Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 - «Информатика и вычислительная техника»

**Отчет по лабораторной работе “Классы” №7**

**по дисциплине**

**«Теория алгоритмов и структуры данных»**

Выполнил студент гр. ИВТ-21-1б

Ахунов Руслан Булатович

Проверил:

Ст. Преподаватель кафедры ИТАС

Яруллин Денис Владимирович

(оценка) (подпись)

г. Пермь - 2022

**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.

Вариант 2:

Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции:

[]– доступа по индексу;

int() – определение размера вектора;

+ вектор – сложение элементов векторов a[i]+b[i];

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.

**Анализ задачи:**

template <class T>

class Vector

{

private:

    int size;

    T\* data;

public:

    Vector(int s, T tmp)

    {

        size = s;

        data = new T[size];

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

        {

            data[i] = tmp;

        }

    }

    /\*Vector(int s, Vector<T> tmp)

    {

        size = s;

        data = new T[size];

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

        {

            data[i] = tmp.data[i];

        }

    }\*/

    Vector(const Vector<T>& tmp)

    {

        size = tmp.size;

        data = new T[size];

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

        {

            data[i] = tmp.data[i];

        }

    }

    ~Vector()

    {

        delete[]data;

        data = 0;

    }

    Vector& operator=(const Vector<T>& tmp)

    {

        if (this == &tmp)

        {

            return \*this;

        }

        size = tmp.size;

        if (data != 0)

        {

            delete[]data;

        }

        data = new T[size];

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

        {

            data[i] = tmp.data[i];

        }

        return \*this;

    }

    T& operator[](int index)

    {

        if (index < size)

        {

            return data[index];

        }

        else

        {

            cout << "Индекс больше размера вектора" << endl;

            return data[0];

        }

    }

    Vector operator+(const Vector<T> tmp)

    {

        T a(0, 0);

        Vector<T> tmp2(size, a);

        if (size >= tmp.size)

        {

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

            {

                tmp2.data[i] = data[i] + tmp.data[i];

            }

        }

        else

        {

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

            {

                tmp2.data[i] = data[i] + tmp.data[i];

            }

        }

        return tmp2;

    }

    int operator()()

    {

        return size;

    }

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

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

};

template <class T>

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

{

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

    {

        out << tmp.data[i] << " ";

    }

    return out;

}

template <class T>

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

{

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

    {

        in >> tmp.data[i];

    }

    return in;

}

Параметризированный класс-контейнер vector, в котором реализовано конструкторы, деструктор, операции ввода-вывода, операцию присваивания. А также были реализованы: []– доступа по индексу;

int() – определение размера вектора;

+ вектор – сложение элементов векторов a[i]+b[i];

class Time

{

private:

    int min;

    int sec;

public:

    Time()

    {

        min = 0;

        sec = 0;

    }

    Time(int m, int s)

    {

        min = m;

        sec = s;

    }

    Time(const Time& t)

    {

        min = t.min;

        sec = t.sec;

    }

    /\*int getMin()

    {

        return min;

    }

    int getSec()

    {

        return sec;

    }

    void setMin(int m)

    {

        min = m;

    }

    void setSec(int s)

    {

        sec = s;

    }\*/

    Time& operator=(const Time& t)

    {

        if (&t == this) {

            return \*this;

        }

        min = t.min;

        sec = t.sec;

        return \*this;

    }

    Time operator+(Time tmp)

    {

        int tm = min \* 60 + tmp.min \* 60 + sec + tmp.sec;

        Time tmp2(tm / 60, tm % 60);

        return tmp2;

    }

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

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

    ~Time()

    {

    }

};

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

{

    return (out << "(" << tmp.min << " : " << tmp.sec << ")");

}

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

{

    cout << "\nmin?";

    in >> tmp.min;

    cout << "\nsec?";

    in >> tmp.sec;

    return in;

}

Так же был реализован класс Time, в котором были перегружены операторы потокового ввода и вывода.

int main()

{

    setlocale(LC\_ALL, "Russian");

    /\*Vector<int>A(5, 0);

    cin >> A;

    cout << A << endl;

    Vector<int>B(10, 1);

    cout << B << endl;

    B = A;

    cout << B << endl;

    cout << A[2] << endl;

    cout << "Размер А = " << A() << endl;

    A = A + B;

    cout << A << endl;\*/

    Time t(10, 45);

    cout << t;

    Vector<Time>Q(5, t);

    cin >> Q;

    cout << Q << endl;

    Vector<Time>W(10, t);

    cout << W << endl;

    W = Q;

    cout << W << endl;

    cout << Q[2] << endl;

    cout << "Размер Q = " << Q() << endl;

    Q = Q + W;

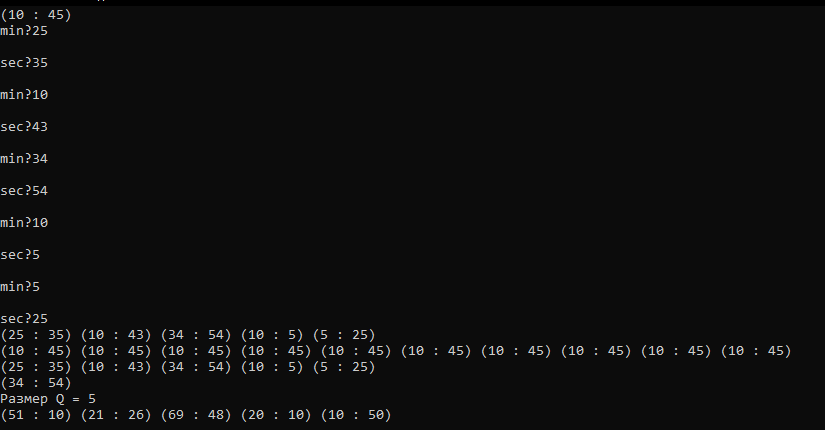
    cout << Q << endl;

    return 0;

}

В функции main, мы выполнили тестирование параметризированного класса Вектор для пользовательского типа данных Time.

**Результат работы программы**

****

Заполнили значения для вектора для пользовательского типа данных Time, выполнили операцию присваивания и выполнили операцию сложения для двух Векторов для пользовательского типа данных Time.

**Код программы**

#include <iostream>

using namespace std;

template <class T>

class Vector

{

private:

    int size;

    T\* data;

public:

    Vector(int s, T tmp)

    {

        size = s;

        data = new T[size];

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

        {

            data[i] = tmp;

        }

    }

    /\*Vector(int s, Vector<T> tmp)

    {

        size = s;

        data = new T[size];

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

        {

            data[i] = tmp.data[i];

        }

    }\*/

    Vector(const Vector<T>& tmp)

    {

        size = tmp.size;

        data = new T[size];

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

        {

            data[i] = tmp.data[i];

        }

    }

    ~Vector()

    {

        delete[]data;

        data = 0;

    }

    Vector& operator=(const Vector<T>& tmp)

    {

        if (this == &tmp)

        {

            return \*this;

        }

        size = tmp.size;

        if (data != 0)

        {

            delete[]data;

        }

        data = new T[size];

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

        {

            data[i] = tmp.data[i];

        }

        return \*this;

    }

    T& operator[](int index)

    {

        if (index < size)

        {

            return data[index];

        }

        else

        {

            cout << "Индекс больше размера вектора" << endl;

            return data[0];

        }

    }

    Vector operator+(const Vector<T> tmp)

    {

        T a(0, 0);

        Vector<T> tmp2(size, a);

        if (size >= tmp.size)

        {

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

            {

                tmp2.data[i] = data[i] + tmp.data[i];

            }

        }

        else

        {

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

            {

                tmp2.data[i] = data[i] + tmp.data[i];

            }

        }

        return tmp2;

    }

    int operator()()

    {

        return size;

    }

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

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

};

template <class T>

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

{

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

    {

        out << tmp.data[i] << " ";

    }

    return out;

}

template <class T>

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

{

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

    {

        in >> tmp.data[i];

    }

    return in;

}

class Time

{

private:

    int min;

    int sec;

public:

    Time()

    {

        min = 0;

        sec = 0;

    }

    Time(int m, int s)

    {

        min = m;

        sec = s;

    }

    Time(const Time& t)

    {

        min = t.min;

        sec = t.sec;

    }

    /\*int getMin()

    {

        return min;

    }

    int getSec()

    {

        return sec;

    }

    void setMin(int m)

    {

        min = m;

    }

    void setSec(int s)

    {

        sec = s;

    }\*/

    Time& operator=(const Time& t)

    {

        if (&t == this) {

            return \*this;

        }

        min = t.min;

        sec = t.sec;

        return \*this;

    }

    Time operator+(Time tmp)

    {

        int tm = min \* 60 + tmp.min \* 60 + sec + tmp.sec;

        Time tmp2(tm / 60, tm % 60);

        return tmp2;

    }

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

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

    ~Time()

    {

    }

};

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

{

    return (out << "(" << tmp.min << " : " << tmp.sec << ")");

}

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

{

    cout << "\nmin?";

    in >> tmp.min;

    cout << "\nsec?";

    in >> tmp.sec;

    return in;

}

int main()

{

    setlocale(LC\_ALL, "Russian");

    /\*Vector<int>A(5, 0);

    cin >> A;

    cout << A << endl;

    Vector<int>B(10, 1);

    cout << B << endl;

    B = A;

    cout << B << endl;

    cout << A[2] << endl;

    cout << "Размер А = " << A() << endl;

    A = A + B;

    cout << A << endl;\*/

    Time t(10, 45);

    cout << t;

    Vector<Time>Q(5, t);

    cin >> Q;

    cout << Q << endl;

    Vector<Time>W(10, t);

    cout << W << endl;

    W = Q;

    cout << W << endl;

    cout << Q[2] << endl;

    cout << "Размер Q = " << Q() << endl;

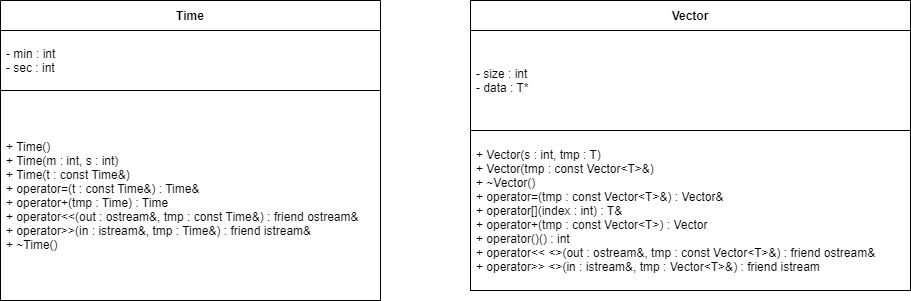
    Q = Q + W;

    cout << Q << endl;

    return 0;

}

**Диаграмма класса**



**Вопросы**

* 1. **В чем смысл использования шаблонов?**

С помощью шаблона функций можно отделить алгоритм от конкретных типов данных, передавая тип в качестве параметра. Шаблоны классов предоставляют аналогичную возможность, позволяя создавать параметризированные классы.

* 1. **Каковы синтаксис/семантика шаблонов функций?**

template <параметры\_шаблона> заголовок\_функции

{тело функции}

* 1. **Каковы синтаксис/семантика шаблонов классов**

template <параметры шаблона> class имя\_класса

{…};

* 1. **Что такое параметры шаблона функции?**

Параметры шаблона являются его формальными аргументами, а типы тех аргументов, которые используются в конкретных обращениях к функции, служат фактическими аргументами шаблона.

Пример:

template <class T> ; Где T – параметр шаблона.

* 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 каждому 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 не параметризованных формальных параметров. Может быть непараметризованно 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?**

<class Параметр шаблона> или <typename Параметр шаблона>

* 1. **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.

* 1. **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;

• Статические переменные шаблонов классов необходимо инициализировать для каждого используемого типа данных.

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

Да

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

Нет

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

Нет

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

Так же как и сам класс, то есть сперва нужно начать с ключевого слова <template…>

* 1. **Что такое инстанцирование шаблона?**

Процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона

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

При включении шаблона класса в программу никакие классы на самом деле не генерируются до тех пор, пока не будет создан экземпляр шаблонного класса, в котором вместо параметра шаблона указывается конкретный тип. Экземпляр создается либо объявлением объекта, либо объявлением указателя на инстанцированный шаблонный тип с присваиванием ему адреса с помощью операции new.

Встретив такие объявления, компилятор генерирует код исходного класса.

В проекте, состоящем из нескольких файлов, определение шаблона класса обычно выносится в отдельный файл. Но для того, чтобы инстанцировался конкретный экземпляр шаблона класса необходимо, чтобы определение шаблона находилось в одной единице трансляции с этим экземпляром. Поэтому все определение шаблонного класса размещается в заголовочном файле, а затем этот файл подключается к нужным файлам с помощью директивы include. Чтобы этот файл не включался повторно, используется директива ifndef.