Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Отчёт к лабораторной работе №10

по дисциплине

«Языки программирования»

Работу выполнила

Студент группы СКБ222 _____ М. Х. Халимов
подпись, дата

Работу проверил _____ С. А. Булгаков
подпись, дата

Москва 2023

Содержание

Постановка задачи

На основе класса *Matrix* из домашней работы №4 разработать шаблон класса *Matrix*, параметр шаблона – тип данных, используемый для хранения значений. Интерфейс и реализацию разместить в файле *matrix.hpp*.

Класс должен позволять выполнять основные арифметические операции вида *m@m*, где *Matrix<T> m*, умножение/деление на целое/вещественное число, а также операции помещения (извлечения) в поток (из потока).

Реализовать методы:

* LU - выполняющий LU-разложение
  (прямой ход метода Гаусса с частичным выбором ведущего элемента);
* M, L, U - для доступа к матрицам перестановок, нижнетреугольной и верхнетреугольной соответственно;
* Solve - выплняющий решение СЛАУ на основе LU-разложения;
* operator() - для доступа к элементам матрицы.

1. Описание шаблона класса *matrix*

Объявляется класс matrix, являющийся совокупностью полей *rows, cols* типа *size_t* и **data* любого типа и методов *matrix(const &matrix), matrix(rows, cols), operator=, operator+, operator-, operator*, operator>>, operator<<, at, M, L, U, LU, Solve.* Метод отвечает за создание матрицы.

2. Описание методов класса

Методы класса реализованы также, как и в домашней работе №4, за исключением смены типа данных с unsigned long на *typename Type*, объявленный в шаблоне класса. Также добавлены методы *M, L, U, LU, Solve* в совокупности отвечающие за декомпозицию матрицы.

3. Функция main

Функция *main* включает в себя создание объектов класса *main* и вызов методов этого же класса. В результате работы функции в консоль выводится сумма, разность, произведение, частное, получаемые из заданных чисел.

4. Результаты тестирования программы

Листинг

```
2
2
2
3
3
0
We got matrix
M:
0 1
1 0

L:
1 0
0.666667 1

U:
3 0
0 3

M*A:
3 0
2 3
```

```
M*L*U:
2 3
3 0

A:
2 3
3 0

b:
1
4

Solving Ax = b...
x:
1.33333
0.333333
```

Приложение А

Исходный код

```
#include <cstddef>
#include <cstring>
#include <iostream>
#include <cstdlib>

template<typename Type>
class Matrix {
    size_t rows;
    size_t cols;
    Type *data;
  public:
    Matrix()
    {
        this->rows = 0;
        this->cols = 0;
        this->data = NULL;
    }
    Matrix(const Matrix<Type>& matrix)
    {
        this->rows = matrix.rows;
        this->cols = matrix.cols;
        this->data = (Type*)new Type[this->rows*this->cols];
        memcpy(this->data,       matrix.data,       matrix.rows       *
matrix.cols*sizeof(Type));
    }
    Matrix(size_t rows, size_t cols)
    {
```

```
        this->rows = rows;
        this->cols = cols;
        this->data = (Type*)new Type[rows*cols];
        for(size_t i = 0; i < rows; i++)
        {
            for(size_t j = 0; j < cols; j++)
                data[i*cols+j] = 0;
        }
    }
   ~Matrix()
    {
        this->rows = 0;
        this->cols = 0;
        delete[] this->data;
    }

    Matrix<Type>& operator=(const Matrix<Type>& matrix)
    {
        if (this == &matrix) {
            return *this;
        }
        this->rows = matrix.rows;
        this->cols = matrix.cols;
        delete[] this->data;
        this->data = new Type[matrix.cols*matrix.rows];
        memcpy(this->data,       matrix.data,       matrix.rows       *
matrix.cols*sizeof(Type));
        return *this;
    }
    const Matrix<Type> operator+(const Matrix<Type>& right)
    {
        if(this->rows != right.rows)
            throw std::logic_error("Count of rows are not the same");
        if (this->cols != right.cols)
            throw std::logic_error("Count of cols are not the same");

        Matrix<Type> task(right.rows, right.cols);
        for(size_t i = 0; i < right.rows; i++)
        {
            for(size_t j = 0; j < right.cols; j++)
                task.data[i*this->cols+j] = this->data[i*right.cols+j]
+ right.data[i*right.cols+j];
        }
        return task;
    }
    const Matrix<Type> operator-(const Matrix<Type>& right)
    {
        if(this->rows != right.rows)
            throw std::logic_error("Count of rows are not the same");
        if (this->cols != right.cols)
            throw std::logic_error("Count of cols are not the same");
```

```
        Matrix<Type> task(right.rows, right.cols);
        for(size_t i = 0; i < right.rows; i++)
        {
            for(size_t j = 0; j < right.cols; j++)
                task.data[i*this->cols+j] = this->data[i*right.cols+j]
- right.data[i*right.cols+j];
        }
        return task;
    }
    const Matrix<Type> operator*(const Matrix<Type>& right)
    {
        if(this->cols != right.rows)
            throw std::logic_error("Count rows of right and count cols
of left are not the same");

        Matrix<Type> task(this->rows, right.cols);
        for(size_t i = 0; i < task.rows; i++)
        {
            for(size_t j = 0; j < task.cols; j++)
            {
                Type sum = 0;
                for(size_t k = 0; k < task.rows; k++)
                    sum += this->data[i*this->cols+k] * right(k,j);
                task.data[i*task.cols+j] = sum;
            }
        }
        return task;
    }
    const Matrix<Type> operator*(const double& coefficient)
    {
        Matrix<Type> task(this->rows, this->cols);
        for(size_t i = 0; i < task.rows; i++)
        {
            for(size_t j = 0; j < task.cols; j++)
                task.data[i*task.cols+j]            =            this-
>data[i*task.cols+j]*coefficient;
        }
        return task;
    }
    friend const Matrix<Type> operator*(const double& coefficient,
Matrix<Type>& matrix)
    {
        Matrix<Type> task(matrix.rows, matrix.cols);
        for(size_t i = 0; i < task.rows; i++)
        {
            for(size_t j = 0; j < task.cols; j++)
                task.data[i*task.cols+j]                                =
matrix.data[i*task.cols+j]*coefficient;
        }
        return task;
    }
```

```
    friend std::istream& operator>>(std::istream & in, Matrix<Type>&
matrix)
    {
        in >> matrix.rows >> matrix.cols;
        delete[] matrix.data;
        matrix.data = new Type[matrix.rows*matrix.cols];
        for(size_t i = 0; i < matrix.rows; i++)
        {
            for(size_t j = 0; j < matrix.cols; j++)
                in >> matrix.data[i*matrix.cols+j];
        }
        return in;
    }
    friend   std::ostream&   operator<<(std::ostream   &   out,const
Matrix<Type>& matrix)
    {
        for(size_t i = 0; i < matrix.rows; i++)
        {
            for(size_t j = 0; j < matrix.cols; j++)
                out << +matrix.data[i*matrix.cols+j] << " ";
            out << std::endl;
        }
        return out;
    }

    Type& at(size_t right, size_t c)
    {
        return data[right*this->cols+c];
    }
    const Type& at(size_t right, size_t c) const
    {
        return data[right*this->cols+c];
    }

    Type& operator()(size_t right, size_t c)
    {
        return data[right*this->cols+c];
    }
    const Type& operator()(size_t right, size_t c) const
    {
        return data[right*this->cols+c];
    }


    Matrix<Type>& M() const
    {
        Matrix<Type> *M = new Matrix<Type>(*this);
        Matrix<Type> L(*this);
        Matrix<Type> U(*this);
        this->LU(&U,&L,M);
        return *M;
    }
```

```
    Matrix<Type>& L() const
    {
        Matrix<Type> M (*this);
        Matrix<Type> *L = new Matrix<Type>(*this);
        Matrix<Type> U(*this);
        this->LU(&U,L,&M);
        return *L;
    }
    Matrix<Type>& U() const
    {
        Matrix<Type> M(*this);
        Matrix<Type> L(*this);
        Matrix<Type> *U = new Matrix<Type>(*this);
        this->LU(U,&L,&M);
        return *U;
    }

    void LU(Matrix<Type> *u, Matrix<Type> *left, Matrix<Type> *matrix)
const
    {
        if(this->cols != this->rows
         || left->rows!=this->rows && left->cols != this->cols
         || u->rows!=this->rows && u->cols != this->cols
         || matrix->rows!=this->rows && matrix->cols != this->cols)
            throw std::logic_error("Count rows of right and count cols
of left are not the same");
        Matrix<Type> res_matrix(*this);

        for(int i = 0; i < matrix->rows; ++i)
        {
            for(int j = 0; j < matrix->cols; j++)
            {
                (*matrix)(i, j) = (j == i);
                (*left)(i, j) = (j == i);
                (*u)(i, j) = 0;
            }
        }

        for (size_t i = 0; i < this->rows; i++) {
            size_t max_row = i;
            for (size_t j = i+1; j < this->cols; j++) {
                    if(res_matrix.data[j*this->cols+i]                >
res_matrix.data[max_row*this->cols+i])
                        max_row = j;
            }
            if(max_row != i)
            {
                Type* temporary = (Type*)new Type[this->cols];
                memcpy(temporary,&res_matrix.data[max_row*this->cols],
this->cols*sizeof(Type));
                memcpy(&res_matrix.data[max_row*this->cols],
&res_matrix.data[i*this->cols], this->cols*sizeof(Type));
```

```
                memcpy(&res_matrix.data[i*this->cols],       temporary,
this->cols*sizeof(Type));

                memcpy(temporary,&matrix->data[max_row*matrix->cols],
matrix->cols*sizeof(Type));
                memcpy(&matrix->data[max_row*matrix->cols],    &matrix-
>data[i*matrix->cols], matrix->cols*sizeof(Type));
                memcpy(&matrix->data[i*matrix->cols],        temporary,
matrix->cols*sizeof(Type));
                delete[] temporary;
            }
        }
        for (size_t i = 0; i < this->rows; i++) {
            for (size_t j = 0; j < this->cols; j++) {

                if(i <= j) {
                    Type sum = 0;
                    for(size_t k = 0; k < i; ++k)
                        sum += (*left)(i,k)*(*u)(k,j);
                    (*u)(i,j)   =   (res_matrix.data[i*this->cols+j]   -
sum);
                }
                else if (i > j)
                {
                    Type sum = 0;
                    for(size_t k = 0; k < j; ++k)
                        sum += (*left)(i,k)*(*u)(k,j);
                    (*left)(i,j)  =  (res_matrix.data[i*this->cols+j]  -
sum)/(*u)(j,j);

                }
            }
        }

    }

    Matrix<Type>& Solve(const Matrix<Type>& b) const
    {

        Matrix<Type> P(this->M());
        Matrix<Type> L(this->L());
        Matrix<Type> U(this->U());

        Matrix<Type> pb(P*b);

        Matrix<Type> y(pb.rows, 1);
        for(int i = 0; i < y.rows; ++i)
        {
            Type temporary = pb(i, 0);
            for(int j = 0; j < (i-1);++j)
                temporary -= L(i,j)*y(j,0);
            y(i,0) = temporary / L(i,i);
```

```
        }

        Matrix<Type> *x = new Matrix<Type>(y.rows, 1);
        for(int i = x->rows-1; i >= 0; --i)
        {
            Type temporary = pb(i, 0);
            for(int j = i+1; j < x->rows;++j)
                temporary -= U(i,j)*(*x)(j,0);
            (*x)(i,0) = temporary / U(i,i);
        }

        return *x;
    }
};
```

Приложение Б