**ИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**«Сибирский государственный университет науки и технологий   
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Языки программирования

|  |
| --- |
| Шаблонные классы |

Руководитель А.В. Проскурин

подпись, дата инициалы, фамилия

Обучающийся БПИ22-02, 221219040 К.В. Трифонов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023 г.

# ЦЕЛЬ РАБОТЫ

Получение практических навыков разработки и отладки программ, c использованием шаблонов функций и классов.

# порядок выполнения работы

1. Ознакомиться с общей постановкой задачи.

2. Ознакомится с вариантом задания – соответствует вашему номеру в списке группы (при нехватке заданий вариант задания вычисляется как номер\_в\_списке\_группы - количество\_заданий).

3. Разработать классы согласно варианту задания.

4. Написать и отладить программу на подготовленных наборах тестовых данных.

5. Подготовить отчет по лабораторной работе. Отчет должен включать в себя:

• титульный лист;

• цель лабораторной работы;

• постановку задачи;

• текст программы с комментариями;

• демонстрацию работы программы (Снимки экрана при выполнении действий программы с описанием).

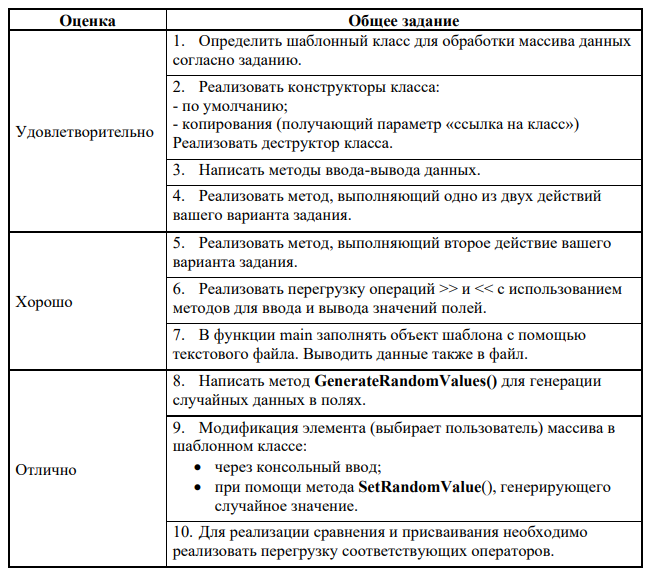
• краткие ответы на контрольные вопросы;

• выводы по лабораторной работе.

6. Защитить лабораторную работу перед преподавателем.

# постановка задачи

Необходимо разработать программу, в которой будет реализован шаблонный класс, выполняющий обработку массива данных согласно варианту задания. В зависимости от оценки, на которую вы претендуете, необходимо выполнить следующие задания (Для каждой следующей оценки нужно выполнить ВСЕ предыдущие задания, если обратное не указано явно):



**Вариант №21.**

Напишите программу, в которой создайте шаблонный класс **TMatrix**, реализующий динамический двумерный массив и функции-методы:

▪ определения количества элементов массива, которые меньше среднего арифметического значения всех элементов четных строк;

▪ нахождения суммы положительных элементов массива (для комплексного числа проверяется действительная часть).

Протестируйте вашу программу для работы с массивами различных типов: **double** и класса **комплексное число** (представлено двумя компонентами). Этот класс необходимо создать.

# ХОД РАБОТЫ

**Описание класса TMatrix**

*template* <*class* Type>

*class* TMatrix{

*int* a,b;

        Type\*\* matrix;

Поля a b для i x j матрицы, указатель на матрицу matrix.

*public:*

*//Конструктор по умолчанию*

        TMatrix() : a(0), b(0), matrix(nullptr)

        {}

*//Конструктор с параметрами*

        TMatrix(*int* *I*, *int* *J*) : a(*I*), b(*J*), matrix(new Type\*[a]){

            for(*int* i = 0; i<a; i++){

                matrix[i] = new Type [b];

            }

        }

*//Конструктор копирования*

        TMatrix(*const* TMatrix*&* *B*) {

            Clear();

            for(*int* i = 0; i<a; i++){

                for (*int* j = 0; j<b; j++){

                    matrix[i][j] = *B*.matrix[i][j];

                }

            }

        }

Выделение динамической памяти в конструкторах

*//Деструктор*

        ~TMatrix(){

            Clear();

        }

*void* Clear(){

            for (*int* i = 0; i<a; i++){

                delete[] matrix[i];

            }

            delete[] matrix;

            matrix=nullptr;

            a = b = 0;

        }

Метод Clear() для очистки динамической памяти и обнуления полей

*//Операторы*

*//Индексирование*

        Type*&* operator ()(*int* *A*,*int* *B*) *const*{

            return matrix[*A*][*B*];

        }

*//присваивание*

*void* operator = (TMatrix*&* *B*){

            Clear();

            a = *B*.a;

            b = *B*.b;

            matrix = new Type \* [a];

            for(*int* i = 0; i<a; i++){

                matrix[i] = new Type [b];

            }

            for(*int* i = 0; i<a; i++){

                for (*int* j = 0; j<b; j++){

                    matrix[i][j] = *B*.matrix[i][j];

                }

            }

        }

*//сравнение*

*bool* operator<(*const* TMatrix*&* *B*){

            if (*this*->Psum()<*B*.Psum()){

                return 1;

            }

            else{

                return 0;

            }

        }

*bool* operator>(*const* TMatrix*&* *B*){

            if (*this*->Psum()>*B*.Psum()){

                return 1;

            }

            else{

                return 0;

            }

        }

Сравнения происходит через существующий метод Psum возвращающий сумму положительных элементов матрицы.

*//сетеры*

*void* GenerateRandomValues(*int* *min*, *int* *max*){

            if (a==0||b==0){

                cout<<"Введите размерность \_ x \_\n";

                cin>>a>>b;

                matrix = new Type \* [a];

                for(*int* i = 0; i<a; i++){

                    matrix[i] = new Type [b];

                }

            }

            srand(time(NULL));

            for(*int* i = 0; i<a; i++){

                for (*int* j = 0; j<b; j++){

                    matrix[i][j] = *min* + rand()%(*max*-*min*+1);

                }

            }

        }

*void* SetRandomValue(*int* *I*, *int* *J*,*int* *min*, *int* *max*){

            matrix[*I*][*J*] = *min* + rand()%(*max*-*min*+1);

        }

        istream*&* set(istream*&* *in*){

            Clear();

            if (&*in* == &cin){

                cout<<"Введите размерность \_ x \_\n";

            }

*in*>>a>>b;

            matrix = new Type \* [a];

            for(*int* i = 0; i<a; i++){

                matrix[i] = new Type [b];

            }

            if (&*in* == &cin){

                cout<<"Введите элементы через пробел:\n";

            }

            for(*int* i = 0; i<a; i++){

                for (*int* j = 0; j<b; j++){

*in*>>matrix[i][j];

                }

            }

            return *in*;

        }

В методе set() очистка занятой прежними данными памяти через метод Clear()

*//гетеры*

        ostream*&* get(ostream*&* *out*){

            if (&*out* == &cout){

                cout<<"Матрица размером "<<a<<" x "<<b<<":\n";

            }

            else{

*out*<<a<<" "<<b<<"\n";

            }

            for(*int* i = 0; i<a; i++){

                for (*int* j = 0; j<b; j++){

*out*<<matrix[i][j]<<" ";

                }

*out*<<"\n";

            }

            return *out*;

        }

*//Модификация элемета*

*void* modify(*int* *I*,*int* *J*){

*int* sw;

            cout<<"1) Модифицировать "<<*I*<<" x "<<*J*<<" случайно\n";

            cout<<"2) Модифицировать "<<*I*<<" x "<<*J*<<" через консоль\n";

            cin>>sw;

            switch(sw){

                case 1:{

*int* min,max;

                    cout<<"Введите диапазон от \_ до \_:\n";

                    cin>>min>>max;

                    SetRandomValue(*I*,*J*,min,max);

                    break;

                }

                case 2:{

                    Type v;

                    cin>>v;

                    matrix[*I*][*J*] = v;

                }

            }

        }

В метод передаётся адрес ячейки и пользователю предлагается выбрать изменение элемента случайно или ввести его через консоль, для задания случайного элемента используется метод SetRandomValue().

*int* LowAwg(){

*float* sum = 0;

*int* c = 0;

            for (*int* i = 0; i<=a/2; i++){

                for (*int* j = 0; j<b; j++){

                    sum+=matrix[i\*2][j];

                    c++;

                }

            }

*float* avg = sum/c;

*int* c2 = 0;

            for(*int* i = 0; i<a; i++){

                for (*int* j = 0; j<b; j++){

                    if (matrix[i][j]<avg){

                        c2++;

                    }

                }

            }

            return c2;

        }

Вычисляет количество элементов матрицы, которые меньше среднего каждой четной строки.

        Type Psum(){

            Type sum=0;

            for(*int* i = 0; i<a; i++){

                for (*int* j = 0; j<b; j++){

                    if (matrix[i][j]>0){

                        sum+=matrix[i][j];

                    }

                }

            }

            return sum;

        }

Вычисляет сумму всех положительных элементов

*//Работа с файлами*

*void* f\_in(string *filename*){

        Clear();

        ifstream fin(*filename*);

        set(fin);

        cout<<"Запись завершена.\n";

    }

*void* f\_out(string *filename*){

        ofstream fout(*filename*);

        get(fout);

        cout<<"Запись завершена.\n";

    }

};

*template*<*class* Type>

ostream*&* operator <<(ostream*&* *out*, TMatrix<Type>*&* *Matr*){

*Matr*.get(*out*);

    return *out*;

}

*template*<*class* Type>

istream*&* operator >>(istream*&* *in*, TMatrix<Type>*&* *Matr*){

*Matr*.set(*in*);

    return *in*;

}

По заданию тестирование требовалось проводить на типах double и complex. Был создан пользовательский класс complex с полями real и img для действительной и мнимой частей числа.

Описание класса **complex**:

*class* complex

{

*private:*

*double* real, img;

*public:*

    complex() { real = 0; img = 0; }

    complex(*double* *r*, *double* *i*=0.0) { real = *r*; img = *i*; }

    complex(*const* complex *&ob*){ real = *ob*.real; img = *ob*.img; };

Конструкторы по умолчанию, с параметрами и копирования

    complex*&* operator = (complex);

    complex*&* operator = (*const* *float* *I*);

    complex operator + (complex);

    complex operator - (complex);

    complex*&* operator += (complex);

*friend* *float* operator+= (*float* *&I*, complex*&* *comp*);

    complex*&* operator -= (complex);

Арифметические операторы, дружественный оператор += принимающий левостоящий float.

*bool* operator == (complex*&* *com*);

*bool* operator != (complex*&* *com*);

*bool* operator > (*const* complex*&* *com*);

*bool* operator < (*const* complex*&* *com*);

*bool* operator < (*const* *float* *I*);

*bool* operator > (*const* *float* *I*);

Логические операторы с перегрузкой для float (используется для метода LowAvg())

*friend* istream*&* operator>>(std::istream*&*, complex*&*);

*friend* ostream*&* operator<< (std::ostream*&*, *const* complex*&*);

};

**Описание операторов класса complex:**

*bool* complex::operator != (complex*&* *comp*) {

    if(*this*->real != *comp*.real || *this*->img != *comp*.img)

       return 1;

    else

        return 0;

}

*bool* complex::operator==(complex*&* *comp*){

    if(*this*->real == *comp*.real && *this*->img == *comp*.img)

        return 1;

    else

        return 0;

}

complex*&* complex::operator =(complex *comp*){

*this*->real = *comp*.real;

*this*->img = *comp*.img;

    return \**this*;

}

complex*&* complex::operator = (*const* *float* *I*){

    real = *I*;

    return \**this*;

}

 Если оператор = принимает значение float, то он приравнивает его только к действительной части (Представление действительного числа как комплексное).

complex complex::operator+(complex *comp*){

*this*->real = *this*->real + *comp*.real;

*this*->img = *this*->img + *comp*.img;

    return \**this*;

}

complex complex::operator-(complex *comp*){

*this*->real = *this*->real - *comp*.real;

*this*->img = *this*->img - *comp*.img;

    return \**this*;

}

complex*&* complex::operator+=(complex *comp*){

    real += *comp*.real;

    img += *comp*.img;

    return \**this*;

}

*//перегрузка дружественного +=*

*float* operator+= (*float* *&I*, complex*&* *comp*){

*I* = *I* + *comp*.real;

    return *I*;

}

 Если оператор += принимает значение float, то он прибавляет его только к действительной части.

complex*&* complex::operator-=(complex *comp*){

    real -= *comp*.real;

    img -= *comp*.img;

    return \**this*;

}

*bool* complex:: operator > (*const* complex*&* *com*)

{

    if(*this*->real > *com*.real)

        return 1;

    else if(*this*->real == *com*.real && *this*->img > *com*.img)

        return 1;

    else

        return 0;

}

*bool* complex ::operator < (*const* complex*&* *com*)

{

    if(*this*->real < *com*.real)

        return 1;

    else if(*this*->real == *com*.real && *this*->img < *com*.img)

        return 1;

    else

        return 0;

}

*bool* complex::operator < (*const* *float* *I*)

{

    if(real < *I*)

        return 1;

    else

        return 0;

}

*bool* complex::operator > (*const* *float* *I*)

{

    if(real > *I*)

        return 1;

    else

        return 0;

}

ostream*&* operator << (std::ostream*&* *out*, *const* complex*&* *com*)

{

    if (&*out* == &cout){

        if(*com*.img < 0)

*out* << *com*.real << " + i(" << *com*.img << ")";

        else

*out* << *com*.real << " + i" << *com*.img ;

    }

    else{

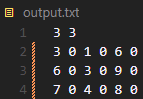
*out* << *com*.real <<" " << *com*.img;

    }

    return *out*;

}

 Если поток cout, то данные выводятся с форматированием, если поток другой (пр. файловый), то данные выводятся в сыром виде без форматирования.



*Матрица комплексных чисел 3x3 с мнимой частью равной нулю.*

istream*&* operator >> (std::istream*&* *in*, complex*&* *com*)

{

*in* >> *com*.real;

*in* >> *com*.img;

    return *in*;

}

# Тестирование

Для тестирования программы было создано меню:

    cout<<"Выберите тип тестируемых данных:\n"

    <<"1) double\n"

    <<"2) complex\n";

    cin>>sw;

*int* a,b;

    cout<<"Введите размерность \_ x \_\n";

    cin>>a>>b;

    if (sw == 1){

        TMatrix<*double*> A(a,b);

*int* f = 1;

        while(f){

        cout<<"1) Заполнить матрицу с консоли\n"

        <<"2) Заполнить матрицу случайно\n"

        <<"3) Вывести матрицу\n"

        <<"4) Модифицировать элемент\n"

        <<"5) Кол-во чисел меньше среднего четных строк - minAvg()\n"

        <<"6) Сумма положительных элементов массива - Psum()\n"

        <<"7) Вывести в файл\n"

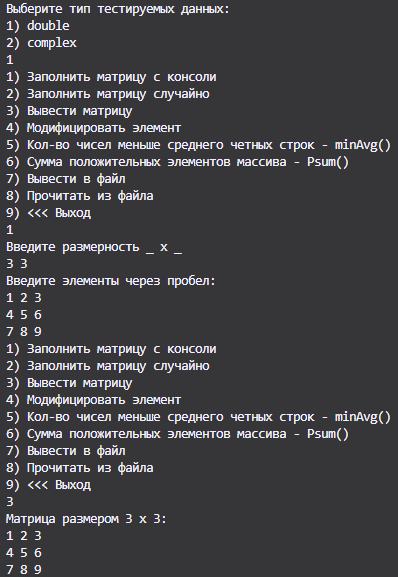
        <<"8) Прочитать из файла\n"

        <<"9) <<< Выход\n";

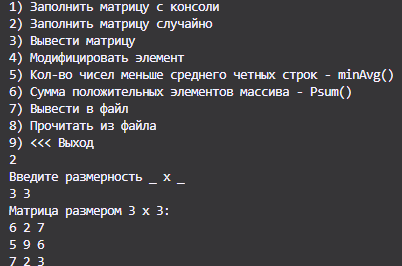
            cin>>sw2;

            switch (sw2){

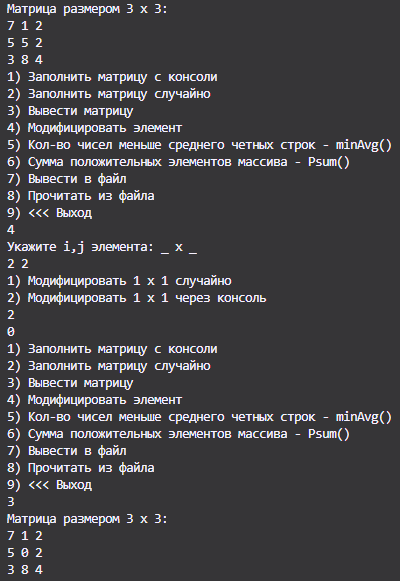
Пример заполнения и вывода матрицы размером 3 x 3:



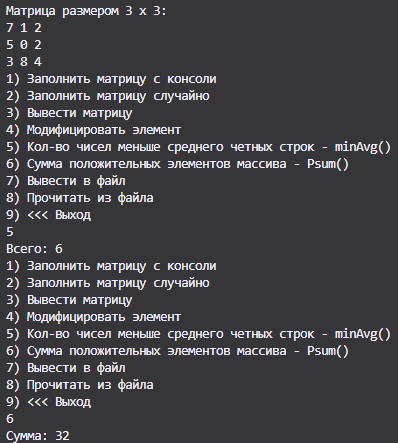
Пример случайного заполнения матрицы размером 3 x 3:



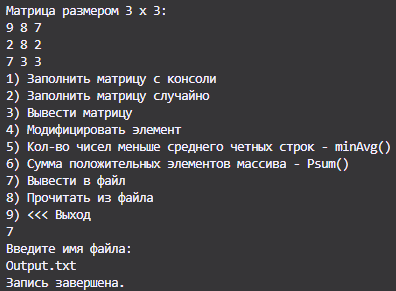
Модификация элемента i, j = 2 через консоль:

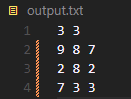


Подсчет кол-ва чисел меньше среднего четных строк и суммы положительных элементов массива:

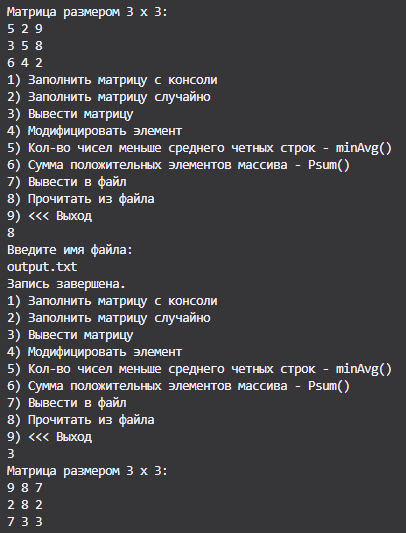


Вывод в файл матрицы размером 3 x 3:

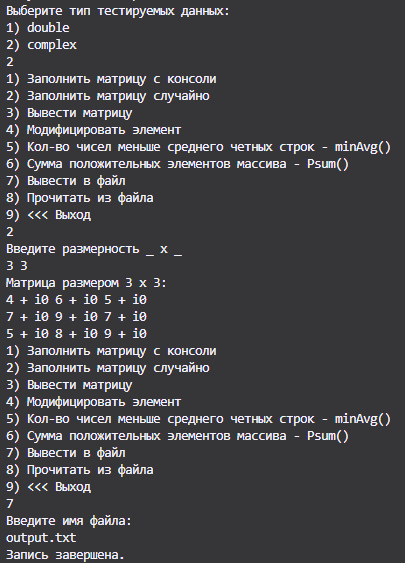


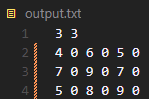


Чтение из файла output.txt:



Работа класса с типом complex (генерация случайных чисел и вывод в файл output.txt):





# ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. **Какой принцип ООП реализуется с помощью шаблонных функций и классов?**

Принцип полиморфизма - способность функции обрабатывать данные разных типов.

1. **Когда целесообразно использовать шаблоны функций?**

Целесообразно использовать шаблоны функций, когда необходимо написать универсальный код для работы с различными типами данных без дублирования.

1. **Каков общий вид объявления шаблона функции?**

*template* <*class* Type>

1. **Каков синтаксис объявления шаблонного класса?**

*template* <*class* Type>

*class* TMatrix{...}

1. **В чем особенности определения шаблонного класса?**

Определение класса и его методов обычно размещается в том же файле, что и объявление.

1. **В чем особенности определения метода шаблонного класса вне класса?**

Методы шаблонного класса могут быть определены вне класса с использованием того же синтаксиса шаблона.

1. **Что такое специализация шаблона?**

Специализация шаблона позволяет предоставить специфическую реализацию для конкретного типа данных.

1. **Как объявить объект шаблонного класса?**

TMatrix<complex> A(a,b);

1. **Что такое инстанцирование?**

Инстанцирование происходит, когда создается экземпляр шаблонного класса или вызывается шаблонная функция с определенным типом данных.

1. **Чем в строке template <class T> может быть T?**

d) T может быть любым типом – как встроенным в язык C++, так и определенным программистом.

# ВЫВОД

Было изучено то, как использовать шаблоны в C++, которые позволяют делать универсальные куски кода для работы с разными типами данных. С их помощью можно писать программы, которые гибко обрабатывают разные данные, что упрощает жизнь и делает код более переиспользуемым. Работа с шаблонами в С++ — это один из способов делать программы более гибкими и удобными.