Structures de données
TP3

Introduction

Dans le TP précédant, vous avez développez en C un implémentation simple de la classe C++ « `Vector` » (documentation ici) permettant de stocker uniquement des `double`.

Dans ce TP, vous allez développer des testes pour évaluer les performances, en temps et en mémoire, de votre implémentation.

1 Évaluation des performances

Pour évaluer les performances d'un tableau dynamique, vous mesurerez :
- le temps d'exécution ;
- le nombres d'allocation mémoire ;
- la quantité moyenne de mémoire alloué.

Dans les scénarios suivant :
- l'ajout et la suppression d'éléments à des indexes aléatoire ;
- l'ajout et la suppression d'éléments en tête et en queue ;
- l'accès aléatoires en lecture et écriture ;
- l'accès séquentielle en lecture et écriture ;
- tri des données

2 Travaux préalable

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

2.2. Recopiez dans votre répertoire de travail « `i3_in9_lib` » les fichiers :
- « `makefile` » : contient les nouvelles instructions pour la compilation automatique du projet ;
- « `test_random.c` » : contient la fonction *main* pour les testes unitaires des fonctions de générations aléatoires ;
- « `random.c` » : contient les fonctions de générations aléatoires ;
- « `random.h` » : contient la définition des prototype des fonctions de générations aléatoires ;
- « `bench_vector.c` » : contient la fonction *main* pour les testes de performances de la structure du tableau dynamique.

2.3. 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 les fichiers exécutables suivant : « `test_vector` », « `test_random` » et « `bench_vector` »

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

2.4. 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.*

2.5. 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 Générateur aléatoire

Dans une premier temps, vous allez développer un ensemble de fonctions pour générer des nombres et chaîne aléatoire.

En C, la fonction permettant de générer des nombres aléatoires est la fonction `int rand(void);`, elle est défini dans `#include <stdlib.h>`. Cette fonction génère des nombres pseudo-aléatoire, à chaque appel, elle retourne un nombre entier compris entre 0 et `RAND_MAX` (inclus).

Le générateur de nombre étant pseudo-aléatoire, il fournie à chaque exécution la même séquence de nombres aléatoires. Pour éviter cela, il est courant d'initialiser la grène du générateur de nombre aléatoire avec la date lors de l'exécution du programme. Pour cela, vous utiliserez une seule fois au début de votre fonction `main` les fonction suivante :

— `void srand(int start)` (documentation ici)
— `int time(int*)` (documentation ici)

3.1. Complétez dans le fichier « `test_random.c` » :

(a) La fonction `double random_double(double a, double b);` qui retourne un nombre aléatoire de type `double` compris entre `a` inclus et `b` exclu. Écrivez un code de test dans « `test_random.c` ».

(b) La fonction `float random_float(float a, float b);` qui retourne un nombre aléatoire de type `float` compris entre `a` inclus et `b` exclu. Écrivez un code de test dans « `test_random.c` ».

(c) La fonction `size_t random_size_t(size_t a, size_t b);` qui retourne un nombre aléatoire de type `size_t` compris entre `a` inclus et `b` exclu. Écrivez un code de test dans « `test_random.c` ».

(d) La fonction `int random_int(int a, int b);` qui retourne un nombre aléatoire de type `int` compris entre `a` inclus et `b` exclu. Écrivez un code de test dans « `test_random.c` ».

(e) La fonction `char random_char(char a, char b);` qui retourne un nombre aléatoire de type `unsigned char` compris entre `a` inclus et `b` exclu. Écrivez un code de test dans « `test_random.c` ».

(f) La fonction `void random_init_string(char * c, size_t n);` qui remplit la chaîne de caractère `c` de lettre majuscule aléatoire,`n` est la taille du tableau pointé par le pointeur `c`. Écrivez un code de test dans « `test_random.c` ».

4 Test de votre vecteur dynamique

Maintenant, vous allez créer un programme pour évaluer les performances de votre vecteur dynamique.

Ce programme prendra en paramètre, le type de test à exécuter et le nombre de test à exécuter, avec la structure suivante :

```
./bench_vector test_type init_size n
```

avec `test_type` le type du teste à exécuter, `init_size` la taille initial du tableau dynamique et `n` le nombre d'exécution du test. Pour cela vous devrez utiliser les arguments `argc` et `argv` de la fonction `int main(int argv, char *argv[])`

4.1. Complétez dans le fichier « `bench_vector.c` » :

(a) Écrivez dans votre fonction `main`, le code pour récupérer les arguments passé au programme et convertissez les arguments `init_size` et `n` en valeur numérique de type `size_t`. (pensez à utilisé la fonction `int sscanf(const char *str, const char * format, ...);` documentation ici.

(b) Après avoir récupérez les paramètres, écrivez dans votre fonction `main`, le code qui alloue un `vector` de taille `init_size` et qui l'initialise aléatoirement.

(c) Écrivez la fonction qui `void insert_erase_random(size_t init_size, size_t n);` qui alloue un `vector` de taille `init_size` et répète `n`-fois l'ajout et suppression d'un élément à des positions aléatoire.

Puis ajoutez dans le `main`, le code pour appeler cette fonction quand le `test_type` est égale à `"insert_erase_random"`.

(d) Écrivez la fonction qui `void insert_erase_head(size_t init_size, size_t n);` qui alloue un `vector` de taille `init_size` et répète `n`-fois l'ajout et la suppression d'un élément en tête.

Puis ajoutez dans le `main`, le code pour appeler cette fonction quand le `test_type` est égale à `"insert_erase_head"`.

(e) Écrivez la fonction qui `void insert_erase_tail(size_t init_size, size_t n);` qui alloue un `vector` de taille `init_size` et répète `n`-fois l'ajout et la suppression d'un élément en queue.

Puis ajoutez dans le `main`, le code pour appeler cette fonction quand le `test_type` est égale à `"insert_erase_tail"`.

(f) Écrivez la fonction qui `void read_write_random(size_t init_size, size_t n);` qui alloue un `vector` de taille `init_size` et répète `n`-fois la lecture et réécriture de la valeur lu incrémenté de 1 à des positions aléatoire.

Puis ajoutez dans le `main`, le code pour appeler cette fonction quand le `test_type` est égale à `"read_write_random"`.

(g) Écrivez la fonction qui `void read_write_sequential(size_t init_size, size_t n);` qui alloue un `vector` de taille `init_size` et répète `n`-fois la lecture et réécriture de la valeur lu incrémenté de 1 avec un parcours de la tête vers la queue.

Puis ajoutez dans le `main`, le code pour appeler cette fonction quand le `test_type` est égale à `"read_write_sequential"`.

(h) Écrivez la fonction qui `void bubble_sort(size_t init_size, size_t n);` qui alloue un `vector` de taille `init_size` et répète `n`-fois l'écriture de tous les éléments du vecteur avec des valeur aléatoire puis tri du vecteur avec le tri à bulles (documentation ici)

Puis ajoutez dans le `main`, le code pour appeler cette fonction quand le `test_type` est égale à `"bubble_sort"`.

4.2. Test des performance de votre structure pour les différents cas de test et différente valeur de `init_size` et `n`.

(a) Utilisez la commande Linux `time` (documentation ici) pour mesurer le temps d'exécution de votre `bench_vector`.

(b) Utilisez la commande Linux `valgrind` (documentation ici) pour mesurer le nombre d'allocation et la quantité total de mémoire alloué lors de l'exécution de votre `bench_vector`.