Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.7**

Дисциплина: «Информатика»

Тема: “Объектно-ориентированное программирование.

Шаблоны классов”

Вариант 19

Выполнил:

Студент группы РИС-20-2Б

Пономарёв Артём Викторович

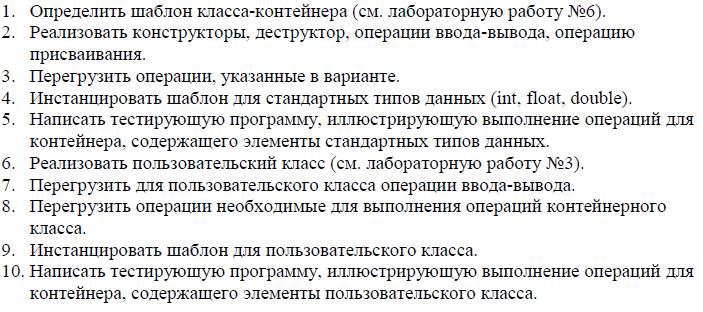
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



2

**Анализ задачи**

**1.** Для решения задачи необходимо:

**1.1.** Организовать класс Time для хранения минут (mins), секунд (secs).

**1.2.** Организовать необходимые методы для ввода данных в поля mins и secs: гетторы, сетторы, конструкторы, деструктор.

**1.3.** Организовать перегрузку оператора >> дружественной классу Time.

**1.4.** Организовать перегрузку оператора << дружественной классу Time.

**1.5.** Организовать перегрузку оператора == дружественной классу Time.

**1.6.** Организовать перегрузку оператора != дружественной классу Time.

**1.7.** Организовать перегрузку оператора + дружественной классу Time.

**1.8.** Организовать класс Vector с полями size для размерности массива и указатель arr на первый элемент массива.

**1.9.** Организовать перегрузку оператора << дружественной классу Vector.

**2.0.** Организовать перегрузку оператора () дружественной классу Vector.

**2.1.** Организовать перегрузку оператора [] дружественной классу Vector.

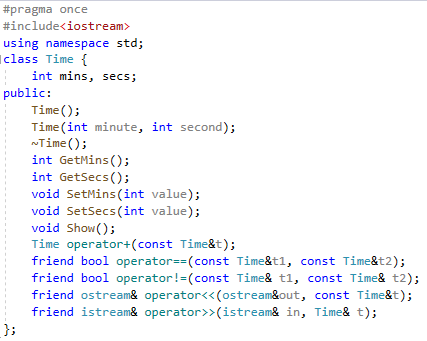
**2.2.** Организовать перегрузку оператора = дружественной классу Vector.

**2.3.** Организовать перегрузку оператора + дружественной классу Vector.

**2.4.** Организовать перегрузку оператора = дружественной классу Vector.

**2.** В ходе работы были использованы следующие типы данных:

**2.1.** Организовать класс Time для хранения минут (mins), секунд (secs). в заголовочном файле Time.h.

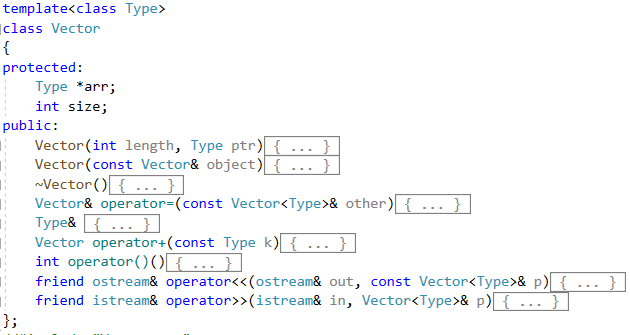


3

**2.2.** Переменные типа int: min, sec, где min – введённое количество минут, sec – введённое количество секунд.

int min, sec;

**2.3.** Организовать класс Vector с полями size для размерности массива и указатель arr на первый элемент массива.



**2.4.** Объекты класса Vector A, B – массивы типа Time.

Vector<Time>A(5, a);

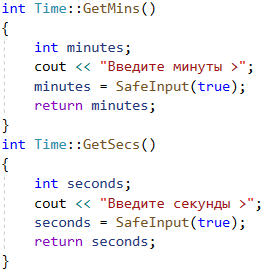
Vector<Time>B(3, b);

**2.5.** Объекты a, b класса Time.

Time a,b;

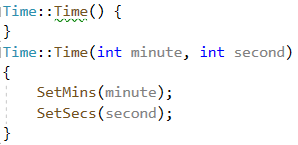
**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Данные вводятся через объекты класса Time через гетторы.



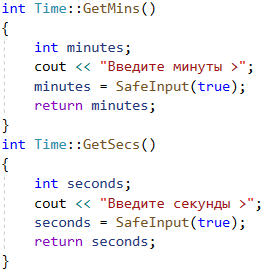
4

**3.2.** Данные также вводятся через объекты класса Time через конструктор с параметрами.

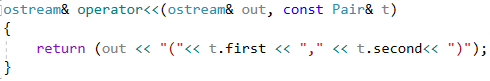


**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для ввода временных интервалов используются гетторы (методы, описанные в классе для ввода), в которых используется функция SafeInput, где используются проверки для корректной работы программы, в которой используется функция cin.



**4.2.** Вывод на консоль может быть осуществлён с помощью перегрузки оператора << для класса Time.



cout << "sum " << a + b << endl;

**4.3.** Вывод на консоль может быть осуществлён с помощью перегрузки оператора << для класса Vector.

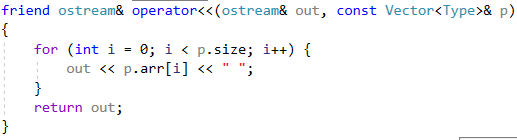
cout <<"Вектор А: " <<A << endl;

cout << "Вектор B: " << B << endl;

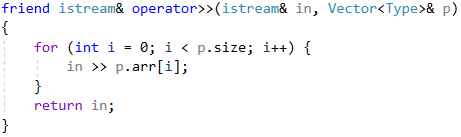
5

cout << "\nA[2]=" << A[2] << endl;

cout << "\nA+a: " <<B;

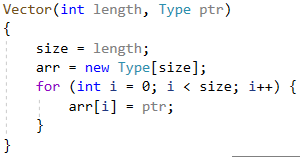


**4.4.** Для ввода элементов в массив класса Vector также может быть использована перегрузка оператора >>

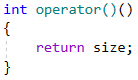


**5.** Поставленные задачи будут решены следующими действиями:

**5.1.** В конструкторе класса Vector в качестве параметров передаётся размерность массива и объект, который надо поместить в массив.

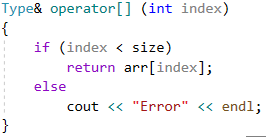


**5.2.** В перегрузке оператора () класса Vector производится вывод размерности объекта класса Vector.



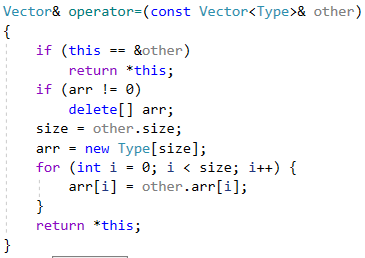
6

**5.3.** В перегрузке оператора [] класса Vector производится вызов элемента множества по индексу объекта класса Vector.



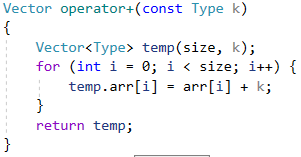
**5.4.** В перегрузке оператора = класса Vector производится вызов элемента множества по индексу объекта класса Vector.

В поле size присваивается значение size нового элемента (other), создаётся новый массив с размерностью size и в цикле for заполняется новый массив из массива объекта other.



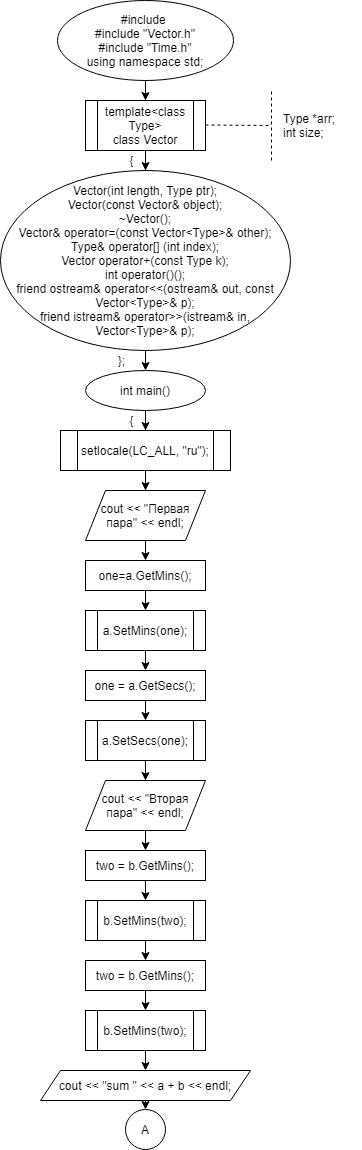
**5.5.** В перегрузке оператора + класса Vector производится вызов элемента множества по индексу объекта класса Vector.

Вводится вспомогательный объект класса Vector temp. В цикле for через поле arr присваивается значение arr и добавляется константа, которая передаётся параметром.

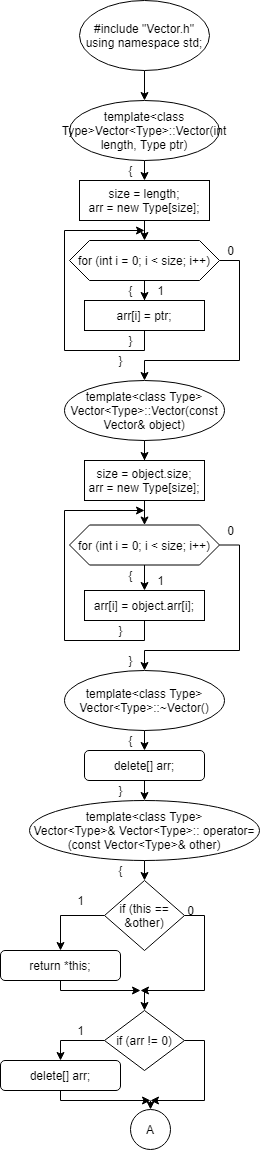


7

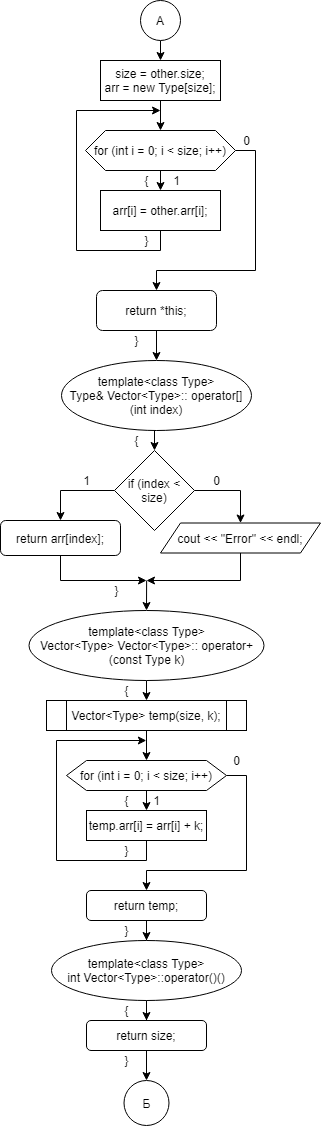
**Блок-схема**



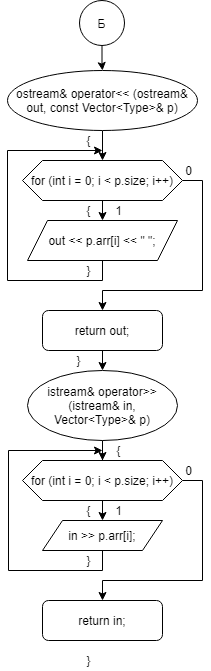
8



9



10



11

**Код**

#include <iostream>

#include "Vector.h"

#include "Time.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

Time a,b; int one, two;

cout << "Первая пара" << endl;

one=a.GetMins();

a.SetMins(one);

one = a.GetSecs();

a.SetSecs(one);

cout << "Вторая пара" << endl;

two = b.GetMins();

b.SetMins(two);

two = b.GetSecs();

b.SetSecs(two);

cout << "sum " << a + b << endl;

Vector<Time>A(5, a);

cout <<"Вектор А: " <<A << endl;

Vector<Time>B(3, b);

cout << "Вектор B: " << B << endl;

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

<< "size B " << B() << endl;

B = A;

cout << "\nВектор B после присвоения вектора A: " << B << endl;

cout << "\nA[2]=" << A[2] << endl;

B = A + a;

cout << "\nA+a: " <<B;

return 0;

}

12

#pragma once

#include<iostream>

using namespace std;

template<class Type>

class Vector

{

protected:

Type \*arr;

int size;

public:

Vector(int length, Type ptr); //конструктор с параметрами

Vector(const Vector& object); //конструктор копирования

~Vector(); //очистка памяти

Vector& operator=(const Vector<Type>& other);

Type& operator[] (int index); //получение значения по индексу

Vector operator+(const Type k); //сложение элемента массива с константой

int operator()(); //получение размера

friend ostream& operator<< (ostream& out, const Vector<Type>& p)

{

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

out << p.arr[i] << " ";

}

return out;

}

friend istream& operator>> (istream& in, Vector<Type>& p)

{

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

in >> p.arr[i];

}

return in;

}

};

#include "Vector.cpp"

13

#ifndef \_VECTOR\_CPP\_

#define \_VECTOR\_CPP\_

#include "Vector.h"

#include <assert.h>

using namespace std;

template<class Type>

Vector<Type>::Vector(int length, Type ptr)

{

size = length;

arr = new Type[size];

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

arr[i] = ptr;

}

template<class Type>

Vector<Type>::Vector(const Vector& object)

{

size = object.size;

arr = new Type[size];

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

arr[i] = object.arr[i];

}

template<class Type>

Vector<Type>::~Vector()

{

delete[] arr;

}

template<class Type>

Vector<Type>& Vector<Type>:: operator=(const Vector<Type>& other)

{

if (this == &other)

return \*this;

if (arr != 0)

delete[] arr;

size = other.size;

arr = new Type[size];

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

arr[i] = other.arr[i];

}

return \*this;

}

template<class Type>

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

{

if (index < size)

return arr[index];

else

cout << "Error" << endl;

}

template<class Type>

Vector<Type> Vector<Type>:: operator+(const Type k)

{

Vector<Type> temp(size, k);

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

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

}

return temp;

}

template<class Type>

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

{

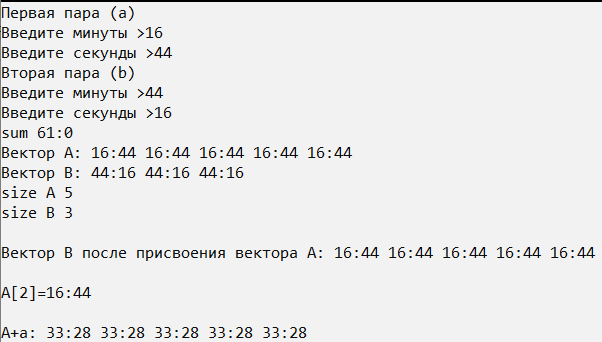
return size;

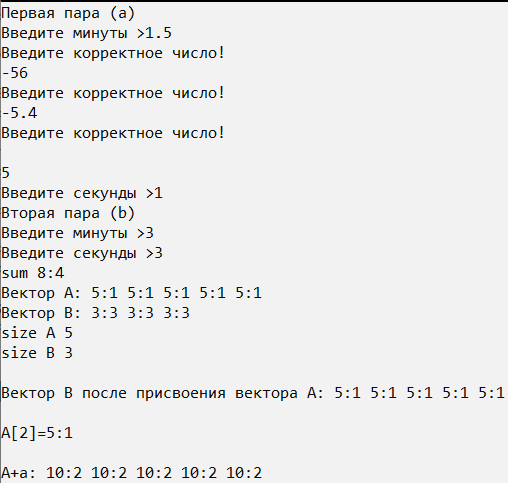
}

#endif13

14

**Скриншоты**

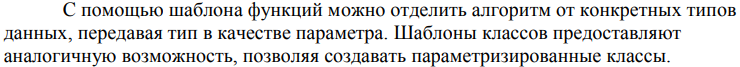




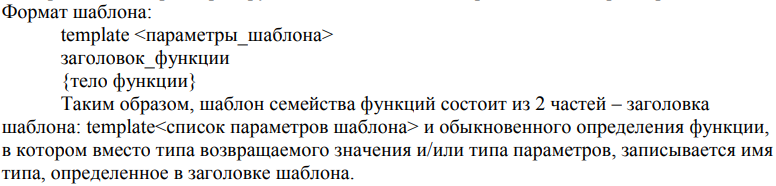
15

**Контрольные вопросы**

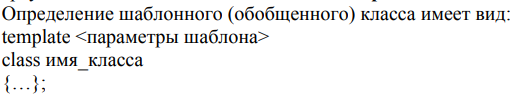


1. 



2. 



3. 



4. Параметры шаблона функции – это параметризированные типы данных, в зависимости от передаваемых данных в параметр.



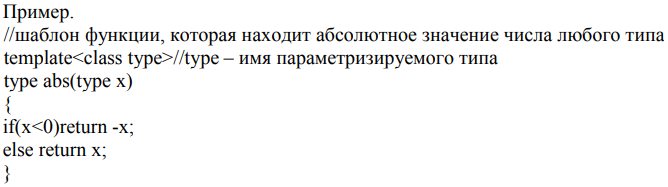
5. Шаблон функции объявляется один раз. Шаблон семейства функции состоит из 2 частей – заголовка шаблона и обыкновенного определения функции.



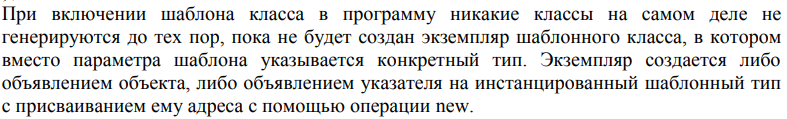
6. Параметр шаблона записывается в определении шаблона: template<typename Type>, далее в описании переменных и методов вместо типа данных необходимо записывать Type.



16

7. Да, можно. 



8. 



9. Компонентные функции параметризованного класса автоматически являются параметризованными.



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



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



12. Реализация компонентной функции шаблона класса, которая находится вне определения шаблона класса, должна включать дополнительно следующие два элемента:

17

* 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.

template<список\_типов>

тип\_возвр\_значения имя\_класса<список\_имен\_параметров>::имя\_функции(список\_параметров)

{ ... }



13. Процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона называется инстанцированием шаблона.



14. На этапе создания объекта перегруженного класса создаётся определение класса по шаблону.

18