ПНИПУ

Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №13.

Стандартные обобщённые алгоритмы библиотеки STL.

Выполнил студент группы РИС-23-3Б

Буковский Денис Владимирович

Проверила доцент кафедры ИТАС О.А. Полякова

2024

1. Постановка задачи

Задача 1.

1. Создать последовательный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан B варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы тер1асе\_1#(), гер1асе\_сору(), replace copy if(), fill()).

4. Удалить элементы B соответствии с заданием (использовать алгоритмы remove (), removeiif 0, removeicopyiif () ‚ remove\_copy())

5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort ()).

6. Найти в контейнере заданный элемент (использовать алгоритмы £ind (), find if(), count(), count if()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for each()). 8. — Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

**Задача 2.**

1. Создать адаптер контейнера.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы replасе\_if(), герlасе\_сору(), replace copy if(), fill()).

4. Удалить элементы B соответствии с заданием (использовать алгоритмы remove (), remove\_if(), removeicopy\_if () ‚ remove\_copy())

5. Найти B контейнере элемент с заданным ключевым полем (использовать алгоритмы £ind (), find if(), count(), count\_if ()).

6. Выполнить задание варианта для полученного контейнера (использовать алгоритм for\_each()).

7. — Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 3.

1. Создать ассоциативный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы тер1асе\_1#(), гер1асе\_сору(), replace copy if(), fill()).

4. Удалить элементы B соответствии с заданием (использовать алгоритмы remove (), remove\_if 0, removeicopy\_if () ‚ remove\_copy())

5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort () ).

6. Найти B контейнере элемент с заданным ключевым полем (использовать алгоритмы £ind (), find if(), count(), count 1Е ()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for each()).

8. — Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

**Задача 1**

1. Контейнер – список

2. Тип элементов Pair (см. лабораторную работу №3).

**Задача 2**

Адаптер контейнера — очередь с приоритетами.

**Задача 3**

Ассоциативный контейнер – словарь.

**Задание 3** Найти среднее арифметическое и добавить его в конец контейнера.

**Задание 4** Найти элементы ключами из заданного диапазона и удалить их из контейнера.

**Задание 5** К каждому элементу добавить сумму минимального и максимального элементов контейнера.

1. Анализ задачи

Выполним каждую задачу в отдельной функции в tasks.h. Протестируем в функции main().

1. Код

Task1.h

#pragma once

#include <list>

#include <utility>

#include <iostream>

#include <algorithm>

using namespace std;

int avgfirst(list<pair<int, double>> data) {

list<pair<int, double>>::iterator it = data.begin();

int sum = 0, i = 0;

while (it != data.end()) {

sum += (\*it).first;

i++;

it++;

}

return sum / i;

}

double avgsecond(list<pair<int, double>> data) {

list<pair<int, double>>::iterator it = data.begin();

double sum = 0;

int i = 0;

while (it != data.end()) {

sum += (\*it).second;

i++;

it++;

}

return sum / i;

}

int addfirst(list<pair<int, double>> data) {

list<pair<int, double>>::iterator it = data.begin();

int min = (\*it).first, max = (\*it).first;

it++;

while (it != data.end()) {

if ((\*it).first < min) min = (\*it).first;

if ((\*it).first > max) max = (\*it).first;

it++;

}

return min + max;

}

double addsecond(list<pair<int, double>> data) {

list<pair<int, double>>::iterator it = data.begin();

double min = (\*it).second, max = (\*it).second;

it++;

while (it != data.end()) {

if ((\*it).second < min) min = (\*it).second;

if ((\*it).second > max) max = (\*it).second;

it++;

}

return min + max;

}

void print(list<pair<int, double>> data) {

list<pair<int, double>>::iterator it = data.begin();

cout << endl;

while (it != data.end()) {

cout << "(" << (\*it).first << ", " << (\*it).second << ")" << endl;

it++;

}

cout << endl;

}

void task1() {

list<pair<int, double>> adap;

pair<int, double> npair;

int n;

cout << "Enter how much elements should be in container: ";

cin >> n;

cout << "Enter elements. Use space as separator in pair." << endl;

for (int i = 1; i <= n; i++) {

cout << "#" << i << ": ";

cin >> npair.first >> npair.second;

adap.push\_back(npair);

}

print(adap); // after filling up

npair.first = avgfirst(adap);

npair.second = avgsecond(adap);

adap.push\_back(npair);

print(adap); // after adding avg element

list<pair<int, double>>::iterator it = adap.begin(), oldit;

int b1, b2;

cout << "Enter index borders (2 values): ";

cin >> b1 >> b2;

it = adap.begin();

for (int i = 0; i < n; i++) {

if (i >= b1 && i <= b2) {

oldit = it;

it++;

i++;

adap.remove(\*oldit);

}

it++;

}

print(adap); //after deleting

adap.sort();

print(adap); // after sorting

cout << "Enter pair to look for: ";

cin >> npair.first >> npair.second;

it = find(adap.begin(), adap.end(), npair);

if (it != adap.end()) cout << endl << "found!" << endl;

std::for\_each(adap.begin(), adap.end(), [adap](pair<int, double>& iter) {iter.first += addfirst(adap); iter.second += addsecond(adap); });

print(adap); // after searching & adding min+max to each element

}

Task2.h

#pragma once

#include <queue>

#include <utility>

#include <iostream>

using namespace std;

void print(priority\_queue<pair<int, double>> data) {

while (!data.empty()) {

cout << endl << "(" << data.top().first << ", " << data.top().second << ")" << endl;

data.pop();

}

}

int addfirst(priority\_queue<pair<int, double>> data) {

int min = data.top().first, max=data.top().first;

data.pop();

while (!data.empty()) {

if (data.top().first < min) min = data.top().first;

if (data.top().first > max) max = data.top().first;

data.pop();

}

return min + max;

}

double addsecond(priority\_queue<pair<int, double>> data) {

int min = data.top().second, max = data.top().second;

data.pop();

while (!data.empty()) {

if (data.top().second < min) min = data.top().second;

if (data.top().second > max) max = data.top().second;

data.pop();

}

return min + max;

}

int avgfirst(priority\_queue<pair<int, double>> data) {

priority\_queue<pair<int, double>> nque = data;

int sum = 0, i = 0;

while (!nque.empty()) { sum += nque.top().first; nque.pop(); i++; }

return sum / i;

}

double avgsecond(priority\_queue<pair<int, double>> data) {

priority\_queue<pair<int, double>> nque = data;

int sum = 0, i = 0;

while (!nque.empty()) { sum += nque.top().second; nque.pop(); i++; }

return sum / i;

}

void task2() {

priority\_queue<pair<int, double>> que;

pair<int, double> npair;

int n;

cout << "Enter how much elements should be in container: ";

cin >> n;

cout << "Enter elements. Use space as separator in pair." << endl;

for (int i = 1; i <= n; i++) {

cout << "#" << i << ": ";

cin >> npair.first >> npair.second;

que.push(npair);

}

print(que); // after creating

npair.first = avgfirst(que);

npair.second = avgsecond(que);

que.push(npair);

print(que); // after adding min+max

int b1, b2, i = 0;

cout << "Enter index borders (2 values): ";

cin >> b1 >> b2;

priority\_queue<pair<int, double>> nque{};

while (!que.empty()) {

if (!(i >= b1 && i <= b2)) {

nque.push(que.top());

}

que.pop();

i++;

}

que = nque;

print(que); // after deleting

nque = que;

cout << "Enter pair to look for: ";

cin >> npair.first >> npair.second;

while (!nque.empty()) {

if (nque.top() == npair) {

cout << endl << "found!" << endl;

break;

}

nque.pop();

}

nque = {};

int addf = addfirst(que);

double adds = addsecond(que);

while (!que.empty()) {

npair.first = que.top().first + addf;

npair.second = que.top().second + adds;

nque.push(npair);

que.pop();

}

que = nque;

print(que);

}

Task3.h

#pragma once

#include <iostream>

#include <utility>

#include <algorithm>

#include <map>

void print(map<int, pair<int, double>> data){

map<int, pair<int, double>>::iterator it = data.begin();

cout << endl;

while (it != data.end()) {

cout << "[" << (\*it).first << "] : (" << (\*it).second.first << ", " << (\*it).second.second << endl;

}

cout << endl;

}

int avgfirst(map<int, pair<int, double>> data) {

map<int, pair<int, double>>::iterator it = data.begin();

int sum = 0, i = 0;

while (it != data.end()) {

sum += (\*it).second.first;

i++;

it++;

}

return sum / i;

}

double avgsecond(map<int, pair<int, double>> data) {

map<int, pair<int, double>>::iterator it = data.begin();

double sum = 0;

int i = 0;

while (it != data.end()) {

sum += (\*it).second.second;

i++;

it++;

}

return sum / i;

}

int addfirst(map<int, pair<int, double>> data) {

map<int, pair<int, double>>::iterator it = data.begin();

int min = (\*it).second.first, max = (\*it).second.first;

it++;

while (it != data.end()) {

if ((\*it).second.first < min) min = (\*it).second.first;

if ((\*it).second.first > max) max = (\*it).second.first;

it++;

}

return min + max;

}

double addsecond(map<int, pair<int, double>> data) {

map<int, pair<int, double>>::iterator it = data.begin();

double min = (\*it).second.second, max = (\*it).second.second;

it++;

while (it != data.end()) {

if ((\*it).second.second < min) min = (\*it).second.second;

if ((\*it).second.second > max) max = (\*it).second.second;

it++;

}

return min + max;

}

void task3() {

map<int, pair<int, double>> dict;

pair<int, double> npair;

int n;

cout << "Enter how much elements should be in container: ";

cin >> n;

cout << "Enter elements. Use space as separator in pair." << endl;

for (int i = 0; i < n; i++) {

cout << "#" << i << ": ";

cin >> npair.first >> npair.second;

dict.insert(pair<int, pair<int, double>>(i, npair));

}

print(dict);

npair.first = avgfirst(dict);

npair.second = avgsecond(dict);

dict.insert(pair<int, pair<int, double>>(dict.size(), npair));

print(dict);

int b1, b2;

cout << "Enter index borders (2 values): ";

cin >> b1 >> b2;

//remove\_if(dict.begin(), dict.end(), [b1, b2](pair<int, pair<int, double>> ind) {return ((ind.second.first >= b1) && (ind.second.second <= b2)); });

print(dict);

map<int, pair<int, double>>::iterator it;

cout << "Enter pair to look for: ";

cin >> npair.first >> npair.second;

it = find\_if(dict.begin(), dict.end(), [npair](pair<int, pair<int, double>> p) {return (p.second == npair); });

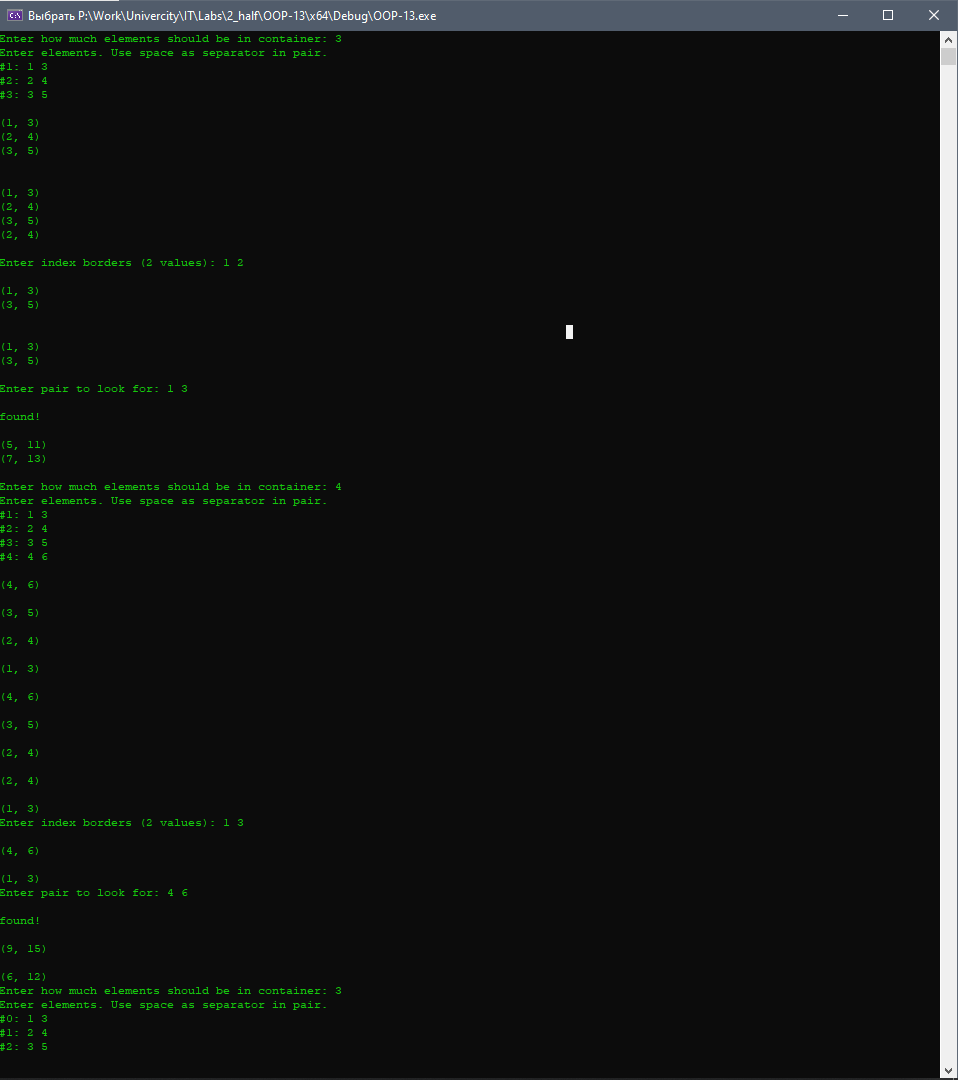
if (it != dict.end()) cout << endl << "found!" << endl;

for\_each(dict.begin(), dict.end(), [dict](pair<int, pair<int, double>> it) { it.second.first += addfirst(dict); it.second.second += addsecond(dict); });

print(dict);

}

1. Вывод программы



1. Ответы на контрольные вопросы

№11.

1. Набор контейнерных классов и набор обобщённых алгоритмов

2. Вектор, список, стек, очередь, двунаправленная очередь, очередь

с приоритетами, словарь, словарь с дубликатами, множество,

множество с дубликатами

3. Подключить соответствующий заголовочный файл

4. Итератор - класс, используемый для поиска элементов в

контейнере

5. Разыменовывание, присваивание, сравнение, перемещение по

всем элементам контейнера

6. for (i = first; i != last; ++i)

7. входные, выходные, прямые, двунаправленные, произвольного

доступа

8. == / !=; ==; clear; insert; erase; size\_type size() const; size\_type

max\_size() const; bool empty() const; iterator begin(), iterator end();

reverse\_iterator begin(); reverse\_iterator end()

9. Вставка/Удаление значения с конца, иначе придётся сдвигать все

значения путём копирования

10. Вставка и удаление произвольного поля

11. Вставка и удаление первого и последнего элемента

12. pop\_back(); push\_back(); insert; erase; [] / at; swap; clear()

13. pop\_back(); push\_back(T&key); pop\_front(); push\_front(T&key);

insert; erase; swap; clear(); splice

14. pop\_back(); push\_back(T&key); pop\_front(); push\_front(T&key);

insert; erase; [] / at

15. Записать все элементы после пятого, удалить с конца все

элемент кроме первого, добавить все записанные элементы.

Удаление из произвольного места возможно, но не эффективно.

16. v.pop\_back()

17. Записать первый элемент, удалить первые пять элементов с

начала, добавить записанный элемент в начало

18. l.pop\_back()

19. Установить итератор на позицию 2, удалить четыре элемента

20. Установить итератор на последнюю позицию. Удалить элемент

21. for (i = first; i != last; i++) { cout << cont[i]; }

22. Контейнеры, реализованные на основе других классов

23. Первый стек инициализирован на базе двусторонней очереди, а

второй на базе списка

24. push(); pop(); top(); empty(); size()

25.push(); pop(); front(); back(); empty(); size()

26. В очереди с приоритетами для извлечения выбирается

максимальные элемент из хранимых в контейнере

27. Записать все элементы до заданного номера в другой стек,

удалить элемент, записать элементы из второго стека в первый

28. Записать все элементы до заданного номера спереди или сзади в

другую очередь, удалить элемент, записать элементы обратно в

первую очередь

29. for (i = first; i != last; i++) { s.top(); s.pop(); }

30. for (i = first; i != last; i++) { q.front(); q.pop\_front(); }

№12.

1. Ассоциативные контейнеры содержат пары значений: ключ и данные

2. Словарь, словарь с дубликатами, множество, множество с дубликатами

3. С помощью итераторов

4. bool empty(); size\_type size() const; size\_type max\_size() const; insert();

erase(); clear(); swap(); key\_comp(); value\_comp(); find(); count();

lower\_bound(); upper\_bound()

5. explicit map(const Comp& c = Comp(), const Allocator& a = Allocator());

6. Упорядоченные по ключу

7. []; =; ==; <=; >=; !=; <; >;

8. m.insert(make\_pair(a, b))

9. m[i] = pair;

10. for (i = first; i != last; i++) { cout << m[i] }

11. for (i = first; i != last; i++) { cout << m[i] }

12. multimap допускает хранение элементов с одинаковыми ключами

13. В множествах значения не играют роли, отслеживаются только ключи

14. В множествах, в отличие от словарей, значения не играют роли,

отслеживаются только ключи

15. set<T> set1;

16. Упорядоченные по ключу

17. insert(); erase(); count();

18. s.insert(a);

19. for (i = set.begin(); i != set.end(); i++) { cout << \*i; }

20. multiset допускает хранение элементов с одинаковыми ключами