1]);
    assert(current->next == NULL);
    assert(list->last == current);
  }
}
```

Elles vérifient que toute la structure interne d'une liste est bien correcte, qu'elle contient `size` éléments, et que les éléments sont égaux aux éléments du tableau `expected_values`. `assert_list_empty` fait la même chose pour une liste vide.

Par exemple, étant donnée une `intlist_t list`, vous pouvez vérifier qu'elle contient exactement les éléments $(4321, 5, 123, 56)$, dans cet ordre, avec

```
const int expected[] = { 4321, 5, 123, 56 };
assert_list_contains(&list, 4, expected);
```

N'oubliez pas d'utiliser les options `-fsanitize=...` ! Notamment pour vous assurer que vous n'utilisez pas de la mémoire détruite, ni n'introduisez de fuite de mémoire.

2. Créer et détruire

Définissez une fonction `create_list()` qui crée une liste vide. Définissez aussi une fonction `free_list(intlist_t *list)` qui libère toute la mémoire d'une liste donnée.

Écrivez un test unitaire (élémentaire) pour `create_list()`.

3. Ajouter au début

Définissez une fonction

```
void list_add_first(intlist_t *list, int value)
```

qui ajoute un élément de valeur `value` au *début* de la liste (côté tête/head).

Testez-la. C'est-à-dire, écrivez des tests unitaires pour cette fonction.

4. Retirer au début

Définissez une fonction

```
void list_remove_first(intlist_t *list)
```

qui retire le premier élément de la liste, c'est-à-dire sa tête.

Testez-la. Vous aurez besoin d'utiliser `list_add_first` dans votre test avant de pouvoir retirer des éléments. C'est acceptable puisque vous avez déjà testé `list_add_first`.

5. Chercher

Définissez une fonction

```
cell_t *list_find(const intlist_t *list, int lookup)
```

qui cherche la première occurrence de la valeur `lookup` dans la liste. Si on trouve une cellule avec cette valeur, la fonction renvoie son adresse. Sinon, la fonction renvoie `NULL`.

Testez-la (bon, vous avez compris, maintenant, on ne vous le dira plus).

6. Ajouter après une cellule donnée

Définissez une fonction

```
void list_add_after(intlist_t *list, cell_t *where, int value)
```

qui ajoute la valeur `value` juste après la cellule `where`.

Si `where == NULL`, à votre avis, où devrait-on ajouter la valeur ? Au début ou à la fin ?

7. Retirer après une cellule donnée

Définissez une fonction

```
void list_remove_after(intlist_t *list, cell_t *where)
```

qui retire la cellule juste après la cellule `where`.

`where` peut-elle être égale à `list->last` ?

Si `where == NULL`, à votre avis, quelle cellule devrait-on retirer ?

8. Ajouter à la fin

Définissez une fonction

```
void list_add_last(intlist_t *list, int value)
```

qui ajoute la valeur donnée à la *fin* de la liste (côté `last`).

Pouvez-vous l'écrire en *une seule ligne* ? Si oui, vous n'avez pas besoin de la tester.

9. Afficher

Définissez une fonction

```
void show_list(const intlist_t *list)
```

qui affiche tous les éléments d'une liste à l'écran.

Comme c'est une fonction qui affiche à l'écran, on ne peut pas la tester avec des tests unitaires. Servez-vous-en dans une petite fonction `main()` qui ajoute quelques éléments à une liste avant de l'afficher.