Structures de données
TP2

Introduction

Dans la suite des TPs de structure de données, vous allez développer une bibliothèque en C dans la quelle vous implémenterez :

- des structures de données :
  - Tableau dynamique
  - Liste doublement chaînée
  - Pile et File
  - ...
- des fonctions pour mesurer les performances de vos structures de données :
  - Mesure du temps d'exécution
  - Mesure du nombre d'allocations mémoire
  - Mesure de la quantité de mémoire allouée

Vous allez également développer des codes de test basés sur plusieurs algorithmes de tri pour évaluer les performances de vos implémentations.

Et vous devez écrire le `makefile` pour la compilation automatique de votre projet.

1 Tableau dynamique

Dans ce TP, vous allez développer un tableau dynamique. C'est une structure de données qui permet :

- de stocker un ensemble de n données ;
- d'accéder aux données par un index ;
- d'ajouter et de supprimer dynamiquement des données.

Classe C++ : `Vector`

Pour développer votre structure de tableau dynamique, vous allez vous inspirer de la classe C++ « `Vector` » (documentation ici). Les principales fonctions de la classe Vector que nous voulons reproduire sont :

- `vector::at`
- `vector::erase`
- `vector::insert`
- `vector::push_back`
- `vector::pop_back`
- `vector::clear`
- `vector::empty`
- `vector::size`
- `vector::capacity`

Pour éviter d'effectuer trop souvent des allocations mémoire lors des insertions et de suppressions de données dans le tableau, la classe C++ `Vector` utilise un stratégie d'anticipation en allouent à l'avance plus de mémoire que nécessaire. Vous aller implémenter un mécanise similaire dans votre structure de tableau dynamique.

2 Travaux préalable

2.1. Créez votre répertoire de travail, nommez le « `i3_in9_lib` ».

2.2. Recopiez dans votre répertoire de travail les fichiers :
« `makefile` » : contient les instructions pour la compilation automatique du projet ;
« `test_vector.c` » : contient la fonction *main* ;
« `vector.c` » : contient les fonctions de la structure du tableau dynamique ;
« `vector.h` » : contient la définition de la structure et les prototype des fonctions ;
« `README` » .

2.3. Éditez le fichier nommé « `README` ». Dans ce fichier, indiquez vos noms et prénoms.

2.4. Ouvrez un terminal dans votre répertoire de travail.

(a) Dans le terminal, tapez la commande `make`. Cette commande permet de compiler automatiquement le projet, vous ne devriez avoir que des `warning` indiquant que les paramètres des fonctions ne sont pas utilisés et qu'il manque des `return` dans des fonctions.
la compilation génère le fichier exécutable « `test_vector` »

(b) Exécutez le fichier « `test_vector` » avec la commande `./test_vector`

(c) Pour nettoyer votre répertoire de tous les fichiers générés lors de la compilation vous pouvez exécuter la commande : `make clean`

2.5. Vous devez, tout au long des TPs, **ajouter des commentaires dans vos codes**, vous devez également ajouté des commentaire sur les codes déjà fournie : sur la définition de la structure, sur les prototypes des fonctions, *etc.*

2.6. Vous devez, tout au long des TPs, **ajouter des tests sur les arguments passés aux fonctions** : vérifier qu'un pointeur n'est pas `NULL`, vérifier qu'un indice de tableau est bien dans la plage défini du tableau, *etc.*

3 Implémentassions - Tableau dynamique de double

Dans une premier temps, vous allez développer une structure de tableau **dynamique simple** (sans mécanisme pour réduire le nombre d'allocation mémoire) pour stocker des `double` et vous ne vous soucierez pas pour le moment de la librations de la mémoire des données stockées.

3.1. Structure pour votre tableau dynamique.

(a) Dans le fichier « `vector.h` » complétez la structure nommée « `struct_vector` », elle doit contenir :
— Une variable pour stocker le nombre d'élément stocker dans la structure
— Un pointeur de `double`.

(b) Ajoutez des commentaires sur les deux lignes de `typedef` (doc. ici) situées au dessous de la déclaration de la structure pour expliquer à quoi servent-elles ?

3.2. Fonctions pour votre tableau dynamique.

(a) Complétez la fonction `p_s_vector vector_alloc(size_t n);` qui alloue et retourne votre structure. Le tableau dynamique contient `n` `double` près initialisés à 0.0.

(b) Complétez la fonction `void vector_free(p_s_vector p_vector);` qui libère votre structure.

(c) Complétez la fonction `void vector_set(p_s_vector p_vector, size_t i, double v);` qui affecte la valeur `v` à l'index `i` de votre tableau dynamique et la fonction `void vector_get (p_s_vector p_vector, size_t i, double *pv);` qui écrit dans le pointeur `pv` la valeur de l'index `i`.

Ces fonctions sont les équivalents de la méthode `vector::at` de la classe C++ `Vector`.

(d) Complétez la fonction `void vector_insert(p_s_vector p_vector, size_t i, double v);` qui insert une nouvelle donnée à l'index `i` de votre tableau dynamique.

Cette fonction est l'équivalent de la méthode `vector::insert` de la classe C++ `Vector`.

(e) Complétez la fonction `void vector_erase(p_s_vector p_vector, size_t i);` qui supprime la donnée à l'index `i` de votre tableau dynamique.

Cette fonction est l'équivalent de la méthode `vector::erase` de la classe C++ `Vector`.

(f) Complétez la fonction `void vector_push_back(p_s_vector p_vector, double v);` qui insert une nouvelle donnée à la fin de votre tableau dynamique.

Cette fonction est l'équivalent de la méthode `vector::push_back` de la classe C++ `Vector`.

(g) Complétez la fonction `void vector_pop_back(p_s_vector p_vector);` qui supprime la dernière donnée de votre tableau dynamique.

Cette fonction est l'équivalent de la méthode `vector::pop_back` de la classe C++ `Vector`.

(h) Complétez la fonction `void vector_clear(p_s_vector p_vector);` qui supprime toutes les données de votre tableau dynamique.

Cette fonction est l'équivalent de la méthode `vector::clear` de la classe C++ `Vector`.

(i) Complétez la fonction `int vector_empty(p_s_vector p_vector);` qui retourne un entier non-nul si votre tableau dynamique est vide et zéros sinon.

Cette fonction est l'équivalent de la méthode `vector::empty` de la classe C++ `Vector`.

(j) Complétez la fonction `size_t vector_size(p_s_vector p_vector);` qui retourne le nombre d'élément stocker dans le tableau dynamique

Cette fonction est l'équivalent de la méthode `vector::size` de la classe C++ `Vector`.

3.3. Dans le fichier « `test_vector.c` » écrivez des fonctions de teste unitaire pour tester toutes vos fonctions de votre vecteur.

3.4. Testez en exécutant la commande `make test_vector` dans votre répertoire de travail.