**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИЕТ**

Лабораторная работа №7 по дисциплине «Программирование»

**Универсальность. Применение шаблонов функций и классов**

Группа: **АВТ-342**

Студенты: **Бондаренко А.В., Фадеев В.А.**

НОВОСИБИРСК 2024

1. **Постановка задачи**

**Вариант 4.**

Задание представляет собой типовую задачу по разработке шаблонов стандартных структур данных. В качестве структур данных взять разработанные классы в лабораторной работе №5.

Протестировать структуру данных. В качестве хранимых объектов использовать встроенные типы С++ (int, float) и класс лаб. раб. №1.

1. **Определение шаблона класса с комментариями**

// Шаблонный класс узла списка  
template<typename T>  
class Node {  
public:  
 T data; // Данные узла  
 Node\* next; // Указатель на следующий узел  
 Node\* prev; // Указатель на предыдущий узел  
  
 // Конструктор узла  
 Node(const T& newData = T()) : data(newData), next(this), prev(this) {}  
};

template<typename T>  
class List {  
private:  
 Node<T>\* head; // Указатель на головной элемент списка  
 int size; // Количество элементов в списке  
  
public:  
 // Конструктор  
 List() : head(nullptr), size(0) {}  
  
 // Деструктор  
 ~List() {  
 deleteList();  
 }  
  
 // Проверка, пустой ли список  
 bool isEmpty() const {  
 return head == nullptr;  
 }  
  
 // Добавление нового узла в конец списка  
 void append(List<T>& list, const T& data)

// Удаление всего списка  
 void deleteList()

// Отображение элементов списка  
 void display(const List<T>& list) const

// Поиск узла по индексу  
 Node<T>\* findNode(const List<T>& list, int index) const

// Вставка узла в произвольное место списка  
 void insertNode(List<T>& list, int index, const T& newValue)

// Удаление узла по индексу  
 void deleteNode(List<T>& list, int index)

**3. Определение пользовательских классов с комментариями**

Вот список пользовательских классов, определенных в проекте, с краткими комментариями о каждом:

**1. Matrix**

* **Описание:** Класс для работы с квадратными матрицами.
* **Основные методы:**
  + Matrix(): Конструктор по умолчанию.
  + Matrix(int size): Конструктор с заданным размером матрицы.
  + Matrix(const Matrix&): Конструктор копирования.
  + ~Matrix(): Деструктор, освобождает память.
  + input(): Ввод элементов матрицы с клавиатуры.
  + output(): Вывод матрицы на экран.
  + add(const Matrix&), subtract(const Matrix&), multiply(const Matrix&): Арифметические операции над матрицами.
  + transpose(): Транспонирование матрицы.
  + determinant(): Вычисление детерминанта матрицы.
  + Перегрузка операторов =, >>, <<: Удобная работа с копированием и вводом/выводом.

**2. Node**

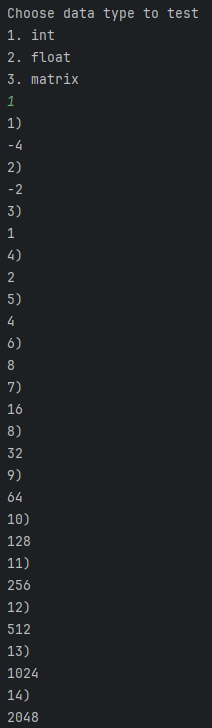
* **Описание:** Шаблонный класс, представляющий узел двусвязного кольцевого списка.
* **Основные члены:**
  + T data: Данные, хранимые в узле.
  + Node\* next: Указатель на следующий узел.
  + Node\* prev: Указатель на предыдущий узел.
  + **Конструктор:** Инициализирует данные узла и автоматически замыкает его на самого себя.

**3. List**

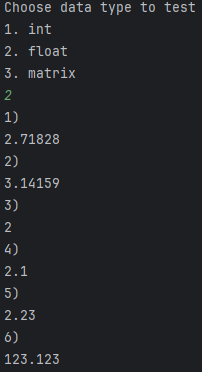
* **Описание:** Шаблонный класс для работы с двусвязным кольцевым списком.
* **Основные методы:**
  + append(List<T>& list, const T& data): Добавление элемента в конец списка.
  + deleteList(): Удаление всех элементов списка.
  + display(const List<T>& list): Вывод списка.
  + isEmpty() const: Проверка, пуст ли список.
  + findNode(const List<T>& list, int index): Поиск узла по индексу.
  + insertNode(List<T>& list, int index, const T& newValue): Вставка элемента на произвольную позицию.
  + deleteNode(List<T>& list, int index): Удаление элемента по индексу.
* **Особенности:** Использует Node<T> для хранения данных. Реализует кольцевую структуру, где последний элемент списка ссылается на первый, и наоборот.

**3. Тестовые примеры выполнения программы**

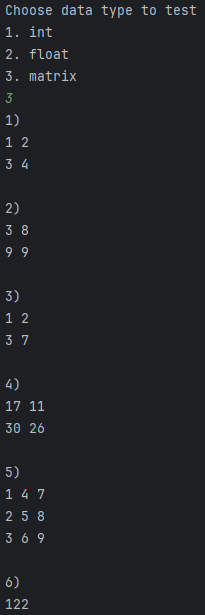
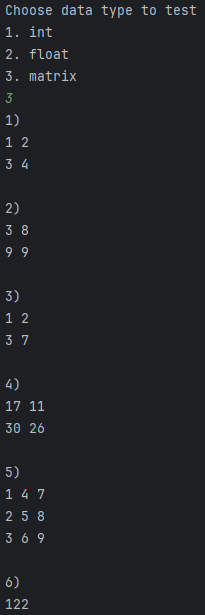
Проверка добавления в список элементов типа int:

Примеры добавления в список элементов типа float:



Примеры добавления в список элементов типа matrix, в том числе и проверка методов класса matrix:

Где узлы:

1 – результат добавления матрицы в список,

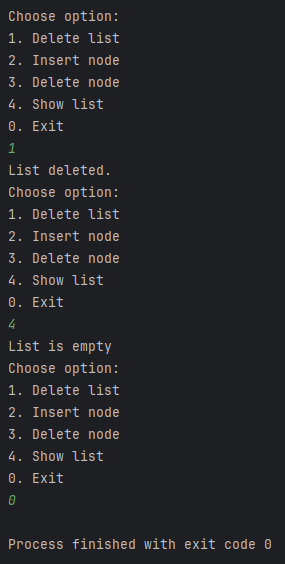
2 – результат сложения матриц,

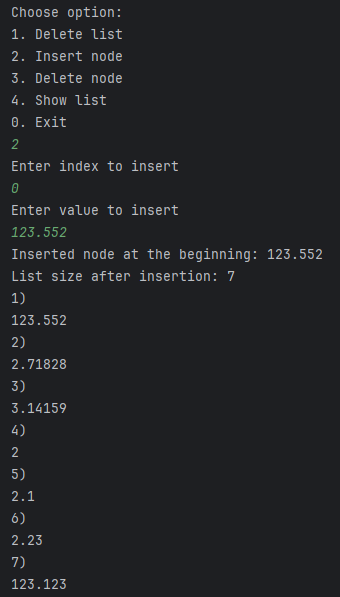
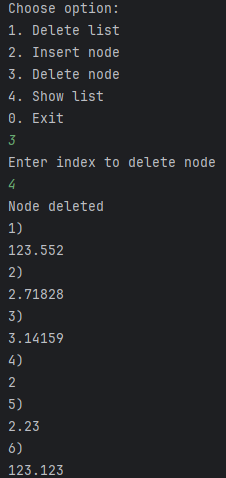
3 – результат вычитания матриц,

4 – результат произведения матриц,

5 – результат транспонирования матрицы,

6 – добавление определителя матрицы в список.

Прочие тесты:

****

**4. Листинг демонстрационной программы**

**main.cpp**

#include <iostream>  
#include "matrix.h"  
using namespace std;  
  
int main() {  
 cout << "Choose data type to test" << endl;  
 cout << "1. int" << endl;  
 cout << "2. float" << endl;  
 cout << "3. matrix" << endl;  
 int type\_choice;  
 cin >> type\_choice;  
  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
  
 List<int> intList;  
 List<float> floatList;  
 List<Matrix> matrixList;  
  
 int type\_choice\_flag;  
  
 while (1) {  
 switch (type\_choice) {  
 case 1: {  
  
 intList.append(intList, -4);  
 intList.append(intList, -2);  
 intList.append(intList, 1);  
 intList.append(intList, 2);  
 intList.append(intList, 4);  
 intList.append(intList, 8);  
 intList.append(intList, 16);  
 intList.append(intList, 32);  
 intList.append(intList, 64);  
 intList.append(intList, 128);  
 intList.append(intList, 256);  
 intList.append(intList, 512);  
 intList.append(intList, 1024);  
 intList.append(intList, 2048);  
 intList.append(intList, 4096);  
 intList.append(intList, 8192);  
 intList.append(intList, 16384);  
 intList.append(intList, 32768);  
 intList.append(intList, 65536);  
 intList.display(intList);  
 type\_choice\_flag = 1;  
 break;  
 }  
 case 2: {  
 floatList.append(floatList, 2.71828);  
 floatList.append(floatList, 3.1415926535);  
 floatList.append(floatList, 2.0);  
 floatList.append(floatList, 2.1);  
 floatList.append(floatList, 2.23);  
 floatList.append(floatList, 123.123);  
 floatList.display(floatList);  
 type\_choice\_flag = 2;  
 break;  
 }  
 case 3: {  
 Matrix matrix(2);  
 matrix.data[0][0] = 1;  
 matrix.data[0][1] = 2;  
 matrix.data[1][0] = 3;  
 matrix.data[1][1] = 4;  
 matrixList.append(matrixList, matrix);  
  
 Matrix matrix1add(2);  
 matrix1add.data[0][0] = 2;  
 matrix1add.data[0][1] = 5;  
 matrix1add.data[1][0] = 6;  
 matrix1add.data[1][1] = 8;  
 Matrix matrix2add(2);  
 matrix2add.data[0][0] = 1;  
 matrix2add.data[0][1] = 3;  
 matrix2add.data[1][0] = 3;  
 matrix2add.data[1][1] = 1;  
 Matrix addresult = matrix1add.add(matrix2add);  
 matrixList.append(matrixList, addresult);  
  
 Matrix matrix1sub(2);  
 matrix1sub.data[0][0] = 2;  
 matrix1sub.data[0][1] = 5;  
 matrix1sub.data[1][0] = 6;  
 matrix1sub.data[1][1] = 8;  
 Matrix matrix2sub(2);  
 matrix2sub.data[0][0] = 1;  
 matrix2sub.data[0][1] = 3;  
 matrix2sub.data[1][0] = 3;  
 matrix2sub.data[1][1] = 1;  
 Matrix subresult = matrix1sub.subtract(matrix2sub);  
 matrixList.append(matrixList, subresult);  
  
 Matrix matrix1mul(2);  
 matrix1mul.data[0][0] = 2;  
 matrix1mul.data[0][1] = 5;  
 matrix1mul.data[1][0] = 6;  
 matrix1mul.data[1][1] = 8;  
 Matrix matrix2mul(2);  
 matrix2mul.data[0][0] = 1;  
 matrix2mul.data[0][1] = 3;  
 matrix2mul.data[1][0] = 3;  
 matrix2mul.data[1][1] = 1;  
 Matrix mulresult = matrix1mul.multiply(matrix2mul);  
 matrixList.append(matrixList, mulresult);  
  
 Matrix matrix1tr(3);  
 matrix1tr.data[0][0] = 1;  
 matrix1tr.data[0][1] = 2;  
 matrix1tr.data[0][2] = 3;  
 matrix1tr.data[1][0] = 4;  
 matrix1tr.data[1][1] = 5;  
 matrix1tr.data[1][2] = 6;  
 matrix1tr.data[2][0] = 7;  
 matrix1tr.data[2][1] = 8;  
 matrix1tr.data[2][2] = 9;  
 Matrix trresult = matrix1tr.transpose();  
 matrixList.append(matrixList, trresult);  
  
 Matrix matrix1det(4);  
 matrix1det.data[0][0] = 3;  
 matrix1det.data[0][1] = 5;  
 matrix1det.data[0][2] = 7;  
 matrix1det.data[0][3] = 8;  
 matrix1det.data[1][0] = -1;  
 matrix1det.data[1][1] = 7;  
 matrix1det.data[1][2] = 0;  
 matrix1det.data[1][3] = 1;  
 matrix1det.data[2][0] = 0;  
 matrix1det.data[2][1] = 5;  
 matrix1det.data[2][2] = 3;  
 matrix1det.data[2][3] = 2;  
 matrix1det.data[3][0] = 1;  
 matrix1det.data[3][1] = -1;  
 matrix1det.data[3][2] = 7;  
 matrix1det.data[3][3] = 4;  
 int intermediate\_detresult = matrix1det.determinant();  
 Matrix detresult(1);  
 detresult.data[0][0] = intermediate\_detresult;  
 matrixList.append(matrixList, detresult);  
  
 type\_choice\_flag = 3;  
  
 matrixList.display(matrixList);  
 break;  
 }  
 default: {  
 cout << "Invalid choice" << endl;  
 break;  
 }  
 }  
 break;  
 }  
 while (1) {  
 int flag = 0;  
 cout << "Choose option:" << endl;  
 cout << "1. Delete list" << endl;  
 cout << "2. Insert node" << endl;  
 cout << "3. Delete node" << endl;  
 cout << "4. Show list" << endl;  
 cout << "0. Exit" << endl;  
 int list\_action\_choice;  
 cin >> list\_action\_choice;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 switch (list\_action\_choice) {  
 case 1: {  
 if (type\_choice\_flag == 1) {  
 intList.deleteList();  
 cout << "List deleted." << endl;  
 break;  
 } else if (type\_choice\_flag == 2) {  
 floatList.deleteList();  
 cout << "List deleted." << endl;  
 break;  
 } else if (type\_choice\_flag == 3) {  
 matrixList.deleteList();  
 cout << "List deleted." << endl;  
 break;  
 }  
 }  
 case 2: {  
 if (type\_choice\_flag == 1) {  
 int index;  
 int value;  
 cout << "Enter index to insert" << endl;  
 cin >> index;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 cout << "Enter value to insert" << endl;  
 cin >> value;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 intList.insertNode(intList, index, value);  
 intList.display(intList);  
 break;  
 }  
 if (type\_choice\_flag == 2) {  
 int index;  
 float value;  
 cout << "Enter index to insert" << endl;  
 cin >> index;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 cout << "Enter value to insert" << endl;  
 cin >> value;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 floatList.insertNode(floatList, index, value);  
 floatList.display(floatList);  
 break;  
 }  
 if (type\_choice\_flag == 3) {  
 int size, index;  
 cout << "Enter index of the node to place:" << endl;  
 cin >> index;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 cout << "Enter the number of rows and columns:" << endl;  
 cin >> size;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 if (size <= 0) {  
 cout << "Invalid matrix dimensions. Rows and columns must be greater than zero." << endl;  
 break;  
 }  
 Matrix value(size);  
 cout << "Enter new matrix data:" << endl;  
 cin >> value;  
 matrixList.insertNode(matrixList, index, value);  
 cout << "Node placed." << endl;  
 matrixList.display(matrixList);  
 break;  
 }  
 }  
 case 3: {  
 int index;  
 if (type\_choice\_flag == 1) {  
 cout << "Enter index to delete node" << endl;  
 cin >> index;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 intList.deleteNode(intList, index);  
 intList.display(intList);  
 break;  
 }  
 if (type\_choice\_flag == 2) {  
 cout << "Enter index to delete node" << endl;  
 cin >> index;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 floatList.deleteNode(floatList, index);  
 floatList.display(floatList);  
 break;  
 }  
 if (type\_choice\_flag == 3) {  
 cout << "Enter index to delete node" << endl;  
 cin >> index;  
 while (1) {  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input" << endl;  
 } else {  
 break;  
 }  
 }  
 matrixList.deleteNode(matrixList, index);  
 matrixList.display(matrixList);  
 break;  
 }  
 }  
 case 4: {  
 if (type\_choice\_flag == 1) {  
 intList.display(intList);  
 } else if (type\_choice\_flag == 2) {  
 floatList.display(floatList);  
 } else if (type\_choice\_flag == 3) {  
 matrixList.display(matrixList);  
 }  
 break;  
 }  
 case 0: {  
 flag = 1;  
 break;  
 }  
 }  
 if (flag == 1) {  
 break;  
 return 0;  
 }  
 }  
}

**matrix.cpp**

#include "matrix.h"  
#include <iostream>  
#include <limits>  
#include <stdexcept>  
#include <math.h>  
using namespace std;  
  
Matrix::Matrix() : size(0), data(nullptr) {}  
  
Matrix::Matrix(int size) : size(size) {  
 data = new int\* [size];  
 for (int i = 0; i < size; i++) {  
 data[i] = new int[size];  
 }  
}  
  
Matrix::Matrix(const Matrix& other) : size(other.size) {  
 data = new int\* [size];  
 for (int i = 0; i < size; i++) {  
 data[i] = new int[size];  
 for (int j = 0; j < size; j++) {  
 data[i][j] = other.data[i][j];  
 }  
 }  
}  
  
Matrix::~Matrix() {  
 freeMemory();  
}  
  
void Matrix::freeMemory() {  
 for (int i = 0; i < size; i++) {  
 delete[] data[i];  
 }  
 delete[] data;  
}  
  
void Matrix::input() {  
 cout << "Add matrix numbers " << size << "x" << size << ":" << endl;  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 while (true) {  
 int value;  
 cin >> value;  
 if (cin.fail()) {  
 cin.clear();  
 cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
 cout << "n/a" << endl;  
 exit(1);  
 } else {  
 data[i][j] = value;  
 break;  
 }  
 }  
 }  
 }  
}  
  
void Matrix::output() const {  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 cout << data[i][j] << " ";  
 }  
 cout << endl;  
 }  
}  
  
Matrix Matrix::add(const Matrix& other) const {  
 Matrix result(size);  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 result.data[i][j] = data[i][j] + other.data[i][j];  
 }  
 }  
 return result;  
}  
  
Matrix Matrix::subtract(const Matrix& other) const {  
 Matrix result(size);  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 result.data[i][j] = data[i][j] - other.data[i][j];  
 }  
 }  
 return result;  
}  
  
Matrix Matrix::multiply(const Matrix& other) const {  
 Matrix result(size);  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 result.data[i][j] = 0;  
 for (int k = 0; k < size; k++) {  
 result.data[i][j] += data[i][k] \* other.data[k][j];  
 }  
 }  
 }  
 return result;  
}  
  
Matrix Matrix::transpose() const {  
 Matrix result(size);  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 result.data[i][j] = data[j][i];  
 }  
 }  
 return result;  
}  
  
int Matrix::determinant() const {  
 if (size == 1) {  
 return data[0][0];  
 }  
 if (size == 2) {  
 return (data[0][0] \* data[1][1] - data[1][0] \* data[0][1]);  
 }  
 if (size == 3) {  
 return ((data[0][0]\*data[1][1]\*data[2][2]) + (data[0][1]\*data[1][2]\*data[2][0])  
 + (data[0][2]\*data[1][0]\*data[2][1]) - (data[0][2]\*data[1][1]\*data[2][0]) -  
 (data[0][1]\*data[1][0]\*data[2][2]) - (data[0][0]\*data[1][2]\*data[2][1]));  
 }  
 else {  
 int det = 0;  
 for (int x = 0; x < size; x++) {  
 Matrix submatrix(size - 1);  
 int subi = 0;  
 for (int i = 1; i < size; i++) {  
 int subj = 0;  
 for (int j = 0; j < size; j++) {  
 if (j == x) {  
 continue;  
 }  
 submatrix.data[subi][subj] = data[i][j];  
 subj++;  
 }  
 subi++;  
 }  
 det += (pow(-1, x) \* data[0][x] \* submatrix.determinant());  
 }  
 return det;  
 }  
}  
  
Matrix& Matrix::operator=(const Matrix& other) {  
 if (this != &other) {  
 freeMemory();  
 size = other.size;  
 data = new int\* [size];  
 for (int i = 0; i < size; i++) {  
 data[i] = new int[size];  
 for (int j = 0; j < size; j++) {  
 data[i][j] = other.data[i][j];  
 }  
 }  
 }  
 return \*this;  
}

**matrix.h**

#ifndef MATRIX\_H  
#define MATRIX\_H  
  
#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;  
  
class Matrix {  
public:  
 Matrix();  
 Matrix(int size);  
 Matrix(const Matrix&);  
 ~Matrix();  
  
 void input();  
 void output() const;  
 Matrix add(const Matrix&) const;  
 Matrix subtract(const Matrix&) const;  
 Matrix multiply(const Matrix&) const;  
 Matrix transpose() const;  
 int determinant() const;  
  
 Matrix& operator=(const Matrix&);  
  
 friend ostream& operator<<(ostream& os, const Matrix& matrix);  
  
 // Перегрузка оператора ввода  
 friend istream& operator>>(istream& is, Matrix& matrix) {  
 cout << "Enter elements for " << matrix.size << "x" << matrix.size << " matrix:" << endl;  
 for (int i = 0; i < matrix.size; i++) {  
 for (int j = 0; j < matrix.size; j++) {  
 is >> matrix.data[i][j];  
 }  
 }  
 return is;  
 }  
  
 // Метод для отладки: вывод матрицы  
 friend ostream& operator<<(ostream& os, const Matrix& matrix) {  
 for (int i = 0; i < matrix.size; i++) {  
 for (int j = 0; j < matrix.size; j++) {  
 os << matrix.data[i][j] << " ";  
 }  
 os << endl;  
 }  
 return os;  
 }  
  
 int size;  
 int\*\* data;  
private:  
 void freeMemory();  
};  
  
template<typename T>  
class Node {  
public:  
 T data;  
 Node\* next;  
 Node\* prev;  
  
 Node(const T& newData = T()) : data(newData), next(this), prev(this) {}  
};  
  
template<typename T>  
class List {  
private:  
 Node<T>\* head;  
 int size;  
public:  
 List() : head(nullptr), size(0) {}  
 ~List() {  
 deleteList();  
 }  
  
 void append(List<T>& list, const T& data) {  
 Node<T>\* newNode = new Node<T>(data);  
 if (list.isEmpty()) {  
 list.head = newNode;  
 } else {  
 Node<T>\* tail = list.head->prev;  
 tail->next = newNode;  
 newNode->prev = tail;  
 newNode->next = list.head;  
 list.head->prev = newNode;  
 }  
 list.size++;  
 }  
   
 void deleteList() {  
 if (!isEmpty()) {  
 Node<T>\* current = head;  
 do {  
 Node<T>\* temp = current;  
 current = current->next;  
 delete temp;  
 } while (current != head);  
 }  
 head = nullptr;  
 size = 0;  
 }  
  
 void display(const List<T>& list) {  
 if (list.isEmpty()) {  
 cout << "List is empty \n";  
 return;  
 }  
 Node<T>\* current = list.head;  
 int counter = 1;  
 do {  
 cout << counter << ") " << endl;  
 cout << current->data << endl;  
 current = current->next;  
 counter++;  
 } while (current != list.head);  
 }  
  
 bool isEmpty() const {  
 return head == nullptr;  
 }  
  
 Node<T>\* findNode(const List<T>& list, int index) {  
 if (index < 0 || index >= list.size) {  
 cout << "Out of bounds \n";  
 return nullptr;  
 }  
 Node<T>\* current = list.head;  
 for (int i = 0; i < index; i++) {  
 current = current->next;  
 }  
 return current;  
 }  
  
 void insertNode(List<T>& list, int index, const T& newValue) {  
 if (index < 0 || index > list.size) {  
 cout << "Out of bounds\n";  
 return;  
 }  
 Node<T>\* newNode = new Node<T>(newValue);  
  
 if (list.size == 0) {  
 list.head = newNode;  
 newNode->next = newNode->prev = newNode;  
 cout << "Inserted first node. Head data: " << newNode->data << endl;  
 } else if (index == 0) {  
 Node<T>\* tail = list.head->prev;  
 newNode->next = list.head;  
 newNode->prev = tail;  
 tail->next = newNode;  
 list.head->prev = newNode;  
 list.head = newNode;  
 cout << "Inserted node at the beginning: " << newNode->data << endl;  
 } else if (index == list.size) {  
 append(list, newValue);  
 return;  
 } else {  
 Node<T>\* current = findNode(list, index);  
 Node<T>\* previous = current->prev;  
 newNode->next = current;  
 newNode->prev = previous;  
 previous->next = newNode;  
 current->prev = newNode;  
 cout << "Inserted node in the middle. Data: " << newNode->data << endl;  
 }  
  
 list.size++;  
 cout << "List size after insertion: " << list.size << endl;  
 }  
  
 void deleteNode(List<T>& list, int index) {  
 if (index < 0 || index >= list.size) {  
 cout << "Out of bounds \n";  
 return;  
 }  
 Node<T>\* removeNode = findNode(list, index);  
 if (list.size == 1) {  
 delete removeNode;  
 list.head = nullptr;  
 cout << "Node deleted \n";  
 }  
 else {  
 Node<T>\* prevNode = removeNode->prev;  
 Node<T>\* nextNode = removeNode->next;  
 prevNode->next = nextNode;  
 nextNode->prev = prevNode;  
 if (removeNode == list.head) {  
 list.head = nextNode;  
 }  
 delete removeNode;  
 cout << "Node deleted \n";  
 }  
 list.size--;  
 }  
};  
  
  
#endif

**Вывод**В ходе выполнения лабораторной работы была разработана программа для работы с двусвязным циклическим списком матриц, целых чисел и десятичных дробей. Программа включает в себя операции сложения, вычитания, умножения, транспонирования, а также нахождение определителя матрицы; операции добавления узла в список, удаления узла из списка, удаления списка.

Классы Node и List сделаны шаблонными, чтобы список мог принимать различные типы данных.