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: «Классы и объекты. Инкапсуляция»

Выполнил студент ИВТ-22-2б:

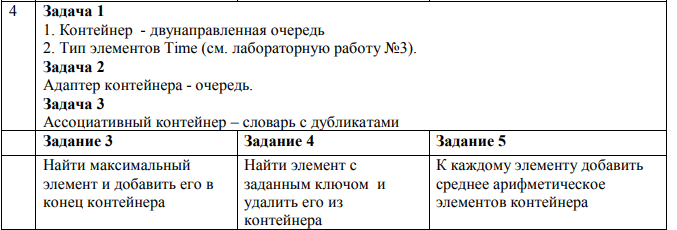
Захаров Дмитрий Сергеевич

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

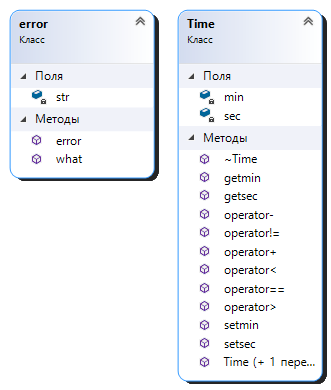
**1 Постановка задачи**

****

**2 Анализ задачи**

Создается двунаправленная очередь, и с помощью алгоритмов stl заполняется. В этой очереди находиться максимальный элемент и пушится в конец, так же высчитывается среднее арифметическое и добавляется к каждому элементу. То же самое с списком.

**3 UML-диаграмма**

****

**4 Код программы**

#include <iostream>

#include "Time.h"

#include "Error.h"

#include <algorithm>

#include <deque>

#include <iterator>

#include <functional>

using namespace std;

ostream& operator<<(ostream& out, deque<Time>& t) {

for\_each(t.begin(), t.end(), [](Time& p) {cout << p << ' '; });

return out;

};

int main()

{

try {

setlocale(LC\_ALL, "ru");

int n;

cout << "Введите размер очереди: ";

cin >> n;

deque<Time> i(n);

Time m;

//найти максимальный элемент и добавить его в конец очереди.

generate(i.begin(), i.end(), []() {Time p; cin >> p; return p; });

cout << "Очередь: " << endl << i << endl;

auto a = max\_element(i.begin(), i.end());

cout << "Максимальный элемент: " << \*a <<endl;

i.push\_back(\*a);

cout << "Очередь после добавления максимального элемента в конец: " << endl << i << endl;

//удалить элемент с заданным ключом

Time tmp;

cout << "Введите элемент: " <<endl;

cin >> tmp;

deque<Time>::iterator s = remove\_if(i.begin(), i.end(), [&tmp](Time& p) {return (tmp == p); });

i.erase(s, i.end());

cout << "Очередь после удаления: " << endl << i<< endl;

//добавление среднего арифметического

Time sum;

for\_each(i.begin(), i.end(), [&sum](Time& y) {sum = sum + y; });

sum.setmin(sum.getmin() / i.size());

sum.setsec(sum.getsec() / i.size());

for\_each(i.begin(), i.end(), [&sum](Time& y) {y = sum + y; });

cout << "Очередь после добавления среднего арифметического к каждому элементу: " << endl << i << endl;

}

catch (error a) {

a.what();

}

return 0;

}

**Time.h**

#pragma once

#include <iostream>

#include "Error.h"

using namespace std;

class Time {

private:

int min;

int sec;

public:

Time() {

min = 0;

sec = 0;

};

Time(int, int);

~Time();

bool operator>(Time& p) {

if (this->min != p.min)

return this->min > p.min;

else return this->sec > p.sec;

};

bool operator<(Time& p) {

if (this->min != p.min)

return this->min < p.min;

else return this->sec < p.sec;

};

void setmin(int min) {

if(min <60 && min >=0)

this->min = min;

};

void setsec(int sec) {

if(sec<60 && sec >=0)

this->sec = sec;

};

int getmin() {

return min;

};

int getsec() {

return sec;

};

friend ostream& operator<<(ostream& stream, const Time& tm);

friend istream& operator>>(istream& in, Time& tm);

Time operator-(int p) {

if (this->sec >= p)

return Time(this->min, this->sec - p);

else {

int tmp = this->min \* 60 + this->sec - p;

return Time(tmp / 60, tmp % 60);

}

};

Time operator+(Time& p) {

if (this->min + p.min < 60 && this->sec + p.sec)

return Time(this->min + p.min, this->sec + p.sec);

else if (this->min + p.min < 60) {

int tmp = this->min \* 60 + p.min \* 60 + this->sec + p.sec;

return Time(tmp / 60, tmp % 60);

}

else throw error("НЕДОПУСТИМОЕ ЗНАЧЕНИЕ ДЛЯ ВРЕМЕНИ!");

};

bool operator==(Time& t) {

return this->min == t.min && this->sec == t.sec;

};

bool operator !=(Time& t) {

return !(this->min == t.min && this->sec == t.sec);

};

};

istream& operator>>(istream& in, Time& tm) {

cout << "Введите минуты и секунды: ";

in >> tm.min >> tm.sec;

if (tm.min < 60 && tm.sec < 60 && tm.min >=0 &&tm.sec >=0)

return in;

else throw error("НЕДОПУСТИМОЕ ЗНАЧЕНИЕ ДЛЯ ВРЕМЕНИ!");

};

Time::Time(int min, int sec) {

if (min < 60 && min >=0 && sec >= 0 && sec < 60) {

this->min = min;

this->sec = sec;

}

else throw error("НЕДОПУСТИМОЕ ЗНАЧЕНИЕ ДЛЯ ВРЕМЕНИ!");

};

Time::~Time() {

};

ostream& operator<<(ostream& stream, const Time& tm) {

if (tm.sec >= 10 && tm.min >=10)

stream << tm.min << ':' << tm.sec;

else if (tm.sec <10 && tm.sec >=0 && tm.min >=10)

stream << tm.min << ":0" << tm.sec;

else if (tm.sec < 10 && tm.sec >= 0 && tm.min < 10 && tm.min >=0)

stream << '0' << tm.min << ":0" << tm.sec;

else if (tm.sec >=10 && tm.min < 10 && tm.min >= 0)

stream << '0' << tm.min << ':' << tm.sec;

return stream;

};

**Error.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

class error {

string str;

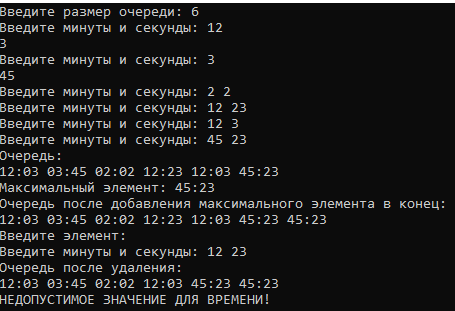
public:

error(string str) { this->str = str; };

void what() { cout << str << endl; };

};

**5 Результаты работы программы**

****