Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №7

Тема: «Шаблоны классов»

Выполнил

Студент группы РИС-22-2б

Арапов М.В.

Проверил доц. Кафедры ИТАС

Полякова О. А.

Пермь 2023

# Постановка задачи

Класс- контейнер ВЕКТОР с элементами типа int.

Реализовать операции:

[] - доступа по индексу;

+ вектор - сложение элементов векторов а[i]+b[i];

+ число - добавляет константу ко всем элементам вектора;

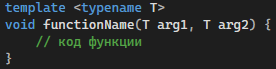
Пользовательский класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. При выводе минуты отделяются от секунд двоеточием.

**Контрольные вопросы**

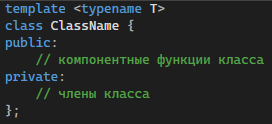
1. В чем смысл использования шаблонов?

Использование шаблонов позволяет создавать обобщенный код, который может работать с различными типами данных без необходимости написания отдельного кода для каждого типа. Это упрощает разработку и поддержку кода, а также улучшает его переносимость и гибкость.

1. Каковы синтаксис/семантика шаблонов функций?

Синтаксис шаблонов функций выглядит следующим образом: 

1. Каковы синтаксис/семантика шаблонов классов?

Синтаксис шаблонов классов выглядит следующим образом: 

1. Что такое параметры шаблона функции?

Параметры шаблона функции - это значения, которые передаются в шаблон функции при его вызове и заменяют параметры шаблона на конкретные типы данных.

1. Перечислите основные свойства параметров шаблона функции.

Основные свойства параметров шаблона функции:

- они могут быть любого типа данных;

- они могут быть константами или ссылками на объекты;

- они могут иметь значение по умолчанию.

6. Как записывать параметр шаблона?

Параметр шаблона записывается в угловых скобках после ключевого слова `template`, например:

template <typename T>

1. Можно ли перегружать параметризованные функции?

Да, можно перегружать параметризованные функции, при этом каждая перегруженная функция должна иметь уникальный список параметров.

1. Перечислите основные свойства параметризованных классов.

Основные свойства параметризованных классов:

- они могут иметь любое количество параметров;

- параметры могут быть любого типа данных;

- параметры могут быть константами или ссылками на объекты;

- параметры могут иметь значение по умолчанию.

9. Все ли компонентные функции параметризованного класса являются параметризованными?

Нет, не все компонентные функции параметризованного класса являются параметризованными. Некоторые функции могут использовать только общие члены класса, которые не зависят от параметров шаблона.

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

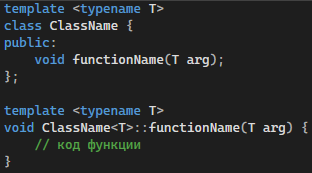
Дружественные функции, описанные в параметризованном классе, могут быть параметризованными или не параметризованными, в зависимости от того, нужно ли им использовать параметры шаблона.

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Да, шаблоны классов могут содержать виртуальные компонентные функции.

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

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса следующим образом:



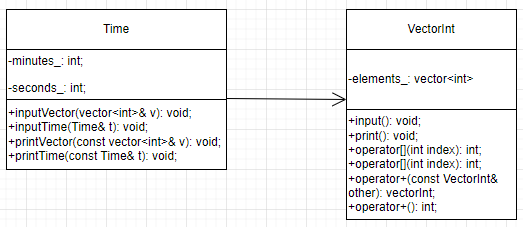
13. Что такое инстанцирование шаблона?

Инстанцирование шаблона - это процесс создания конкретной версии шаблона с заменой параметров шаблона на конкретные типы данных.

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

Генерирование определения класса по шаблону происходит на этапе компиляции, когда вызывается шаблонная функция или создается объект класса с конкретными параметрами шаблона. На этом этапе компилятор заменяет параметры шаблона на конкретные типы данных и генерирует определение класса.

UML таблица



Код программы

#include <iostream>

#include <vector>

using namespace std;

class Time {

public:

Time(int minutes = 0, int seconds = 0) : minutes\_(minutes), seconds\_(seconds) {}

void setMinutes(int minutes) { minutes\_ = minutes; }

void setSeconds(int seconds) { seconds\_ = seconds; }

int getMinutes() const { return minutes\_; }

int getSeconds() const { return seconds\_; }

private:

int minutes\_;

int seconds\_;

};

void inputVector(vector<int>& v) {

int n;

cout << "Enter the size of the vector: ";

cin >> n;

v.resize(n);

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

cout << "Enter element " << i << ": ";

cin >> v[i];

}

}

void inputTime(Time& t) {

int m, s;

cout << "Enter minutes: ";

cin >> m;

cout << "Enter seconds: ";

cin >> s;

t = Time(m, s);

}

void printVector(const vector<int>& v) {

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

cout << v[i] << " ";

}

cout << endl;

}

void printTime(const Time& t) {

cout << t.getMinutes() << ":" << t.getSeconds() << endl;

}

class VectorInt {

public:

VectorInt() {}

VectorInt(int size) : elements\_(size) {}

VectorInt(const VectorInt& other) : elements\_(other.elements\_) {}

~VectorInt() {}

void input() {

inputVector(elements\_);

}

void print() const {

printVector(elements\_);

}

int operator[](int index) const {

return elements\_[index];

}

int& operator[](int index) {

return elements\_[index];

}

VectorInt operator+(const VectorInt& other) const {

VectorInt result(elements\_.size());

for (int i = 0; i < elements\_.size(); i++) {

result[i] = elements\_[i] + other[i];

}

return result;

}

int operator+() const {

return elements\_.size();

}

private:

vector<int> elements\_;

};

int main() {

VectorInt v1, v2;

cout << "Enter the first vector:" << endl;

v1.input();

cout << "Enter the second vector:" << endl;

v2.input();

cout << "First vector elements: ";

v1.print();

cout << "Second vector elements: ";

v2.print();

VectorInt v3 = v1 + v2;

cout << "Sum of vectors: ";

v3.print();

int size = +v1;

cout << "Size of the first vector: " << size << endl;

Time t1, t2;

cout << "Enter the first time interval:" << endl;

inputTime(t1);

cout << "Enter the second time interval:" << endl;

inputTime(t2);

cout << "First time interval: ";

printTime(t1);

cout << "Second time interval: ";

printTime(t2);

return 0;

}