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

ЛАБОРАТОРНАЯ РАБОТА

СТАНДАРТНЫЕ ОБОБЩЕННЫЕ АЛГОРИТМЫ БИБЛИОТЕКИ STL

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**Постановка задачи**

**Постановка задачи**

Задача 1.

1. Создать последовательный 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

replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

1. Удалить элементы в соответствии с заданием (использовать

алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере заданный элемент (использовать алгоритмы

find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера

(использовать алгоритм for\_each()) .

1. 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

replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

1. Удалить элементы в соответствии с заданием (использовать

алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере элемент с заданным ключевым полем

(использовать алгоритмы find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера

(использовать алгоритм for\_each()) .

1. 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

replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

1. Удалить элементы в соответствии с заданием (использовать

алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере элемент с заданным ключевым полем

(использовать алгоритмы find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера

(использовать алгоритм for\_each()) .

1. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Для варианта 15:

Задача 1:

1. Контейнер – список
2. Тип элементов Pair (из лаб. 3)

Задача 2:

Адаптер контейнера – очередь с приоритетами.

Задача 3:

Ассоциативный контейнер – словарь

Для задач:

Найти среднее арифметическое и добавить в конец контейнера

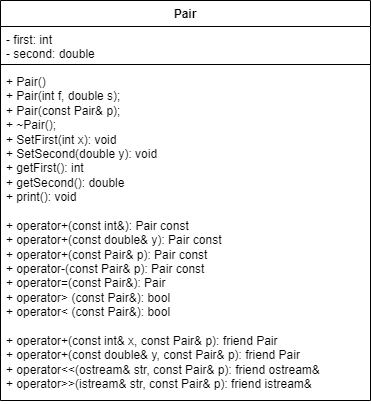
Найти элементы ключами из заданного диапазона и удалить их из контейнера

К каждому элементу добавить сумму минимального и максимального элементов контейнера.

**Анализ задачи**

1. Для использования стандартных контейнеров подключаются соответствующие библиотеки: задача 1: «список» - библиотека <list>, задача 2: «очередь с приоритетами» - <queue>, задача 3: «словарь» - <map>
2. Для использования лямбда функций, подключаются библиотеки <algorithm> и <functional>
3. Для использования метода sort() в первой задаче необходимо перегрузить операторы сравнения для пользовательского класса (в других задачах применение методов сортировки невозможно с использованием только стандартных библиотек из-за особенностей организации контейнеров)
4. Для заполнения контейнеров используется ДСЧ
5. Для удобства организации доступ ко всем программам осуществляется через стартовое меню, вызывающее необходимую по заданию программу (Task1.cpp – для задачи 1, Task2.cpp – задачи 2, Task3.cpp – задачи 3)

**UML диаграмма (общая для задач)**



**Код**

Pair.h

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int f, double s);

Pair(const Pair& p);

~Pair();

void SetFirst(int x);

void SetSecond(double y);

int getFirst();

double getSecond();

void print();

//перегруженные операции

Pair operator+(const int& x) const;

Pair operator+(const double& y) const;

Pair operator+(const Pair& p) const;

Pair operator-(const Pair& p) const;

Pair operator=(const Pair&);

bool operator> (const Pair&);

bool operator< (const Pair&);

friend Pair operator+(const int& x, const Pair& p);

friend Pair operator+(const double& y, const Pair& p);

friend ostream& operator<<(ostream& str, const Pair& p);

friend istream& operator>>(istream& str, Pair& p);

};

Pair.cpp

#pragma once

#include "Pair.h"

//перегрузка операции присваивания

Pair::Pair() {

first = 0;

second = 0;

}

Pair::Pair(int f = 0, double s = 0) {

this->first = f;

this->second = s;

}

Pair::Pair(const Pair& p) {

this->first = p.first;

this->second = p.second;

}

Pair::~Pair() {

}

int Pair::getFirst() { return this->first; }

void Pair::SetFirst(int x) { this->first = x; }

double Pair::getSecond() { return this->second; }

void Pair::SetSecond(double y) { this->second = y; }

void Pair::print() {

cout << this->first << " : " << this->second << endl;

}

//перегруженные операции

Pair Pair::operator+(const int& x) const {

return Pair(this->first + x, this->second);

}

Pair Pair::operator+(const double& y) const {

return Pair(this->first, this->second + y);

}

Pair Pair::operator+(const Pair& p) const {

return Pair(this->first + p.first, this->second + p.second);

}

Pair Pair::operator-(const Pair& p) const {

return Pair(this->first - p.first, this->second - p.second);

}

Pair Pair::operator=(const Pair& p) {

this->first = p.first;

this->second = p.second;

return \*this;

}

bool Pair::operator> (const Pair& p) {

if (this->first > p.first && this->second > p.second ||

this->first + this->second > p.first + p.second ||

this->first > p.first + p.second ||

this->second > p.first + p.second) {

return true;

}

return false;

}

bool Pair::operator< (const Pair& p )

{

if (this->first < p.first && this->second < p.second ||

p.first + p.second > this->first + this->second ||

p.first > this->first + this->second ||

p.second > this->first + this->second) {

return true;

}

return false;

}

//Дружественные перегруженные операции

Pair operator+(const int& x, const Pair& p) {

return p + x;

}

Pair operator+(const double& y, const Pair& p) {

return p + y;

}

ostream& operator<<(ostream& str, const Pair& p) {

str << " " << p.first << " : " << p.second;

return str;

}

istream& operator>>(istream& str, Pair& p) {

cout << "Enter int value: ";

str >> p.first;

cout << "Enter double value: ";

str >> p.second;

return str;

}

Tasks.h

#pragma once

int funk\_task1();

int funk\_task2();

int funk\_task3();

Task1.cpp

#include "Pair.h"

#include "Tasks.h"

#include <iostream>

#include <functional>

#include <algorithm>

#include <random>

#include <list>

using namespace std;

Pair LrandomPair() {

int a = rand() % 1000;

double b = rand() % 100000 \* 0.01;

Pair newPair;

newPair.SetFirst(a);

newPair.SetSecond(b);

return newPair;

}

ostream& operator<<(ostream& stream, list<Pair>& l) {

for\_each(l.begin(), l.end(), [](Pair& p) {cout << p; });

return stream;

}

int funk\_task1() {

int sizeVal;

do {

cout << "Enter list size: ";

cin >> sizeVal;

} while (sizeVal < 1);

list<Pair> list1(sizeVal);

generate(list1.begin(), list1.end(), LrandomPair);

cout << "List generated:\n" << list1 << endl;

int maxIntVal = list1.begin()->getFirst(), minIntVal = list1.begin()->getFirst();

double maxDoubleVal = list1.begin()->getSecond(), minDoubleVal = list1.begin()->getSecond();

Pair midArifm, addPair;

list1.sort();

cout << "List sorted:\n" << list1 << endl;

list1.sort([](Pair& a, Pair& b) {return a > b; });

cout << "Reverse sort:\n" << list1 << endl;

for\_each(list1.begin(), list1.end(),

[&midArifm, &maxIntVal, &minIntVal, &maxDoubleVal, &minDoubleVal](Pair& p) {

midArifm = midArifm + p;

if (p.getFirst() > maxIntVal) maxIntVal = p.getFirst();

if (p.getFirst() < minIntVal) minIntVal = p.getFirst();

if (p.getSecond() > maxIntVal) maxDoubleVal = p.getSecond();

if (p.getSecond() < minIntVal) minDoubleVal = p.getSecond();

});

midArifm.SetFirst(midArifm.getFirst() / sizeVal);

midArifm.SetSecond(midArifm.getSecond() / sizeVal);

addPair.SetFirst(minIntVal + maxIntVal);

addPair.SetSecond(minDoubleVal + maxDoubleVal);

list1.push\_back(midArifm);

cout << "List's arithmetical mean:\n" << midArifm << endl;

cout << "List after adding arithmetical mean:\n" << list1 << endl;

double lowLim, highLim;

cout << "Enter value range for deleting:\n";

cout << "Lower limit: ";

cin >> lowLim;

cout << "Higher limit: ";

cin >> highLim;

if (lowLim > highLim) swap(lowLim, highLim);

list<Pair>::iterator iter = remove\_if(list1.begin(), list1.end(),

[lowLim, highLim](Pair& p) {

return p.getFirst() + p.getSecond() > lowLim && p.getFirst() + p.getSecond() < highLim;

});

list1.erase(iter, list1.end());

if (list1.size() == 0) {

cout << "List is empty" << endl;

return 0;

}

cout << "List after clearing:\n" << list1 << endl;

cout << "Sum of min and max elements: " << addPair << endl;

for\_each(list1.begin(), list1.end(), [addPair](Pair& p) {p = p + addPair; });

cout << "List after adding sum of elements:\n" << list1 << endl;

cout << "Enter search key (int): ";

cin >> sizeVal;

auto tempPair = find\_if(list1.begin(), list1.end(),

[sizeVal](Pair& p) {

return sizeVal == p.getFirst();

});

if (tempPair != list1.end()) {

cout << "Found pair: " << \*tempPair << endl;

}

else {

cout << "Pair not found" << endl;

}

return 0;

}

Task2.cpp

#include "Pair.h"

#include "Tasks.h"

#include <iostream>

#include <functional>

#include <algorithm>

#include <random>

#include <queue>

using namespace std;

class GreaterThanPair {

public:

bool operator()(Pair a, Pair b) {

return a > b;

}

};

Pair QrandomPair() {

int a = rand() % 1000;

double b = rand() % 100000 \* 0.01;

Pair newPair;

newPair.SetFirst(a);

newPair.SetSecond(b);

return newPair;

}

ostream& operator<<(ostream& stream, priority\_queue<Pair, vector<Pair>, GreaterThanPair> q) {

while (!q.empty()) {

cout << q.top() << endl;

q.pop();

}

return stream;

}

void generateQueue(priority\_queue<Pair, vector<Pair>, GreaterThanPair>& q, int sizeVal) {

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

q.push(QrandomPair());

}

}

void minmaxSearch(priority\_queue<Pair, vector<Pair>, GreaterThanPair> q, Pair& midArifm, Pair addPair) {

Pair p = q.top();

int maxIntVal = p.getFirst(), minIntVal = p.getFirst();

double maxDoubleVal = p.getSecond(), minDoubleVal = p.getSecond();

while (!q.empty()) {

p = q.top();

q.pop();

midArifm = midArifm + p;

if (p.getFirst() > maxIntVal) maxIntVal = p.getFirst();

if (p.getFirst() < minIntVal) minIntVal = p.getFirst();

if (p.getSecond() > maxIntVal) maxDoubleVal = p.getSecond();

if (p.getSecond() < minIntVal) minDoubleVal = p.getSecond();

}

addPair.SetFirst(minIntVal + maxIntVal);

addPair.SetSecond(minDoubleVal + maxDoubleVal);

}

void rangeRemove(priority\_queue<Pair, vector<Pair>, GreaterThanPair>& q, int lowLim, int highLim) {

priority\_queue<Pair, vector<Pair>, GreaterThanPair> q1;

while (!q.empty()) {

Pair p = q.top();

if (!(p.getFirst() + p.getSecond() > lowLim && p.getFirst() + p.getSecond() < highLim)) {

q1.push(p);

}

q.pop();

}

q = q1;

}

void applyToAll(priority\_queue<Pair, vector<Pair>, GreaterThanPair>& q, Pair addPair) {

priority\_queue<Pair, vector<Pair>, GreaterThanPair> q1;

while (!q.empty()) {

Pair p = q.top();

q1.push(p + addPair);

q.pop();

}

q = q1;

}

bool searchPair(priority\_queue<Pair, vector<Pair>, GreaterThanPair> q, int key, Pair& tempPair) {

while (!q.empty())

{

Pair p = q.top();

q.pop();

if (p.getFirst() == key) {

tempPair = p;

return true;

}

}

return false;

}

int funk\_task2() {

int sizeVal;

//Creating

do {

cout << "Enter queue size: ";

cin >> sizeVal;

} while (sizeVal < 1);

priority\_queue<Pair, vector<Pair>, GreaterThanPair> queue1;

generateQueue(queue1, sizeVal);

cout << "Queue generated:\n" << queue1 << endl;

//Sorting

//Queue cannot be sorted

//Min max values search + sum of all

Pair midArifm, addPair;

minmaxSearch(queue1, midArifm, addPair);

midArifm.SetFirst(midArifm.getFirst() / sizeVal);

midArifm.SetSecond(midArifm.getSecond() / sizeVal);

queue1.push(midArifm);

cout << "Queue's arithmetical mean:\n" << midArifm << endl;

cout << "Queue after adding arithmetical mean:\n" << queue1 << endl;

double lowLim, highLim;

//Deleting

cout << "Enter value range for deleting:\n";

cout << "Lower limit: ";

cin >> lowLim;

cout << "Higher limit: ";

cin >> highLim;

if (lowLim > highLim) swap(lowLim, highLim);

rangeRemove(queue1, lowLim, highLim);

if (queue1.size() == 0) {

cout << "Queue is empty" << endl;

return 0;

}

cout << "Queue after clearing:\n" << queue1 << endl;

//MinMax applying

cout << "Sum of min and max elements: " << addPair << endl;

applyToAll(queue1, addPair);

cout << "Queue after adding sum of elements:\n" << queue1 << endl;

//Search

cout << "Enter search key (int): ";

cin >> sizeVal;

Pair tempPair;

if (searchPair(queue1, sizeVal, tempPair)) {

cout << "Found pair: " << tempPair << endl;

}

else {

cout << "Pair not found" << endl;

}

return 0;

}

Task3.cpp

#include "Pair.h"

#include "Tasks.h"

#include <iostream>

#include <functional>

#include <algorithm>

#include <random>

#include <map>

using namespace std;

Pair MrandomPair() {

int a = rand() % 1000;

double b = rand() % 100000 \* 0.01;

Pair newPair;

newPair.SetFirst(a);

newPair.SetSecond(b);

return newPair;

}

ostream& operator<<(ostream& stream, pair<int, Pair> p) {

stream << "Key: " << p.first << endl << "Value: " << p.second << endl;

return stream;

}

ostream& operator<<(ostream& stream, map<int, Pair> m) {

map<int, Pair>::const\_iterator iter = m.begin();

while (iter != m.end()) {

stream << \*iter++;

}

return stream;

}

void generateMap(map<int, Pair>& m, int sizeVal) {

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

Pair p = MrandomPair();

m[p.getFirst()] = p;

}

}

void rangeRemove(map<int, Pair>& m, int lowLim, int highLim) {

map<int, Pair>::iterator iter = m.begin();

int key;

while (iter != m.end()) {

key = iter->first;

if (!(key > lowLim && key < highLim)) {

++iter;

}

else {

iter = m.erase(iter);

}

}

}

int funk\_task3() {

int sizeVal;

//Creating

do {

cout << "Enter map size: ";

cin >> sizeVal;

} while (sizeVal < 1);

map<int, Pair> map1;

generateMap(map1, sizeVal);

cout << "Map generated:\n" << map1 << endl;

//Sorting

//Map cannot be sorted

//Min max values search + sum of all

int maxIntVal = map1.begin()->second.getFirst(), minIntVal = map1.begin()->second.getFirst();

double maxDoubleVal = map1.begin()->second.getSecond(), minDoubleVal = map1.begin()->second.getSecond();

Pair midArifm, addPair;

for\_each(map1.begin(), map1.end(),

[&midArifm, &maxIntVal, &minIntVal, &maxDoubleVal, &minDoubleVal](pair<int, Pair> p) {

midArifm = midArifm + p.second;

if (p.second.getFirst() > maxIntVal) maxIntVal = p.second.getFirst();

if (p.second.getFirst() < minIntVal) minIntVal = p.second.getFirst();

if (p.second.getSecond() > maxIntVal) maxDoubleVal = p.second.getSecond();

if (p.second.getSecond() < minIntVal) minDoubleVal = p.second.getSecond();

});

addPair.SetFirst(minIntVal + maxIntVal);

addPair.SetSecond(minDoubleVal + maxDoubleVal);

midArifm.SetFirst(midArifm.getFirst() / sizeVal);

midArifm.SetSecond(midArifm.getSecond() / sizeVal);

map1[midArifm.getFirst()] = midArifm;

cout << "Map's arithmetical mean:\n" << midArifm << endl;

cout << "Map after adding arithmetical mean:\n" << map1 << endl;

double lowLim, highLim;

//Deleting

cout << "Enter value range for deleting:\n";

cout << "Lower limit: ";

cin >> lowLim;

cout << "Higher limit: ";

cin >> highLim;

if (lowLim > highLim) swap(lowLim, highLim);

rangeRemove(map1, lowLim, highLim);

if (map1.size() == 0) {

cout << "Map is empty" << endl;

return 0;

}

cout << "Map after clearing:\n" << map1 << endl;

//MinMax applying

cout << "Sum of min and max elements: " << addPair << endl;

for\_each(map1.begin(), map1.end(), [&map1, addPair](pair<int, Pair> p) {map1[p.first] = p.second + addPair; });

cout << "Map after adding sum of elements:\n" << map1 << endl;

//Search

cout << "Enter search key (int): ";

cin >> sizeVal;

auto tempPair = find\_if(map1.begin(), map1.end(),

[sizeVal](pair<int, Pair> p) {

return sizeVal == p.first;

});

if (tempPair != map1.end()) {

cout << "Found pair: " << tempPair->second << endl;

}

else {

cout << "Pair not found" << endl;

}

return 0;

}

Main.cpp

#include "Tasks.h"

#include <iostream>

int main() {

int i;

std::cout << "Enter task number: " << std::endl;

std::cin >> i;

switch (i)

{

case 1:

funk\_task1();

break;

case 2:

funk\_task2();

break;

case 3:

funk\_task3();

break;

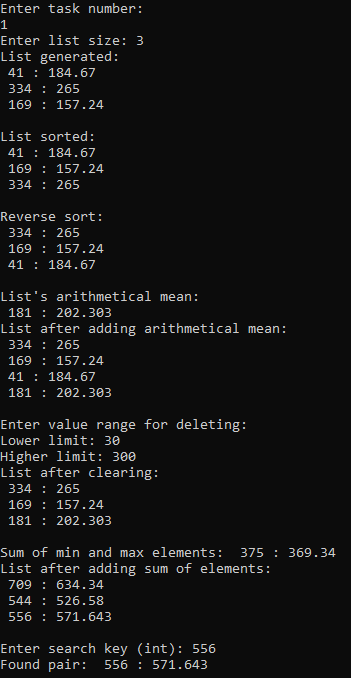
default:

break;

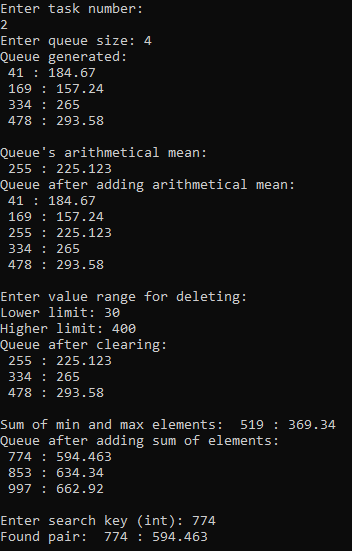
}

}

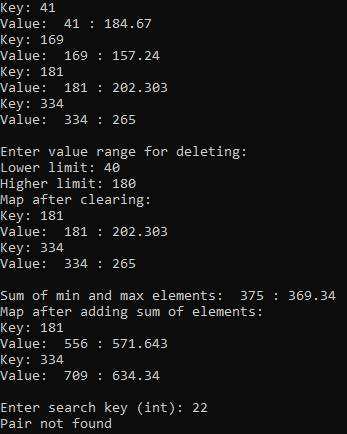
**Решение**



Задание 1



Задание 2



Задание 3

**Выводы**

В ходе выполнения лабораторной работы били изучены STL библиотеки C++ и методы работы с ними.

**Github**

<https://github.com/Hitikov/Lab_OOP_13>