Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Стандартные обобщённые алгоритмы библиотеки STL.

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Токарев Павел Аркадьевич

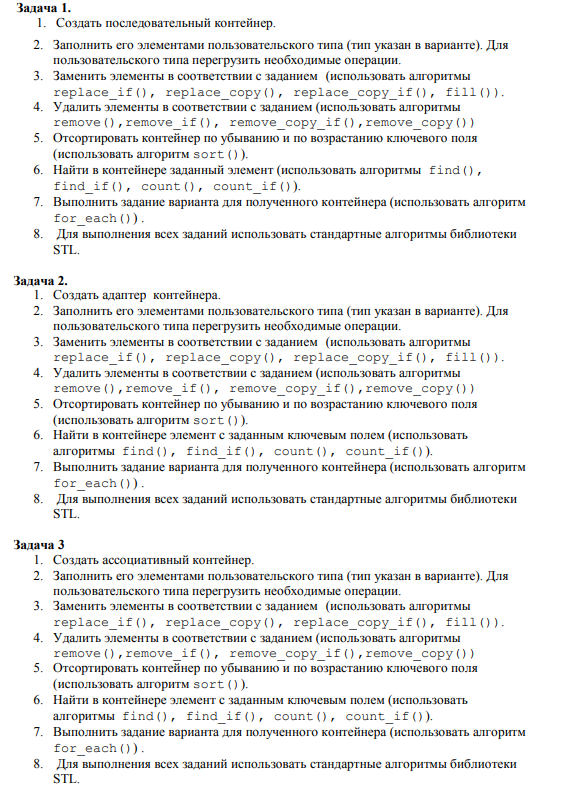
Проверил

Доцент кафедры ИТАС

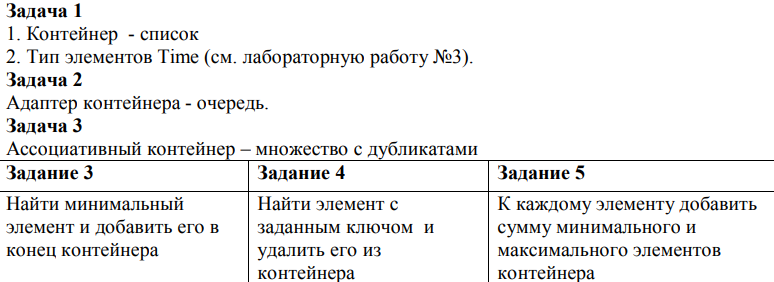
Полякова Ольга Андреевна

Г. Пермь-2023

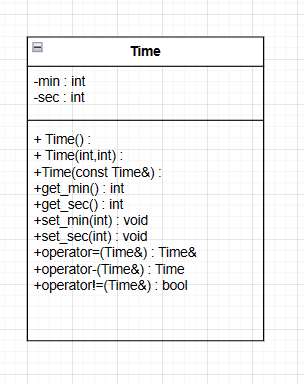
**Постановка задачи**

****

Вариант 2:



**Диаграмма UML**

****

**Код программы**

task1.h

#pragma once

#include<algorithm>

#include<list>

#include"Time.h"

#include<functional>

using namespace std;

bool fnd = false;

void make\_list(list<Time>& lst,int n) {

Time tmp;

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

cin >> tmp;

lst.push\_back(tmp);

}

}

bool eq(Time& a, Time& b) {

if (a == b) {

return true;

}

else {

return false;

}

}

void show(list<Time>& lst) {

for\_each(lst.begin(), lst.end(), [](const Time& t) {

cout << t << " ";

});

cout << endl;

}

void find\_time(list<Time> lst, Time& t) {

auto it = find\_if(lst.begin(), lst.end(), bind(eq, placeholders::\_1, t));

if (it == lst.end())

{

cout << "Элемент не найден" << endl;

return;

}

for (auto i = lst.begin(); i != lst.end(); ++i)

if (i == it)

cout << "Элемент найден - " << \*i<<endl;

}

void addmin(list<Time> &lst) {

Time tmp(100000,10000);

for\_each(lst.begin(), lst.end(), [&tmp](Time& t) {if (tmp > t) { tmp = t; }});

lst.push\_back(tmp);

}

void pop\_key(list<Time> &lst, Time key) {

auto j=remove\_if(lst.begin(), lst.end(), [&key](Time& t) {return t == key;});

lst.erase(j, lst.end());

}

void add\_min\_max(list<Time> &lst) {

Time min(100000, 10000),max(0,0);

for\_each(lst.begin(), lst.end(), [&min](Time& t) {if (min > t) { min = t; }});

for\_each(lst.begin(), lst.end(), [&max](Time& t) {if (max<t) { max = t; }});

list <Time> lsttmp;

for\_each(lst.begin(), lst.end(), [&lsttmp,&min,&max](Time& t) {lsttmp.push\_back(t+min+max);});

lst = lsttmp;

}

void task1() {

list<Time> new\_list;

cout << "Введите размер списка : ";

int n;

cin >>n;

make\_list(new\_list, n);

cout << "Добавим в конец минимальный элемент" << endl;

addmin(new\_list);

show(new\_list);

cout << "Проверим элемент на вхождение : " << endl;

Time a;

cin >> a;

find\_time(new\_list,a);

cout << "Удалим элемент с заданным ключём" << endl;

cout << "Ключ:"<<endl;

cin >> a;

pop\_key(new\_list, a);

show(new\_list);

cout << "Добавим к всем элементам списка min+max" << endl;

add\_min\_max(new\_list);

show(new\_list);

cout << "Отсортируем список" << endl;

new\_list.sort(mem\_fun\_ref(&Time::operator<));

show(new\_list);

}

task2.h

#pragma once

#include<algorithm>

#include<queue>

#include"Time.h"

#include<functional>

#include<vector>

#include<list>

using namespace std;

void make\_list(queue<Time>& lst, int n) {

Time tmp;

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

cin >> tmp;

lst.push(tmp);

}

}

bool q\_equal(Time& a, Time& b) {

if (a == b) {

return true;

}

else {

return false;

}

}

void qshow(queue<Time>& lst) {

vector<Time> Copy;

Time q;

size\_t lst\_sz = lst.size();

for (int i = 0;i < lst\_sz;i++) {

q = lst.front();

cout << q<<" ";

Copy.push\_back(q);

lst.pop();

}

int k = 0;

for (int t = 0;t < Copy.size();t++) {

lst.push(Copy[k]);

k++;

}

cout << endl;

}

void find\_time(queue<Time> lst, Time& t) {

vector<Time> Copy;

Time q;

size\_t lst\_sz = lst.size();

for (int i = 0;i < lst\_sz;i++) {

q = lst.front();

Copy.push\_back(q);

lst.pop();

}

auto it = find\_if(Copy.begin(), Copy.end(), bind(eq, placeholders::\_1, t));

if (it == Copy.end())

{

cout << "Элемент не найден" << endl;

return;

}

for (auto i = Copy.begin(); i != Copy.end(); ++i)

if (i == it)

cout << "Элемент найден - " << \*i << endl;

}

void addmin(queue<Time>& lst) {

vector<Time> Copy;

Time q;

size\_t lst\_sz = lst.size();

for (int i = 0;i < lst\_sz;i++) {

q = lst.front();

Copy.push\_back(q);

lst.pop();

}

Time tmp(100000, 10000);

for\_each(Copy.begin(), Copy.end(), [&tmp](Time& t) {if (tmp > t) { tmp = t; }});

for\_each(Copy.begin(), Copy.end(), [&tmp, &lst](Time& t) {lst.push(t);});

lst.push(tmp);

}

void pop\_key(queue<Time>& lst, Time key) {

vector<Time> temp;

size\_t q\_size=lst.size();

for (int i = 0; i <q\_size; i++)

{

temp.push\_back(lst.front());

lst.pop();

}

bool fl = false;

for\_each(temp.begin(), temp.end(), [&fl, &key, &lst](Time& p) {

if (fl == false && p == key) {

fl = true;

}

else { lst.push(p); }});

}

void add\_min\_max(queue<Time>& lst) {

vector<Time> Copy;

Time q;

size\_t lst\_sz = lst.size();

for (int i = 0;i < lst\_sz;i++) {

q = lst.front();

Copy.push\_back(q);

lst.pop();

}

Time min(100000, 10000);

Time max(0, 0);

for\_each(Copy.begin(), Copy.end(), [&min, &max](Time& t) {if (max < t) { max = t; }if (min > t) { min = t; }});

for\_each(Copy.begin(), Copy.end(), [&min, &max](Time& t) {t = t + min + max;});

for (auto i = Copy.begin(); i != Copy.end(); i++)

lst.push(\*i);

}

void task2() {

queue<Time> new\_list;

cout << "Введите размер очереди : ";

int n;

cin >> n;

make\_list(new\_list, n);

cout << "Добавим в конец минимальный элемент" << endl;

addmin(new\_list);

qshow(new\_list);

cout << "Проверим элемент на вхождение : " << endl;

Time a;

cin >> a;

find\_time(new\_list, a);

cout << "Удалим элемент с заданным ключём" << endl;

cout << "Ключ:" << endl;

cin >> a;

pop\_key(new\_list, a);

qshow(new\_list);

cout << "Добавим к всем элементам очереди min+max" << endl;

add\_min\_max(new\_list);

qshow(new\_list);

cout << "Отсортируем очередь" << endl;

qshow(new\_list);

}

Task3.h

#pragma once

#include<algorithm>

#include<queue>

#include"Time.h"

#include<functional>

#include<vector>

#include<list>

#include<set>

using namespace std;

void make\_list(multiset<int>& lst, int n) {

int tmp;

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

cout << i + 1 << " : ";

cin >> tmp;

lst.insert(tmp);

}

}

bool m\_equal(Time& a, Time& b) {

if (a == b) {

return true;

}

else {

return false;

}

}

void sshow(multiset<int>& lst) {

vector<int> Copy;

int q;

size\_t lst\_sz = lst.size();

for (auto& i : lst) {

cout << i << " ";

}

cout << endl;

}

void find\_time(multiset<int> lst, int& t) {

vector<int> Copy;

int q;

for (auto& i : lst) {

Copy.push\_back(i);

}

for\_each(Copy.begin(), Copy.end(), [&t](int k) {if (k == t) { cout << "Элемент найден" << endl; }});

}

void addmin(multiset<int>& lst) {

int mn = 100000;

for (auto& i : lst) {

mn = min(mn, i);

}

lst.insert(mn);

}

void pop\_key(multiset<int>& lst, int& key) {

vector<int> temp;

for (auto& i : lst) {

temp.push\_back(i);

}

multiset<int> pt;

bool fl = false;

for\_each(temp.begin(), temp.end(), [&fl, &key, &pt](int& p) {

if (fl == false && p == key) {

fl = true;

}

else { pt.insert(p); }});

lst = pt;

}

void add\_min\_max(multiset<int>& lst) {

multiset<int> ns;

vector<int> Copy;

int q;

size\_t lst\_sz = lst.size();

for (auto& i : lst) {

Copy.push\_back(i);

}

int min = 1000000;

int max = 0;

for\_each(Copy.begin(), Copy.end(), [&min, &max](int& t) {int g = t;if (max < g) { max = g; }if (min > g) { min = g; }});

for\_each(Copy.begin(), Copy.end(), [&min, &max](int& t) {t = t + min + max;});

for (auto i = Copy.begin(); i != Copy.end(); i++)

ns.insert(\*i);

lst = ns;

}

void task3() {

multiset<int> new\_list;

cout << "Введите размер очереди : ";

int n;

cin >> n;

make\_list(new\_list, n);

sshow(new\_list);

cout << "Добавим в конец минимальный элемент" << endl;

addmin(new\_list);

sshow(new\_list);

cout << "Проверим элемент на вхождение : " << endl;

int a;

cin >> a;

find\_time(new\_list, a);

cout << "Удалим элемент с заданным ключём" << endl;

cout << "Ключ:" << endl;

cin >> a;

pop\_key(new\_list, a);

sshow(new\_list);

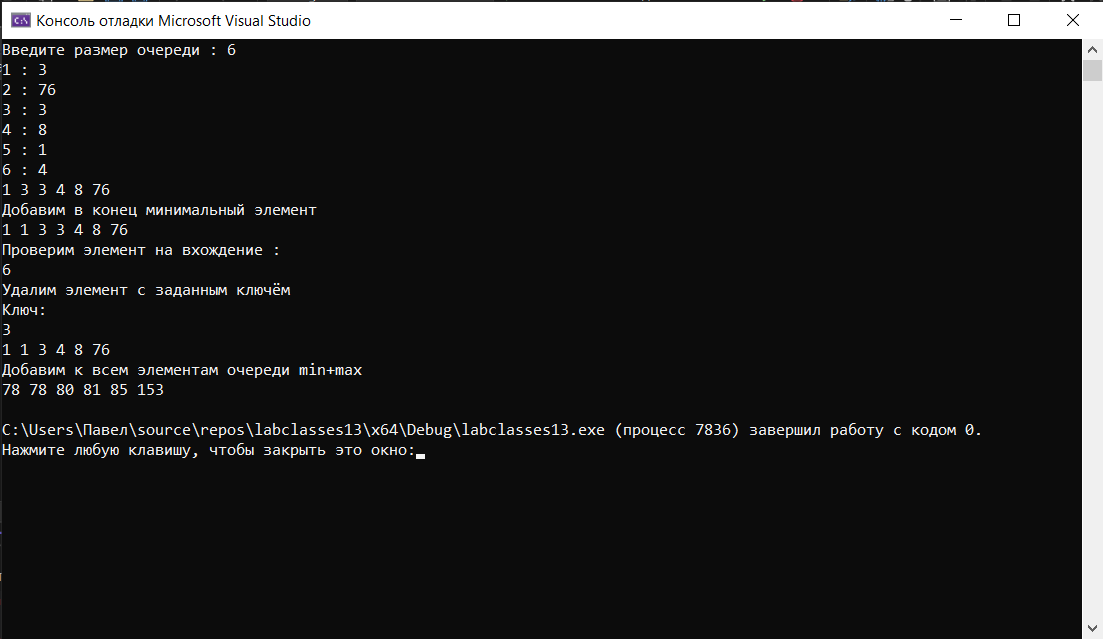
cout << "Добавим к всем элементам очереди min+max" << endl;

add\_min\_max(new\_list);

sshow(new\_list);

}

**Результат работы программы**

****

**Анализ результатов**

Удалось успешно реализовать задачи из лабораторной работы 12 с помощью стандартных алгоритмов STL.