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: Шаблоны классов.

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Токарев Павел Аркадьевич

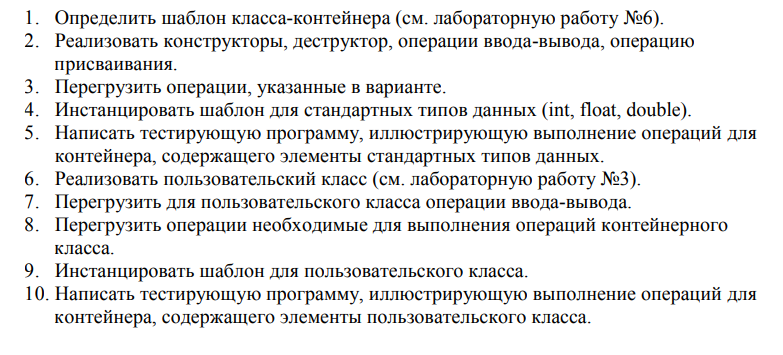
Проверил

Доцент кафедры ИТАС

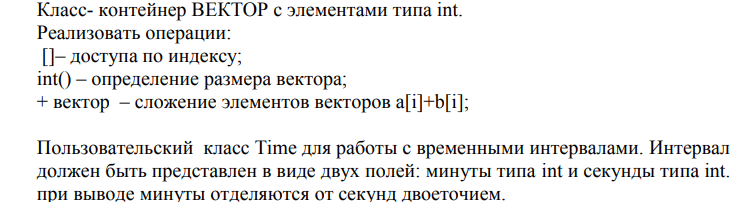
Полякова Ольга Андреевна

Г. Пермь-2023

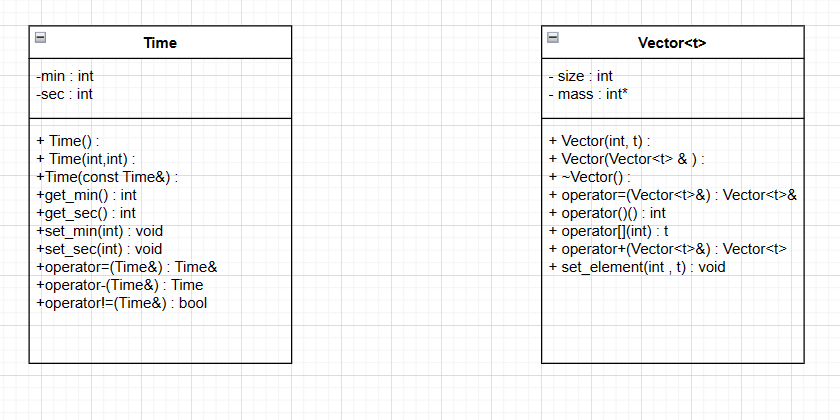
**Постановка задачи**

****

Вариант 2:

****

**Диаграмма UML**

****

**Код программы**

Time.h

#pragma once

#include <iostream>

using namespace std;

class Time {

public:

int minute;

int second;

Time(void);

Time(int, int);

Time(const Time&);

~Time();

friend ostream& operator<<(ostream&out,const Time&);

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

Time& operator=(const Time&);

Time operator-(const Time&);

Time operator+(const Time&);

bool operator!=(const Time&);

};

Time.cpp

#include "Time.h"

#include<iostream>

using namespace std;

Time::Time() {

minute = 0;

second = 0;

}

Time::Time(int a, int b) {

minute = a;

second = b;

}

Time::Time(const Time&a) {

minute = a.minute;

second = a.second;

}

Time::~Time() {

}

ostream& operator<<(ostream& out, const Time& a) {

out << a.minute << ":" << a.second;

return out;

}

Time Time::operator+(const Time& a) {

int tmp1 = (minute \* 60) + second;

int tmp2 = (a.minute \* 60) + a.second;

Time newtime;

newtime.minute = (tmp1 + tmp2) / 60;

newtime.second = (tmp1 + tmp2) % 60;

return newtime;

}

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

cout << "Минута:";

in >> a.minute;

cout << "Секунда:";

in >> a.second;

return in;

}

Time& Time::operator=(const Time& t) {

if (&t == this) {

return \*this;

}

else {

minute = t.minute;

second = t.second;

return \*this;

}

}

Time Time::operator-(const Time& t) {

int tmp1 = (minute \* 60) + second;

int tmp2 = (t.minute \* 60) + t.second;

Time newtime;

newtime.minute = (tmp1 - tmp2) / 60;

newtime.second = (tmp1 - tmp2) % 60;

return newtime;

}

bool Time::operator!=(const Time& t) {

if ((minute != t.minute) && (second != t.second)) {

return true;

}

else {

return false;

}

};

Vector.h

#pragma once

#include <iostream>

#include"Time.h"

#include<vector>

using namespace std;

template <typename t>

class Vector {

private:

int size;

vector<t> mass;

public:

Vector(int s, t k) {

size = s;

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

mass.push\_back(k);

}

}

Vector(const Vector<t>& a) {

size = a.size;

for (int i = 0;i < a.size;i++) {

mass.push\_back(a.mass[i]);

}

}

~Vector() {

}

friend ostream& operator<<(ostream& out, const Vector<t>& a) {

for (int i = 0;i < a.size;i++) {

out << i + 1 << " " << a.mass[i] << endl;

}

return out;

}

friend istream& operator>>(istream& in, Vector<t>& a) {

for (int i = 0;i < a.size;i++) {

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

in >> a.mass[i];

}

return in;

}

Vector<t>& operator=(const Vector<t>& a) {

for (int i = 0;i < a.size;i++) {

mass[i] = a.mass[i];

}

return \*this;

}

int operator()() {

return size;

}

t operator[](int k) {

return mass[k];

}

Vector<t> operator+(const Vector<t>&a) {

Time aw(2, 67);

Vector<t> tmp(a.size, aw);

for (int i = 0;i < a.size;i++) {

tmp.mass.pop\_back();

}

for (int i = 0;i < a.size;i++) {

tmp.mass.push\_back(mass[i] + a.mass[i]);

}

return tmp;

}

void set\_element(int index, t& a) {

mass[index] = a;

}

};

main.cpp

#include <iostream>

#include"Time.h"

#include"Vector.h"

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Заполните вектор1 для 2 элементов" << endl;

Time b(2,3),a(1,1);

Vector<Time>container1(2, a);

cin >> container1;

cout << "Заполните вектор2 для 2 элементов" << endl;

Vector<Time>container2(2,b);

cin >> container2;

cout << "Результат сложения 2 векторов" << endl;

Vector<Time>container3(2, b);

container3 = container1+container2;

cout << container3;

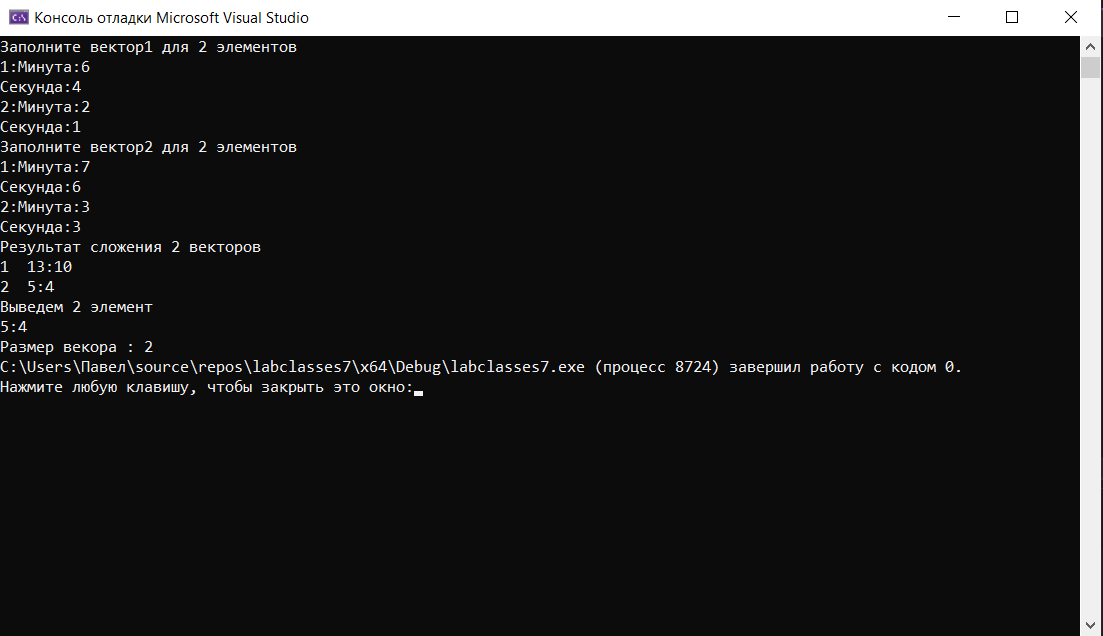
cout << "Выведем 2 элемент" << endl;

cout << container3[1]<<endl;

cout << "Размер векора : " << container3();

}

**Результат работы программы**

****

**Анализ результатов**

Удалось успешно реализовать шаблонный класс. Все методы в шаблонном классе работают для все типов данных.

**Контрольные вопросы**

1. В чем смысл использования шаблонов?

— Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разные типы данных.

2.Каковы синтаксис/семантика шаблонов функций?

— Template <class t> //t – имя параметризируемого типа

3.Каковы синтаксис/семантика шаблонов классов?

— Template <class T>

4.Что такое параметры шаблона функции?

— Параметр – это аргумент, который может принимать в себя различные типы данных.

5.Перечислите основные свойства параметров шаблона функции.

— Шаблон семейства функций состоит из 2 частей – заголовка шаблона: template и обыкновенного определения функции, в котором вместо типа возвращаемого значения и/или типа параметров, записывается имя типа, определенное в заголовке шаблона.

6.Как записывать параметр шаблона?

— template//type – имя параметризируемого типа

type abs(type x)

7.Можно ли перегружать параметризованные функции?

— Нет, подразумевается, что шаблонные функции уже заточены под работу с разными типами данных, при этом имя лишь одну реализацию.

8.Перечислите основные свойства параметризованных классов.

— Экземпляр создается либо объявлением объекта, либо объявлением указателя на инстанцированный шаблонный тип с присваиванием ему адреса с помощью операции new.

9.Все ли компонентные функции параметризованного класса являются параметризованными?

— Да

10.Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?

— Нет

11.Могут ли шаблоны классов содержать виртуальные компонентные функции?

— Нет

12.Как определяются компонентные функции параметризованных классов вне определения шаблона класса?

— Определение должно начинаться с ключевого слова template, за которым следует такой же список\_параметров\_типов в угловых скобках, какой указан в определении шаблона класса. За именем\_класса, предшествующим операции области видимости (::), должен следовать список\_имен\_параметров шаблона.

13.Что такое инстанцирование шаблона?

— Это генерация кода функции или класса по шаблону для конкретных параметров.

14.На каком этапе происходит генерирование определения класса по шаблону?

— Когда происходит явное или неявное инстанцирование.