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 Travaux préalable

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

1.2. Dans votre répertoire de travail, créez un fichier nommé « `README` ». Dans ce fichier, indiquez vos noms et prénoms.

1.3. Dans votre répertoire de travail, créez un fichier nommé « `makefile` ». Vous le laisserez vide pour le moment.

2 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.

2.1 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.

2.1. Créez dans votre répertoire de travail les fichiers
— « `vector_v1_double.h` » ;
— « `vector_v1_double.c` ».
(a) Ajoutez dans le fichier « `vector_v1_double.h` », les instruction de précompilation pour sécuriser votre fichier contre les doubles inclusions.
(b) Ajoutez dans le fichier « `makefile` », les lignes pour compiler le fichier « `vector_v1_double.c` » et obtenir le fichier « `vector_v1_double.o` ».
(c) Testez en exécutant la commande `make vector_v1_double.o` dans votre répertoire de travail.

2.2. Structure pour votre tableau dynamique.
(a) Déclarez dans le fichier « `vector_v1_double.h` » une structure, que vous nommerez « `struct_vector_v1_double` » et qui contient :
— Une variable pour stocker le nombre d'élément stocker dans la structure
— Un pointeur de `double`.
(b) Utilisez la commende C `typedef` (doc. ici) pour redéfinir votre `struct struct_vector_v1_double` en `s_vector_v1_double` et pour définir le type `p_s_vector_v1_double` qui est un pointeur sur la structure de `s_vector_v1_double`.

2.3. Fonctions pour votre tableau dynamique.
(a) Écrivez la fonction `p_s_vector_v1_double vector_v1_double_alloc(size_t n);` qui alloue et retourne votre structure. Le tableau dynamique contient `n` `double` près initialisés à 0.0.
(b) Écrivez la fonction `void vector_v1_double_free(p_s_vector_v1_double p_vector);` qui libère votre structure.
(c) Écrivez la fonction `void vector_v1_double_set(p_s_vector_v1_double p_vector, size_t i, double v);` qui affecte la donnée `pobjet` à l'index `i` de votre tableau dynamique et la fonction `double get(p_s_vector_v1_double p_vector, size_t i);` qui retourne la donnée de l'index `i`.
Ces fonctions sont les équivalents de la méthode `vector::at` de la classe C++ `Vector`.

(d) Écrivez la fonction `void vector_v1_double_insert(p_s_vector_v1_double 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) Écrivez la fonction `void vector_v1_double_erase(p_s_vector_v1_double 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) Écrivez la fonction `void vector_v1_double_push_back(p_s_vector_v1_double 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) Écrivez la fonction `void vector_v1_double_pop_back(p_s_vector_v1_double 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) Écrivez la fonction `void vector_v1_double_clear(p_s_vector_v1_double 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) Écrivez la fonction `int vector_v1_double_empty(p_s_vector_v1_double 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) Écrivez la fonction `size_t vector_v1_double_size(p_s_vector_v1_double 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`.

2.4. Fichier de test

(a) Créez le fichier « `test_vector_v1_double.c` ».

(b) Ajoutez dans ce fichier le code minimum pour la fonction `int main(int argc, char *argv[])`.

(c) Ajoutez dans le fichier « `makefile` », les lignes pour compiler le fichier « `test_vector_v1_double.c` » obtenir le fichier « `test_vector_v1_double.o` »

(d) Testez en exécutant la commande `make test_vector_v1_double.o` dans votre répertoire de travail.

(e) Ajoutez dans le fichier « `makefile` », les lignes pour effectuer le linkage de « `test_vector_v1_double.` « `vector_v1_double.o` » et obtenir le fichier « `test_vector_v1_double` »

(f) Testez en exécutant la commande `make test_vector_v1_double` dans votre répertoire de travail.

(g) Dans le fichier « `test_vector_v1_double.c` » écrivez des fonctions de teste unitaire pour tester toute vos fonctions de votre vecteur.