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 4

Выполнил:

Студент группы РИС-20-1Б

Еске Вячеслав Сергеевич

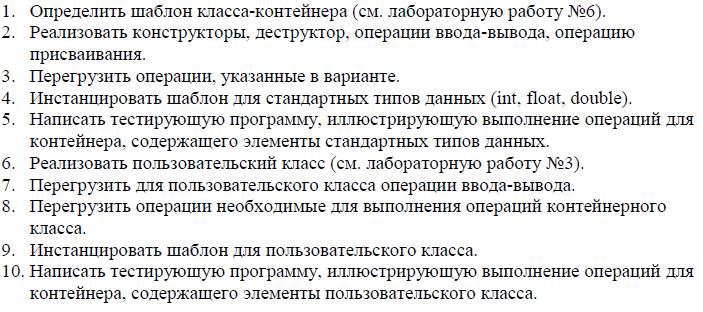
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



Класс- контейнер ВЕКТОР с элементами типа int.   
Реализовать операции:  
[] – доступа по индексу;

() – определение размера вектора;   
\* число – умножает все элементы вектора на число;

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.

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.

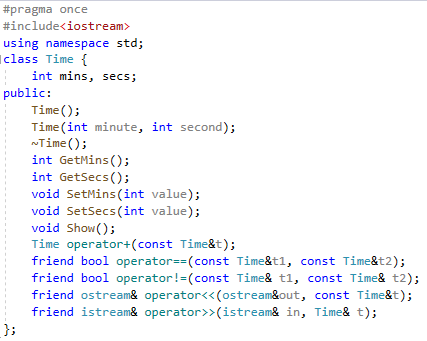
**2.2.** Организовать перегрузку оператора = дружественной классу Vector.

**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.

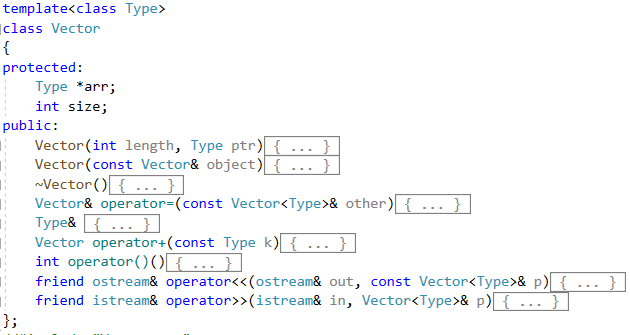


3

**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);

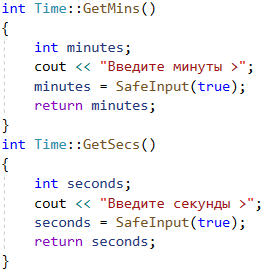
Vector<Time>B(3, b);

**2.5.** Объекты a, b класса Time.

Time a,b;

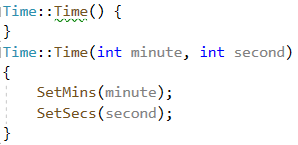
**3.** Для решения задачи данные были представлены в следующем виде:

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



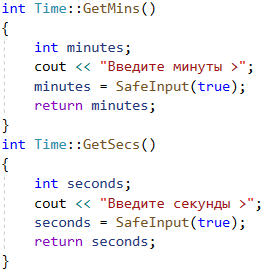
4

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

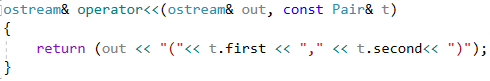


**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**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.

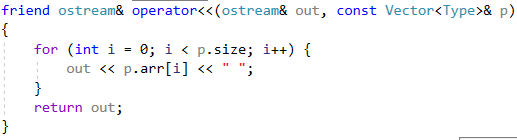
cout <<"Вектор А: " <<A << endl;

cout << "Вектор B: " << B << endl;

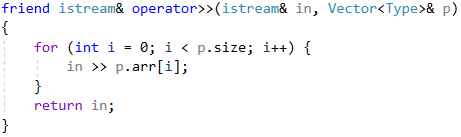
5

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

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

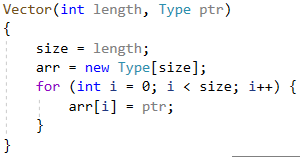


**4.4.** Для ввода элементов в массив класса Vector также может быть использована перегрузка оператора >>

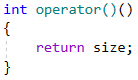


**5.** Поставленные задачи будут решены следующими действиями:

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

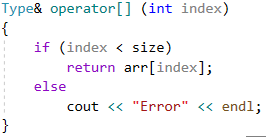


**5.2.** В перегрузке оператора () класса Vector производится вывод размерности объекта класса Vector.



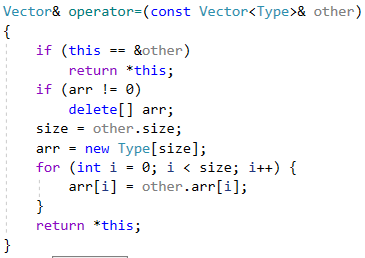
6

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



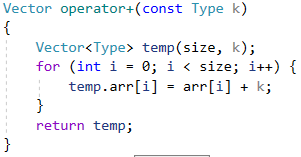
**5.4.** В перегрузке оператора = класса Vector производится вызов элемента множества по индексу объекта класса Vector.

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.



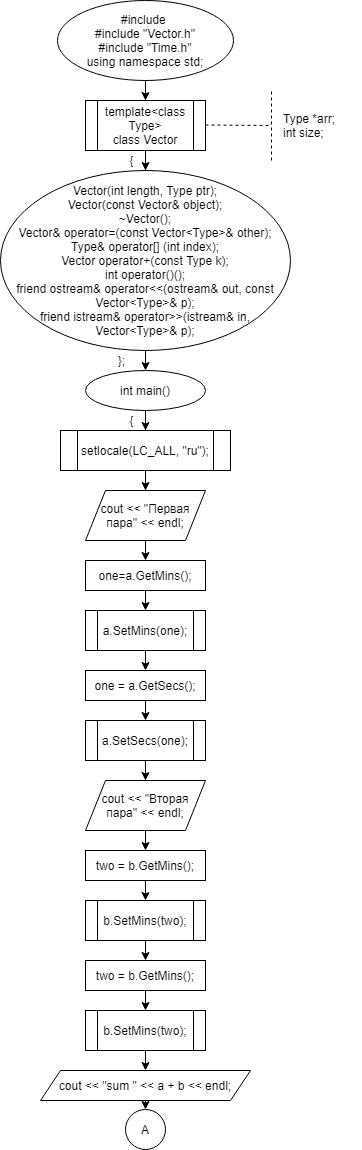
**5.5.** В перегрузке оператора + класса Vector производится вызов элемента множества по индексу объекта класса Vector.

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

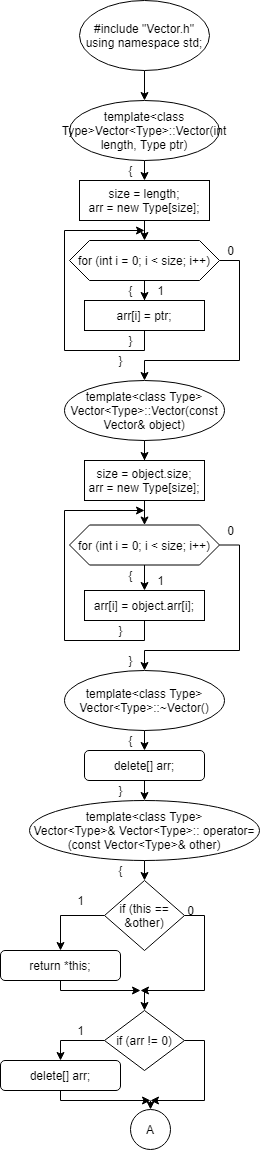


7

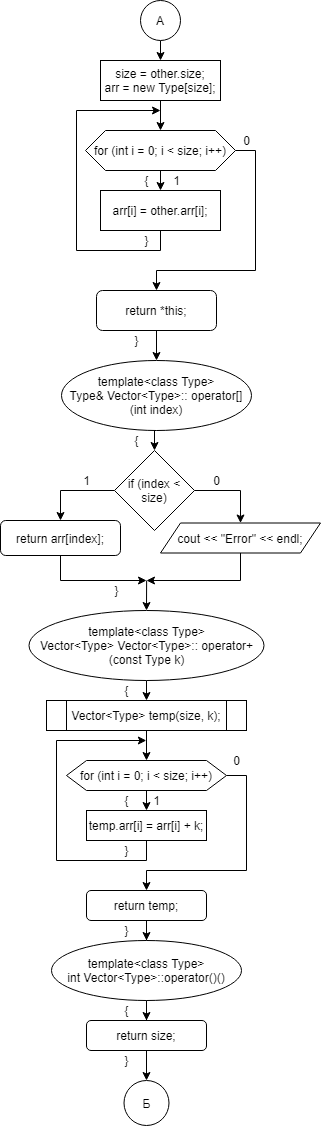
**Блок-схема**



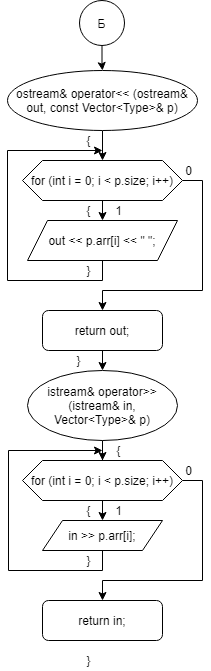
8



9



10



11

**Код**

#include <iostream>

#include "Vector.h"

#include "Time.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

/\*//инициализация, ввод и вывод значений вектора

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 умножения на константу

B=A\*10;

cout << B << endl;

\*/

cout << "Введите t" << endl;

Time t;

cin >> t;

cout << "t: " << endl;

cout << t << endl;

Vector<Time>A(5, t);

cout << "Введите вектор А" << endl;

cin >> A;

cout << "Вектор А" << endl;

cout << A << endl;

Vector <Time>B(10, t);

cout << "Вектор B" << endl;

cout << B << endl;

B = A;

cout << "Вектор B после присваивания" << endl;

cout << B << endl;

cout << "Второй элемент вектора А" << endl;

cout << A[2] << endl;

cout << "Размер А=" << A() << endl;

B = A \* t;

cout << "Вектор B после умножения вектора А на t" << endl;

cout << A\*t << endl;

}

#include <iostream>

using namespace std;

class Time {

public:

Time(void);

Time(int, int);

Time(const Time&);

Time& operator=(const Time&); //перегруженные операции ввода-вывода

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

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

Time operator\*(Time k);

public:

virtual ~Time(void) {};

private:

int min, sec;

};

#include "Time.h"

Time::Time(void)

{

min = sec = 0;

}

Time::Time(int M, int S)

{

min = M;

sec = S;

}

Time::Time(const Time& t)

{

min = t.min;

sec = t.sec;

}

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

{

min = t.min;

sec = t.sec;

return\*this;

}

ostream& operator<<(ostream& out, const Time& t)

{

out << t.min << ":" << t.sec;

return out;

}

istream& operator>>(istream& in, Time& t)

{

cout << "\nВведите минуты: "; in >> t.min;

do {

cout << "\nВведите секунды: "; in >> t.sec;

} while ((t.sec < 0) || (t.sec >= 60));

return in;

}

Time Time::operator\*(Time k)

{

int t = min \* k.min\*60;

int kt = sec \* k.sec;

t += kt;

Time temp(t / 60, t % 60);

return temp;

}

#pragma once

#include <iostream>

using namespace std;

template <class T>

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

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

};

//конструктор с параметрами

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)

{

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)

{

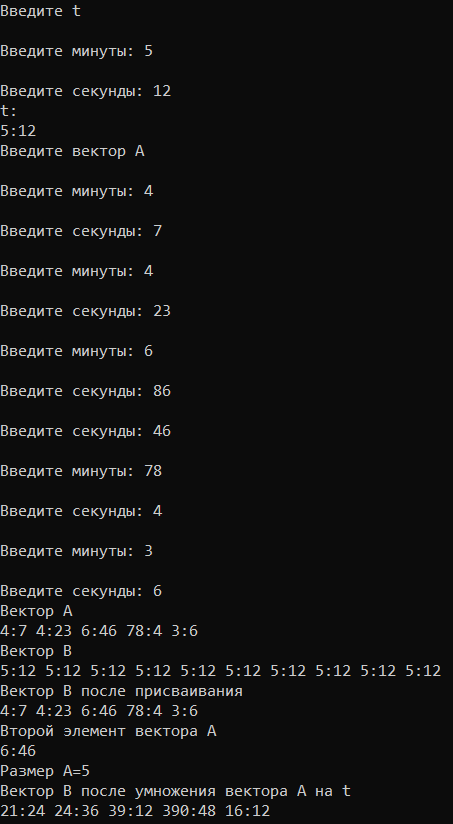
for (int i = 0; i < a.size; ++i) in >> a.data[i];

return in;

}

14

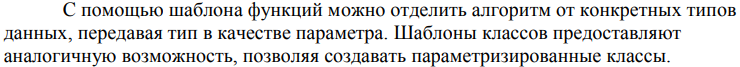
**Скриншоты**



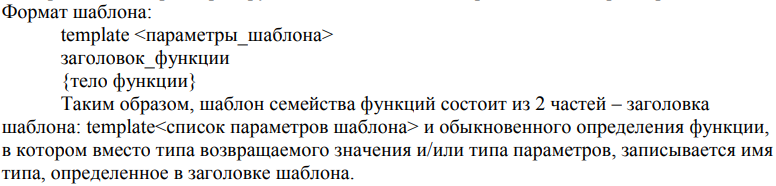
15

**Контрольные вопросы**

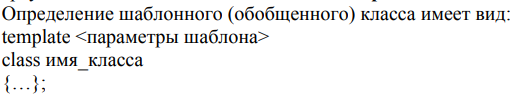


1. 



2. 



3. 



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



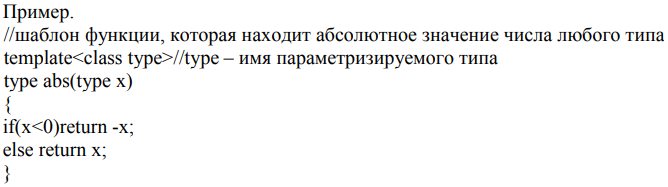
5. Шаблон функции объявляется один раз. Шаблон семейства функции состоит из 2 частей – заголовка шаблона и обыкновенного определения функции.



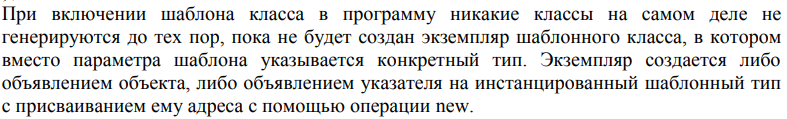
6. Параметр шаблона записывается в определении шаблона: template<typename Type>, далее в описании переменных и методов вместо типа данных необходимо записывать Type.



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