Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «ООП. Стандартные обобщенные алгоритмы библиотеки STL»

Выполнил студент ИВТ-22-2б:

Казанцев Антон Васильевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2023

**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()).
4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(),remove\_if(), remove\_copy\_if(),remove\_copy()).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
6. Найти в контейнере заданный элемент (использовать алгоритмы find(), 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.

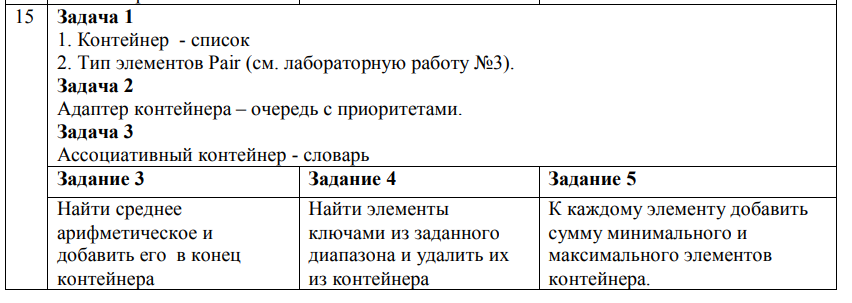
Задача 2.

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 replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).
4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(),remove\_if(), remove\_copy\_if(),remove\_copy()).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), 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 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()).
4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(),remove\_if(), remove\_copy\_if(),remove\_copy()).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), 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.

*Вариант 15:*



**Программное решение**

**Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair {

int first;

double second;

public:

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

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

Pair(int, double);

Pair();

Pair(const Pair&);

~Pair();

Pair& operator=(const Pair&);

Pair operator-(const Pair&);

Pair& operator+(int);

Pair& operator+(double);

Pair& operator++();

Pair operator++(int);

int GetFirst() const;

double GetSecond() const;

friend bool operator<(const Pair&, const Pair&);

Pair& operator + (const Pair& m\_pair);

void SetFirst(int m\_first);

void SetSecond(double m\_second);

};

**Pair.cpp**

#include "Pair.h"

Pair::Pair(int first, double second) {//конструктор с параметрами

this->first = first;

this->second = second;

}

istream& operator>>(istream& in, Pair& m\_pair) {//перегрузка оператора ввода

in >> m\_pair.first;

in >> m\_pair.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& m\_pair) {//перегрузка оператора вывода

return (out << m\_pair.first << " : " << m\_pair.second);

}

Pair::Pair() {//конструктор без параметров

first = 0;

second = 0;

}

Pair::~Pair() {//деструктор

}

Pair::Pair(const Pair& m\_pair) {//конструктор копирования

first = m\_pair.first;

second = m\_pair.second;

}

Pair& Pair::operator=(const Pair& m\_pair) {//перегрузка оператора присваивания

if (&m\_pair != this) {

first = m\_pair.first;

second = m\_pair.second;

}

return \*this;

}

Pair Pair::operator-(const Pair& m\_pair) {//перегрузка оператора вычитания

Pair result(first - m\_pair.first, second - m\_pair.second);

return result;

}

Pair& Pair::operator+(int first) {//перегрузка оператора сложения - целый аргумент

this->first += first;

return \*this;

}

Pair& Pair::operator+(double second) {//перегрузка оператора сложения - дробный аргумент

this->second += second;

return \*this;

}

Pair& Pair::operator++() {//перегрузка оператора ++ префикс

++first;

++second;

return \*this;

}

Pair Pair::operator ++(int) {//перегрузка оператора ++ постфикс

Pair temp = \*this;

this->first++;

this->second++;

return temp;//сначала нужно вывести без изменений

}

int Pair::GetFirst() const{

return first;

}

double Pair::GetSecond() const{

return second;

}

bool operator < (const Pair& pair1, const Pair& pair) {

return (pair1.GetFirst() + pair1.GetSecond() < pair.GetFirst() + pair.GetSecond());

}

Pair& Pair::operator + (const Pair& m\_pair) {

first += m\_pair.first;

second += m\_pair.second;

return \*this;

}

void Pair::SetFirst(int m\_first) {

first = m\_first;

}

void Pair::SetSecond(double m\_second) {

second = m\_second;

}

**task1.cpp**

#include "Pair.h"

#include <iostream>

#include <list>

#include <numeric>

#include <algorithm>

using namespace std;

void PushArithmeticMiddle(list<Pair>& m\_list) {

if (m\_list.size() > 0) {

Pair Middle = accumulate(m\_list.begin(), m\_list.end(), Pair(0, 0));

Middle.SetFirst(Middle.GetFirst() / m\_list.size());

Middle.SetSecond(Middle.GetSecond() / m\_list.size());

m\_list.push\_back(Middle);

}

else {

cout << "Список пуст!\n";

}

}

void EraseIfInRange(list<Pair>& m\_list, const double BeginValue, const double EndValue) {

auto i = remove\_if(m\_list.begin(), m\_list.end(), [BeginValue, EndValue](Pair p) {

return ((BeginValue <= p.GetFirst() && p.GetFirst() <= EndValue) || (BeginValue <= p.GetSecond() && p.GetSecond() <= EndValue));

});

m\_list.erase(i, m\_list.end());

}

void PlusMinAndMax(list<Pair>& m\_list) {

if (m\_list.size() > 0) {

Pair max = \*max\_element(m\_list.begin(), m\_list.end());

Pair min = \*min\_element(m\_list.begin(), m\_list.end());

for (Pair& c : m\_list) {

c + max + min;

}

}

}

void ShowList(const list<Pair>& m\_list) {

if (m\_list.size() > 0) {

for\_each(m\_list.begin(), m\_list.end(), [](Pair p) {

cout << p << '\n';

});

}

else {

cout << "Список пуст!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

list<Pair> a = { Pair(6, 4), Pair(7, 9), Pair(2, 3.5) };

cout << "Список а:\n";

ShowList(a);

cout << "Добавляем среднее арифмитическое списка а в конец контейнера\n";

PushArithmeticMiddle(a);

cout << "Список а:\n";

ShowList(a);

cout << "Удаляем из списка а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Список а:\n";

ShowList(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

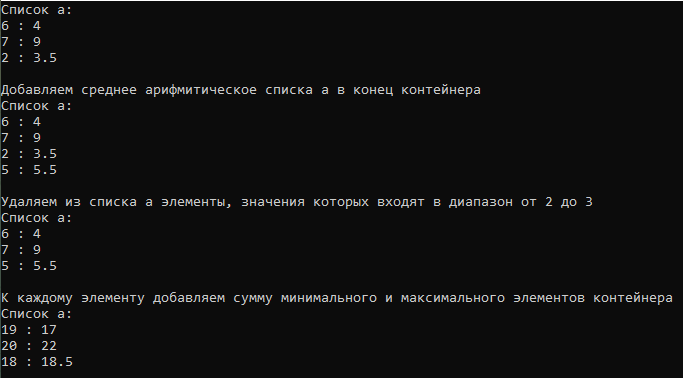
PlusMinAndMax(a);

cout << "Список а:\n";

ShowList(a);

return 0;

}



**Задача 2:**

**Программное решение**

#include "Pair.h"

#include <iostream>

#include <list>

#include <queue>

#include <numeric>

#include <algorithm>

using namespace std;

list<Pair> priority\_queueTolist(priority\_queue<Pair>& m\_priority\_queue) {

priority\_queue <Pair> tmp\_priority\_queue = m\_priority\_queue;

list <Pair> m\_list;

while (tmp\_priority\_queue.size() > 0) {

m\_list.push\_back(tmp\_priority\_queue.top());

tmp\_priority\_queue.pop();

}

return m\_list;

}

void PushArithmeticMiddle(priority\_queue<Pair>& m\_priority\_queue) {

if (m\_priority\_queue.size() > 0) {

list <Pair> m\_list = priority\_queueTolist(m\_priority\_queue);

Pair Middle = accumulate(m\_list.begin(), m\_list.end(), Pair(0, 0));

Middle.SetFirst(Middle.GetFirst() / m\_list.size());

Middle.SetSecond(Middle.GetSecond() / m\_list.size());

m\_priority\_queue.push(Middle);

}

else {

cout << "Очередь пуста!\n";

}

}

void EraseIfInRange(priority\_queue<Pair>& m\_priority\_queue, const double BeginValue, const double EndValue) {

if (m\_priority\_queue.size() > 0) {

list <Pair> m\_list = priority\_queueTolist(m\_priority\_queue);

auto i = remove\_if(m\_list.begin(), m\_list.end(), [BeginValue, EndValue](Pair p) {

return (BeginValue <= p.GetFirst() && p.GetFirst() <= EndValue) || (BeginValue <= p.GetSecond() && p.GetSecond() <= EndValue);

});

m\_priority\_queue = {};

if (i != m\_list.begin()) {

for\_each(m\_list.begin(), i--, [&m\_priority\_queue](Pair p) {

m\_priority\_queue.push(p);

});

}

}

}

void PlusMinAndMax(priority\_queue<Pair>& m\_priority\_queue) {

if (m\_priority\_queue.size() > 0) {

list <Pair> m\_list = priority\_queueTolist(m\_priority\_queue);

m\_priority\_queue = {};

Pair max = \*max\_element(m\_list.begin(), m\_list.end());

Pair min = \*min\_element(m\_list.begin(), m\_list.end());

for (Pair& c : m\_list) {

m\_priority\_queue.push(c + max + min);

}

}

}

void Show\_priority\_queue(priority\_queue<Pair>& m\_priority\_queue) {

if (m\_priority\_queue.size() > 0) {

list <Pair> m\_list = priority\_queueTolist(m\_priority\_queue);

for\_each(m\_list.begin(), m\_list.end(), [](Pair p) {

cout << p << '\n';

});

}

else {

cout << "Очередь пуста!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

priority\_queue <Pair> a;

for (Pair i : { Pair(6, 4), Pair(7, 9), Pair(2, 3.5) }) {

a.push(i);

}

cout << "Очередь с приоритетами а:\n";

Show\_priority\_queue(a);

cout << "Добавляем среднее арифмитическое очереди а в контейнер\n";

PushArithmeticMiddle(a);

cout << "Очередь а:\n";

Show\_priority\_queue(a);

cout << "Удаляем из очереди а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Очередь а:\n";

Show\_priority\_queue(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

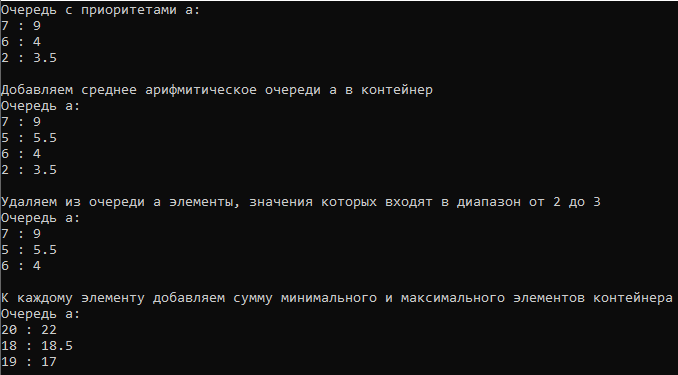
PlusMinAndMax(a);

cout << "Очередь а:\n";

Show\_priority\_queue(a);

return 0;

}



**Задача 3:**

**Программное решение**

#include <map>

#include <iostream>

#include <list>

#include <numeric>

#include <algorithm>

using namespace std;

void PushArithmeticMiddle(map <int, int>& m\_map) {

if (m\_map.size() > 0) {

int MaxKey = max\_element(m\_map.begin(), m\_map.end(), [](const auto& p1, const auto& p2) {

return p1.first < p2.first;

})->first + 1;

int MiddleSecond = accumulate(m\_map.begin(), m\_map.end(), 0, [](int value, const auto& p) {

return value + p.second;

}) / m\_map.size();

m\_map[MaxKey] = MiddleSecond;

m\_map.insert({ MaxKey, MiddleSecond });

}

else {

cout << "Словарь пуст!\n";

}

}

void EraseIfInRange(map <int, int>& m\_map, const int BeginValue, const int EndValue) {

for (auto p = m\_map.begin(); p != m\_map.end();) {

if (BeginValue <= (\*p).second && (\*p).second <= EndValue) {

p = m\_map.erase(p);

}

else {

p++;

}

};

}

void PlusMinAndMax(map <int, int>& m\_map) {

if (m\_map.size() > 0) {

int MaxValue = max\_element(m\_map.begin(), m\_map.end(), [](const auto& p1, const auto& p2) {

return p1.second < p2.second;

})->second;

int MinValue = min\_element(m\_map.begin(), m\_map.end(), [](const auto& p1, const auto& p2) {

return p1.second < p2.second;

})->second;

for (auto& p : m\_map) {

p.second += MaxValue + MinValue;

}

}

}

void ShowMap(const map <int, int>& m\_map) {

if (m\_map.size() > 0) {

for (auto& p : m\_map) {

cout << p.first << " : " << p.second << '\n';

}

}

else {

cout << "Словарь пуст!";

}

cout << '\n';

}

int main() {

system("chcp 1251 > NULL");

map <int, int> a = { {6, 4}, {7, 9}, {2, 3} };

cout << "Словарь а:\n";

ShowMap(a);

cout << "Добавляем среднее арифмитическое словаря а в конец контейнера\n";

PushArithmeticMiddle(a);

cout << "Словарь а:\n";

ShowMap(a);

cout << "Удаляем из словаря а элементы, значения которых входят в диапазон от 2 до 3\n";

EraseIfInRange(a, 2, 3);

cout << "Словарь а:\n";

ShowMap(a);

cout << "К каждому элементу добавляем сумму минимального и максимального элементов контейнера\n";

PlusMinAndMax(a);

cout << "Словарь а:\n";

ShowMap(a);

return 0;

}

