Cours de programmation séquentielle

Liste simplement chaînée

1 Buts

— Utilisation de liste chaînée.
— Allocation dynamique de mémoire.
— Utilisation pointeurs de fonctions.

2 Énoncé

Lors du travail pratique sur la structure `vector`, vous avez implémenté une librairie de tableaux dont la taille pouvait varier et dont la structure de données sous-jacente était un tableau dynamique.

Dans ce travail, vous allez implémenter une librairie avec les mêmes fonctionnalités, mais où la différence est la structure de données sous-jacente. En effet, ici, vous utiliserez une liste simplement chaînée.

Notre liste chaînée d'entiers est définie par

```
typedef int type;

typedef struct _element {
   type data;
   struct _element* next;
} element;

typedef element* lst_vector;
```

Chaque élément de notre liste, contient des données (`data`) et un pointeur vers l'élément suivant.

Dans ce travail pratique, vous allez implémenter une structure `lst_vector`, qui est un tableau dynamique performant pour ajouter/enlever des éléments en tête, mais moins performant pour lire les éléments (contrairement aux vecteurs que vous avez déjà implémentés). Nous pourrions créer une structure complètement générique à l'aide du type `void *`, mais cela demanderait de gros efforts qui ne seront pas forcément récompensés. Nous "émulerons" la généricité à l'aide d'un `typedef`.

2.1 La structure `lst_vector`

Puis il faudra implémenter les fonctions suivantes:

1. La fonction `lst_vector_init(lst_vector *v)` qui initialise un vecteur vide.
2. Une fonction `lst_vector_length(lst_vector *v, int *length)` qui stocke la longueur d'un vecteur dans `length`.
3. Une fonction `lst_vector_push(lst_vector *v, type element)` qui ajoute `element` au début du vecteur (le fonctionnement ici est différent de `vector`).
4. La fonction `lst_vector_pop(lst_vector *v, type *element)` qui retire le premier élément du vecteur passé en argument et le stocke dans `element` (ici aussi le fonctionnement ici est différent de `vetor`).
5. La fonction `lst_vector_set(lst_vector *v, int index, type element)` qui assigne la `index`-ème valeur du vecteur à la valeur de `element`.
6. La fonction `lst_vector_get(lst_vector *v, int index, type *element)` qui copie le `index`-ème élément du vecteur dans `element`.
7. La fonction `lst_vector_remove(lst_vector *v, int index)` qui retire le `index`-ème élément du vecteur.
8. La fonction `lst_vector_insert(lst_vector *v, type element, int index)` qui insère `element` au `index`-ème indice du vecteur.
9. La fonction `lst_vector_empty(lst_vector *v)` qui vide un vecteur et libère la mémoire.

Puis implémenter également deux fonctions un peu plus complexes syntaxiquement.

10. La fonction `lst_vector_map(lst_vector *v, type (*f)(type), lst_vector *rhs)` qui itère sur tous les élément du vecteur `v`, leur applique la fonction `f`, et stocke le résultat dans `rhs`.
11. La fonction `lst_vector_filter(lst_vector *v, bool (*f)(type), lst_vector *rhs)` applique le prédicat `f` sur tous les éléments d'un vecteur et stocke ceux qui le satisfont dans le vecteur `rhs`.

Afin d'utiliser les fonctions `lst_vector_map()` et `lst_vector_filter()` vous devez écrire deux fonctions. La première, `type square(type elem)`, calculera le carré d'un élément. La seconde, `bool is_even(type elem)`, vérifiera si `elem` est pair. Vous pouvez implémenter d'autres fonctions si vous le souhaitez.

2.2 La gestion des erreurs

Chacune des fonctions ci-dessus peut échouer: une allocation mémoire pourrait échouer, on pourrait tenter d'enlever un élément inexistant, ... Afin de gérer ces erreurs au mieux, toutes les fonctions doivent retourner un code d'erreur:

```
typedef enum error_code {
    ok, out_of_bounds, memory_error, uninitialized
} error_code;
```

Afin de vous aider à débugger ou à vérifier les problèmes possibles implémentez une fonction qui lit le code d'erreur et retourne un message d'erreur.

2.3 Tests

Adaptez si besoin les tests fournis dans `c_vecror_ci`, dont le dépôt se trouve à l'adresse https://githepia.hesge.ch/programmation_sequentielle/travaux_pratiques/c_lang/c_vector_ci, pour pouvoir tester automatiquement votre nouvelle librairie.