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 ПО ЛАБОРАТОРНОЙ РАБОТЕ №7**

Дисциплина: «Основы алгоритмизации и программирования»

Тема : "Шаблоны классов"

Выполнил работу:

Студент группы РИС-22-2Б

Бехтольт Д.А.

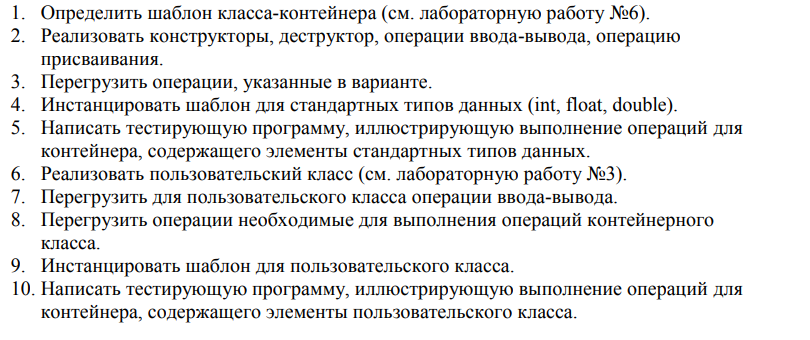
Проверил:

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**



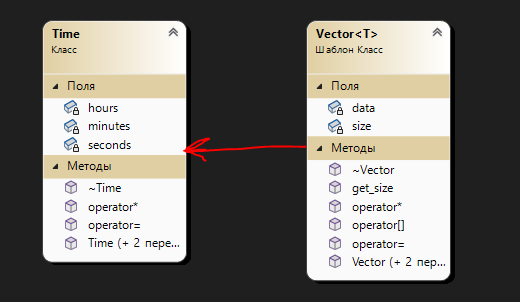
Вариант 5:

Шаблонный класс вектор

Пользовательский класс Time

Определить операторы присваивания, поиска по индексу умножения

**Диаграмма классов**

****

**Описание параметритизированного класса**

template <typename T>

class Vector {

private:

T\* data; // указатель на массив, хранящий элементы вектора

int size; // размер вектора

public:

Vector() : data(nullptr), size(0) {} // конструктор по умолчанию

Vector(int size, T value = T()) : size(size) { // конструктор, принимающий размер вектора и начальное значение элементов

data = new T[size];

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

data[i] = value;

}

}

~Vector() { // деструктор

delete[] data;

}

Vector(const Vector& other) : size(other.size) { // конструктор копирования

data = new T[size];

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

data[i] = other.data[i];

}

}

Vector& operator=(const Vector& other) { // оператор присваивания

if (this != &other) {

delete[] data;

size = other.size;

data = new T[size];

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

data[i] = other.data[i];

}

}

return \*this;

}

T& operator[](int index) { // перегруженный оператор доступа по индексу

return data[index];

}

int get\_size() const { // метод, возвращающий размер вектора

return size;

}

Vector operator\*(const Vector& other) const { // перегруженный оператор умножения векторов

int n = std::min(size, other.size); // определяем количество элементов для умножения

Vector result(n);

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

result.data[i] = data[i] \* other.data[i];

}

return result;

}

};}

**Описание пользовательского класса Time**

#pragma once

#include <iostream>

class Time {

private:

int hours; // часы

int minutes; // минуты

int seconds; // секунды

public:

Time() : hours(0), minutes(0), seconds(0) {} // конструктор по умолчанию

Time(int hours, int minutes, int seconds) : hours(hours), minutes(minutes), seconds(seconds) {} // конструктор, принимающий часы, минуты и секунды

Time(const Time& other) : hours(other.hours), minutes(other.minutes), seconds(other.seconds) {} // конструктор копирования

~Time() {} // деструктор

Time& operator=(const Time& other) { // оператор присваивания

if (this != &other) {

hours = other.hours;

minutes = other.minutes;

seconds = other.seconds;

}

return \*this;

}

friend std::ostream& operator<<(std::ostream& os, const Time& t) { // оператор вывода

os << t.hours << ":" << t.minutes << ":" << t.seconds;

return os;

}

friend std::istream& operator>>(std::istream& is, Time& t) { // оператор ввода

is >> t.hours >> t.minutes >> t.seconds;

return is;

}

Time operator\*(const Time& other) const { // оператор умножения времени

int total\_seconds = hours \* 3600 + minutes \* 60 + seconds; // переводим первое время в секунды

int other\_total\_seconds = other.hours \* 3600 + other.minutes \* 60 + other.seconds; // переводим второе время в секунды

int result\_total\_seconds = total\_seconds \* other\_total\_seconds; // перемножаем секунды

int result\_hours = result\_total\_seconds / 3600; // переводим обратно в часы

int result\_minutes = (result\_total\_seconds % 3600) / 60; // переводим обратно в минуты

int result\_seconds = result\_total\_seconds % 60; // переводим обратно в секунды

return Time(result\_hours, result\_minutes, result\_seconds);

}

};

**Функция main**

#include <iostream>

#include "Vector.h"

#include "Time.h"

int main()

{

Vector<int> v1(5, 2); // создаем вектор размера 5, заполненный двойками

Vector<int> v2(5, 3); // создаем вектор размера 5, заполненный тройками

// выводим элементы векторов

for (int i = 0; i < v1.get\_size(); i++) {

std::cout << v1[i] << " ";

}

std::cout << std::endl;

for (int i = 0; i < v2.get\_size(); i++) {

std::cout << v2[i] << " ";

}

std::cout << std::endl;

Vector<int> v3 = v1 \* v2; // производим поэлементное умножение векторов

// выводим элементы результата

for (int i = 0; i < v3.get\_size(); i++) {

std::cout << v3[i] << " ";

}

std::cout << std::endl;

Time t1(1, 10, 0);

Time t2(1, 5, 15);

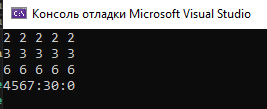
Time t3 = t1 \* t2;

std::cout << t3 << std::endl; // выводим произведение временных интервалов

return 0;

}

**Результаты работы программы**

****

**Ответы на контрольные вопросы**

1. В чем смысл использования шаблонов?

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные.

2. Каковы синтаксис/семантика шаблонов функций?

template<параметры\_шаблона>

заголовок функции

{тело функции}

пример:

template<class type>

type abs(type x)

{

if (x<0) return -x;

else return x;

}

3. Каковы синтаксис/семантика шаблонов классов?

template<параметры\_шаблона>

class имя\_класса

{ ... };

пример:

template<class T>

class Point

{

T x, y;

public:

Point(T x = 0, T y = 0):x(x), y(y){}

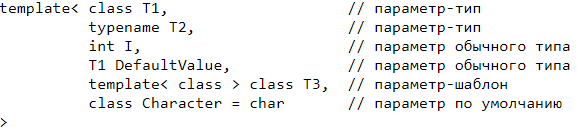
void Show();

{;

4. Что такое параметры шаблона функции?

Параметр шаблона - то, что указывается в <>.

Параметрами шаблонов могут быть:

* параметры-типы;
* параметры обычных типов;
* параметры-шаблоны.
* 

5. Перечислите основные свойства параметров шаблона функции.

- шаблоны не могут быть виртуальными

- шаблоны могут содержать статические элементы, дружественные функции и классы

- шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.

6. Как записывать параметр шаблона?

в угловых скобочках < >

7. Можно ли перегружать параметризованные функции?

Да, меняя тип данных параметра, либо меняя параметры местами, в том случае, если они разного типа, а также можно выполнять перегрузку функции добавляя или исключая параметры.

8. Перечислите основные свойства параметризованных классов.

- Параметры шаблона могут быть любыми типами данных, включая примитивные типы, пользовательские типы и другие шаблоны классов.

- Параметры шаблона могут быть как значениями, так и типами. Например, можно создать шаблон класса с параметром, который задает размер массива.

-Параметры шаблона могут использоваться внутри класса для объявления переменных, функций и типов.

-Каждый конкретный экземпляр параметризованного класса является отдельным типом, который определяется параметрами шаблона.

-Шаблонные классы могут быть унаследованы от других шаблонных классов и могут иметь шаблонные функции-члены.

9. Bce ли компонентные функции параметритизированного класса являются параметризованными?

Да, все компонентные функции (методы) параметризированного класса являются параметризованными, то есть они используют те же параметры шаблона, что и сам класс.

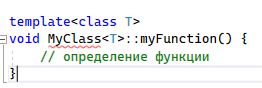
10. Являются ли дружественные функции, описанные в  
классе, параметризованными?  
Да, дружественные функции, описанные внутри класса, являются параметризованными, если класс является параметризованным. Дружественные функции имеют доступ к закрытым членам класса, но они также должны быть параметризованы, чтобы иметь доступ к параметрам шаблона класса.

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Да, шаблоны классов могут содержать виртуальные компонентные функции. Виртуальные функции позволяют реализовать полиморфизм и позволяют вызывать функцию на объекте, указатель на который имеет тип базового класса, но указывает на объект, созданный из класса-наследника.

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

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса с помощью использования template<class T> перед каждым определением функции, где T - это параметр шаблона.



13. Что такое инстанцирование шаблона?

Инстанцирование шаблона - это процесс создания конкретной реализации шаблона класса или функции на основе заданных параметров шаблона. При инстанцировании компилятор создает отдельный экземпляр шаблона для конкретных аргументов шаблона. То есть компилятор подставляет значения параметров шаблона в шаблонный код и создает реализацию класса или функции, которая может быть использована в программе.

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

На этапе компиляции.