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 № 18.7**

Дисциплина: Информатика

Тема: «Объектно-ориентированное программирование. Шаблоны классов»

Вариант 1

Выполнил работу:

студент группы РИС-20-1Б

Азмагулов Артём Вадимович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь

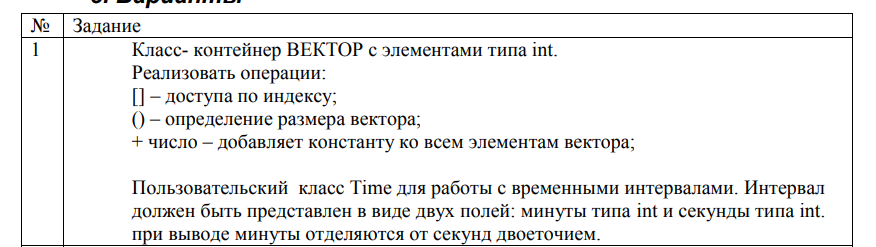
2021

**Цель работы**

* 1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
  2. Реализация шаблона класса-контейнера.

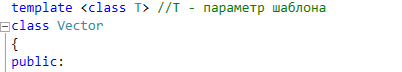
**Постановка задачи**

1. Определить шаблон класса-контейнера (см. лабораторную работу №6).
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Инстанцировать шаблон для стандартных типов данных (int, float, double).
5. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.
6. Реализовать пользовательский класс (см. лабораторную работу №3).
7. Перегрузить для пользовательского класса операции ввода-вывода.
8. Перегрузить операции необходимые для выполнения операций контейнерного класса.
9. Инстанцировать шаблон для пользовательского класса.
10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.

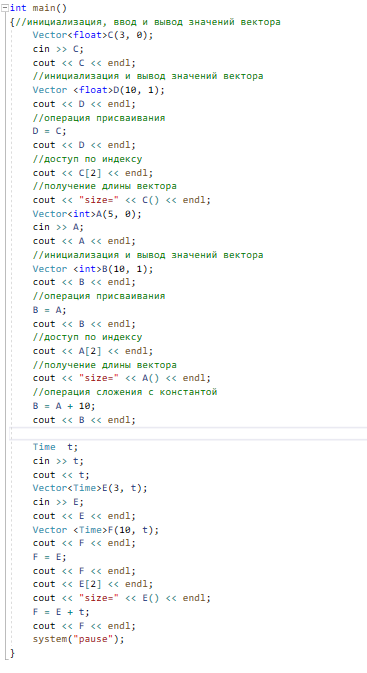


**Анализ задачи**

1. Необходимые действия
2. Реализовать шаблон класса-контейнера для работа с данными разных типов



1. Взять реализацию класса Time из лабораторной работы номер 6
2. Продемонстрировать работу множества с типом int, float и double (операции над данными будут одинаковыми, отличие будет лишь в типе передаваемых данных)
3. Продемонстрировать все функции класса:



1. Программный код

Vector.h

#include <iostream>

using namespace std;

template <class T> //T - параметр шаблона

class Vector

{

public:

//конструктор с параметрами: выделяет память под s элементов и заполняет их

//значением k

Vector(int s, T k);

//конструктор с параметрами

Vector(const Vector<T>& a);

//деструктор

~Vector();

//оператор присваивания

Vector& operator=(const Vector<T>& a);

//операция доступа по индексу

T& operator[](int index);

//операция для добавление константы

Vector operator+(const T k);

//операция, возвращающая длину вектора

int operator()();

//перегруженные операции ввода-вывода

// <> - указывают на то, что функция является шаблоном

friend ostream& operator<< <>(ostream& out, const Vector<T>& a);

friend istream& operator>> <>(istream& in, Vector<T>& a);

private:

int size;//размер вектора

T\* data;//укзатель на динамический массив значений вектора

};

template <class T>

Vector<T>::Vector(int s, T k)

{

size = s;

data = new T[size];

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

data[i] = k;

}

//конструктор копирования

template <class T>

Vector<T>::Vector(const Vector& a)

{

size = a.size;

data = new T[size];

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

data[i] = a.data[i];

}

//деструктор

template <class T>

Vector<T>::~Vector() {

delete[]data;

data = 0;

}

//операция присваивания

template <class T>

Vector<T>& Vector<T>::operator=(const Vector<T>& a)

{

if (this == &a)return \*this;

size = a.size;

if (data != 0) delete[]data;

data = new T[size];

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

data[i] = a.data[i];

return \*this;

}

//операция доступа по индексу

template <class T>

T& Vector<T>::operator[](int index)

{

if (index < size) return data[index];

else cout << "\nError! Index>size";

}

//операция для добавления константы

template <class T>

Vector<T> Vector<T>::operator+(const T k)//+k

{

Vector<T> temp(size, k);//инициализируем временный вектор любым значением

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

temp.data[i] = data[i] + k;

return temp;

}

//операция для получения длины вектора

template <class T>

int Vector<T>::operator ()()

{

return size;

}

//операции для ввода-вывода

template <class T>

ostream& operator<< (ostream& out, const Vector<T>& a)

{

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

out << a.data[i] << " ";

return out;

}

template <class T>

istream& operator>> (istream& in, Vector<T>& a)

{

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

in >> a.data[i];

return in;}

main.cpp

// 18.7.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include "Vector.h"

#include <iostream>

#include "Time.h"

using namespace std;

int main()

{//инициализация, ввод и вывод значений вектора

Vector<float>C(3, 0);

cin >> C;

cout << C << endl;

//инициализация и вывод значений вектора

Vector <float>D(10, 1);

cout << D << endl;

//операция присваивания

D = C;

cout << D << endl;

//доступ по индексу

cout << C[2] << endl;

//получение длины вектора

cout << "size=" << C() << endl;

Vector<int>A(5, 0);

cin >> A;

cout << A << endl;

//инициализация и вывод значений вектора

Vector <int>B(10, 1);

cout << B << endl;

//операция присваивания

B = A;

cout << B << endl;

//доступ по индексу

cout << A[2] << endl;

//получение длины вектора

cout << "size=" << A() << endl;

//операция сложения с константой

B = A + 10;

cout << B << endl;

Time t;

cin >> t;

cout << t;

Vector<Time>E(3, t);

cin >> E;

cout << E << endl;

Vector <Time>F(10, t);

cout << F << endl;

F = E;

cout << F << endl;

cout << E[2] << endl;

cout << "size=" << E() << endl;

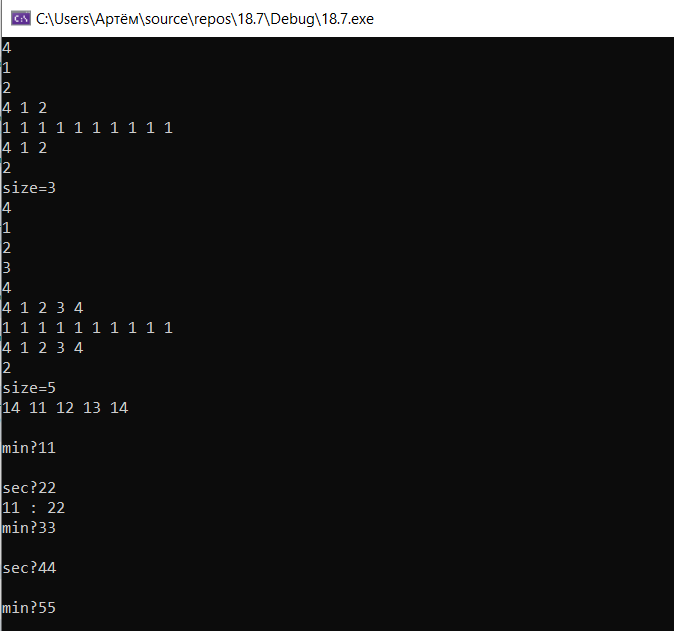
F = E + t;

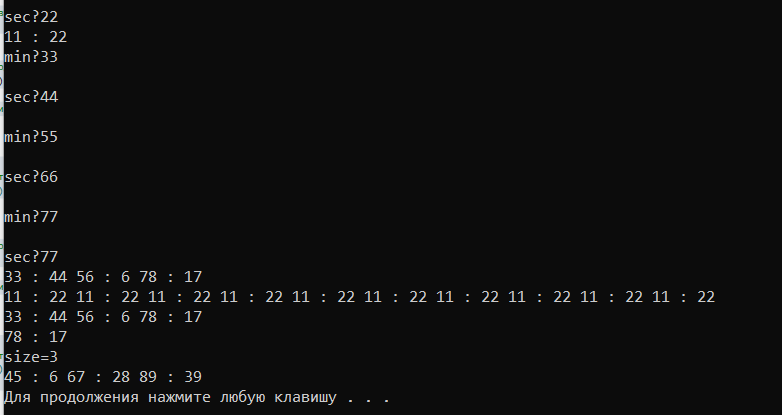
cout << F << endl;

system("pause");

}

1. Демонстрация работы программы





1. Контрольные вопросы
   1. В чем смысл использования шаблонов?

- С помощью шаблона можно отделить алгоритм от конкретных типов данных, передавая тип в качестве параметра.

* 1. Каковы синтаксис/семантика шаблонов функций?

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

template<параметры шаблона>

заголовок\_функции

{тело функции}

* 1. Каковы синтаксис/семантика шаблонов классов?

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

template<параметры шаблона>

class имя\_класса

{…};

* 1. Что такое параметры шаблона функции?

- тип данных, с которым будет работать шаблонная функция

* 1. Перечислите основные свойства параметров шаблона функции.

- Имена параметров шаблона должны быть уникальными во всем определении шаблона.  
− Список параметров шаблона не может быть пустым.  
− В списке параметров шаблона может быть несколько параметров, и каж-дому из них должно предшествовать ключевое слово class.  
− Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.  
− Определенная с помощью шаблона функция может иметь любое количество непараметризированных формальных параметров. Может быть непараметризированно и возвращаемое функцией значение.  
− В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона.  
− При конкретизации параметризованной функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были одинаковы.

* 1. Как записывать параметр шаблона?

- параметр шаблона необходимо записывать в угловых скобках <>

* 1. Можно ли перегружать параметризованные функции?
     + нельзя перегружать параметризованные функции
  2. Перечислите основные свойства параметризованных классов.

- Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью template.

- Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.

- Если friend-функция содержит в своем описании параметр типа параметризованного класса, то для каждого созданного по данному шаблону класса имеется собственная friend-функция.

- В рамках параметризованного класса нельзя определить friend-шаблоны (дружественные параметризованные классы).

- С одной стороны, шаблоны могут быть производными (наследоваться) как от шаблонов, так и от обычных классов, с другой стороны, они могут использоваться в качестве базовых для других шаблонов или классов.

- Шаблоны функций, которые являются членами классов, нельзя описывать как virtual.

- Локальные классы не могут содержать шаблоны в качестве своих элементов.

* 1. Все ли компонентные функции параметризованного класса являются параметризованными?

- Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью template.

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

- Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.

* 1. Могут ли шаблоны классов содержать виртуальные компонентные функции?

- шаблоны классов не могут содержать виртуальные компонентные функции

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

template <class tt>

class A {

private:

tt number;

public:

void set\_number(tt num);

tt get\_number();

};

template <class tt>

void A<tt>::set\_number(tt num) {

number = num;

}

template <class tt>

tt A<tt>::get\_number() {

return number;

}

* 1. Что такое инстанцирование шаблона?

- Процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона называется инстанцированиемшаблона

* 1. На каком этапе происходит генерирование определения класса по шаблону?

- генерирование определения класса по шаблону происходит на этапе создание экземпляра шаблонного класса с указанием конкретного типа