**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Лабораторная работа № 3

*Перегрузка операторов*

*Вариант 6*

**Выполнил студент группы № M3111**

Гонтарь Тимур Сергеевич

**Подпись:**



Санкт-Петербург

2023

Условие ЛР:

Согласно варианту описать указанные типы данных и поместить их в отдельный

заголовочный файл, в нем же описать операторы, указанные в варианте. Реализацию

функций поместить с отдельный cpp файл.

Решение:

customqueue.h – header файл с классом очереди целых чисел

#ifndef LAB3\_CUSTOMQUEUE\_H  
#define LAB3\_CUSTOMQUEUE\_H  
  
class CustomQueue {  
private:  
 static const int maximum = 100;  
 int head;  
 int tail;  
 int \*cqueue;  
public:  
 CustomQueue();  
  
 CustomQueue(int, int \*);  
  
 CustomQueue(CustomQueue &);  
  
 ~CustomQueue();  
  
 CustomQueue &operator=(CustomQueue const &);  
  
 bool operator<<(int);  
  
 int operator>>(int);  
  
 bool pushBack(int);  
  
 int popFront();  
  
 void printQueue();  
};  
  
#endif //LAB3\_CUSTOMQUEUE\_H

customqueue.cpp – реализация класса очереди

#ifndef LAB3\_CUSTOMQUEUE\_C  
#define LAB3\_CUSTOMQUEUE\_C  
  
#include "customqueue.h"  
  
#include <iostream>  
  
using std::cout;  
using std::endl;  
  
//default constructor  
CustomQueue::CustomQueue() {  
 head = 0;  
 tail = 0;  
 cqueue = new int[maximum]{};  
}  
  
//constructor with given array and capacity  
CustomQueue::CustomQueue(int n, int \*arr) {  
 head = 0;  
 tail = 0;  
 cqueue = new int[maximum]{};  
 for (int i = 0; i < n; i++) {  
 if (!pushBack(arr[i])) {  
 break;  
 }  
 }  
}  
  
//constructor of copying  
CustomQueue::CustomQueue(CustomQueue &q) {  
 head = q.head;  
 tail = q.tail;  
 cqueue = new int[maximum]{};  
  
 for (int i = head; i <= tail; i++) {  
 cqueue[i % 100] = q.cqueue[i % 100];  
 }  
}  
  
//destructor  
CustomQueue::~CustomQueue() {  
 delete[] cqueue;  
}  
  
//overloading assignment operator with copying  
CustomQueue &CustomQueue::operator=(CustomQueue const &q) {  
 if (&q == this) {  
 return \*this;  
 }  
  
 head = q.head;  
 tail = q.tail;  
  
 delete[]cqueue;  
 cqueue = new int[maximum];  
  
 for (int i = head; i <= tail; i++) {  
 cqueue[i % 100] = q.cqueue[i % 100];  
 }  
  
 return \*this;  
}  
  
//overloading << operator with pushing back element  
bool CustomQueue::operator<<(int n) {  
 if (pushBack(n)) {  
 return true;  
 } else {  
 return false;  
 }  
}  
  
//overloading >> operator with pop front element  
int CustomQueue::operator>>(int n) {  
 int ans = popFront();  
 return ans;  
}  
  
//push into queue function  
bool CustomQueue::pushBack(int n) {  
 if (head != tail and (head % 100) == (tail % 100)) {  
 return false;  
 } else {  
 tail += 1;  
 cqueue[tail % 100] = n;  
 return true;  
 }  
}  
  
//pop from queue function  
int CustomQueue::popFront() {  
 if (head == tail) {  
 return INT32\_MIN;  
 } else {  
 int tmp = cqueue[head % 100];  
 cqueue[head % 100] = 0;  
 head += 1;  
 return tmp;  
 }  
}  
  
//print queue  
void CustomQueue::printQueue() {  
 cout << "Queue:" << endl;  
 for (int i = head; i <= tail; i++) {  
 cout << cqueue[i % 100] << " ";  
 }  
 cout << endl;  
}  
  
#endif //LAB3\_CUSTOMQUEUE\_C

testqueue.cpp – тесты для класса очереди

#include "customqueue.h"  
  
#include <iostream>  
  
using std::cin;  
using std::cout;  
using std::endl;  
  
inline CustomQueue &createQueue() {  
 static const int maximum = 100;  
 cout << "Enter queue: " << endl;  
  
 //make a dynamic array of start queue  
 cout << "Enter number of elements in queue" << endl;  
 int curlen;  
 cin >> curlen;  
 while (curlen <= 0 or curlen > maximum) {  
 cout << "The number of elements should be between 0 and 100" << endl;  
 cin >> curlen;  
 }  
  
 cout << "Enter the elements of queue" << endl;  
 int \*temparr = new int[curlen];  
 for (int i = 0; i < curlen; i++) {  
 cin >> temparr[i];  
 }  
  
 CustomQueue \*testqueue = new CustomQueue(curlen, temparr);  
  
 delete[] temparr;  
  
 return \*testqueue;  
}  
  
inline void testingQueue() {  
 cout << "Testing queue. First, enter a queue: " << endl;  
  
 CustomQueue testqueue = createQueue();  
  
 cout << "Enter elements to push" << endl;  
 int curpush;  
 cin >> curpush;  
 for (int i = 0; i < curpush; i++) {  
 int temp;  
 cin >> temp;  
 if (testqueue << temp) {  
 cout << "Element " << temp << " pushed successfully" << endl;  
 } else {  
 cout << "Unable to push element " << temp << ": queue overflow" << endl;  
 }  
 }  
  
 cout << "Now, enter the number of elements to pop" << endl;  
 int curpop;  
 cin >> curpop;  
 for (int i = 0; i < curpop; i++) {  
 int temp;  
 temp = testqueue.popFront();  
 if (temp != INT32\_MIN) {  
 cout << temp << endl;  
 } else {  
 cout << "Unable to pop element: queue is empty" << endl;  
 }  
 }  
  
 cout << endl;  
}

matrix.h – header файл для класса матрицы

#ifndef LAB3\_MATRIX\_H  
#define LAB3\_MATRIX\_H  
  
class Matrix {  
private:  
 static const int dim = 3;  
 float \*\*matrix;  
 float determinant;  
  
 static float countDeterminant(Matrix &);  
  
public:  
 Matrix();  
  
 Matrix(float \*\*);  
  
 Matrix(Matrix &);  
  
 ~Matrix();  
  
 Matrix &operator=(Matrix const &);  
  
 Matrix &operator\*(Matrix &);  
  
 Matrix &operator\*(float);  
  
 Matrix &operator+(Matrix &);  
  
 Matrix &operator-(Matrix &);  
  
 bool operator==(Matrix &);  
  
 bool operator!=(Matrix &);  
  
 bool operator>(Matrix &);  
  
 bool operator<(Matrix &);  
  
 void printMatrix();  
};  
  
  
#endif //LAB3\_MATRIX\_H

matrix.cpp – реализация класса матрицы

#ifndef LAB3\_MATRIX\_C  
#define LAB3\_MATRIX\_C  
  
#include "matrix.h"  
  
#include <iostream>  
  
using std::cout;  
using std::endl;  
  
//default constructor  
Matrix::Matrix() {  
 matrix = new float \*[dim];  
 for (int i = 0; i < dim; i++) {  
 matrix[i] = new float[dim]{};  
 }  
  
 determinant = 0;  
}  
  
//constructor with given array  
Matrix::Matrix(float \*\*arr) {  
 matrix = new float \*[dim];  
 for (int i = 0; i < dim; i++) {  
 matrix[i] = new float[dim];  
 for (int j = 0; j < dim; j++) {  
 matrix[i][j] = arr[i][j];  
 }  
 }  
  
 determinant = countDeterminant(\*this);  
}  
  
//constructor of copying  
Matrix::Matrix(Matrix &m) {  
 matrix = new float \*[dim];  
 for (int i = 0; i < dim; i++) {  
 matrix[i] = new float[dim];  
 for (int j = 0; j < dim; j++) {  
 matrix[i][j] = m.matrix[i][j];  
 }  
 }  
  
 determinant = m.determinant;  
}  
  
//destructor  
Matrix::~Matrix() {  
 for (int i = 0; i < dim; i++) {  
 delete[] matrix[i];  
 }  
 delete[] matrix;  
}  
  
//overloading assignment operator with copying  
Matrix &Matrix::operator=(Matrix const &m) {  
 if (&m == this) {  
 return \*this;  
 }  
  
 for (int i = 0; i < dim; i++) {  
 delete[] matrix[i];  
 }  
 delete[] matrix;  
  
 matrix = new float \*[dim];  
 for (int i = 0; i < dim; i++) {  
 matrix[i] = new float[dim];  
 for (int j = 0; j < dim; j++) {  
 matrix[i][j] = m.matrix[i][j];  
 }  
 }  
  
 determinant = m.determinant;  
  
 return \*this;  
}  
  
//overloading multiply operator for multiplying 2 matrices  
Matrix &Matrix::operator\*(Matrix &m) {  
 Matrix \*multiply = new Matrix();  
  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 float cur = 0;  
 for (int k = 0; k < dim; k++) {  
 cur += matrix[i][k] \* m.matrix[k][j];  
 }  
 multiply->matrix[i][j] = cur;  
 }  
 }  
  
 multiply->determinant = countDeterminant(\*multiply);  
  
 return \*multiply;  
}  
  
//overloading multiply operator for multiplying a matrix by a float number  
Matrix &Matrix::operator\*(float n) {  
 Matrix \*multiplyFloat = new Matrix();  
  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 multiplyFloat->matrix[i][j] = matrix[i][j] \* n;  
 }  
 }  
  
 multiplyFloat->determinant = n \* determinant;  
  
 return \*multiplyFloat;  
}  
  
//overloading addition operator for 2 matrices  
Matrix &Matrix::operator+(Matrix &m) {  
 Matrix \*addition = new Matrix();  
  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 addition->matrix[i][j] = matrix[i][j] + m.matrix[i][j];  
 }  
 }  
  
 addition->determinant = countDeterminant(\*addition);  
  
 return \*addition;  
}  
  
//overloading subtraction operator for 2 matrices  
Matrix &Matrix::operator-(Matrix &m) {  
 Matrix \*subtraction = new Matrix();  
  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 subtraction->matrix[i][j] = matrix[i][j] - m.matrix[i][j];  
 }  
 }  
  
 subtraction->determinant = countDeterminant(\*subtraction);  
  
 return \*subtraction;  
}  
  
//overloading comparison operator - equality  
bool Matrix::operator==(Matrix &m) {  
 if (determinant != m.determinant) {  
 return false;  
 }  
  
 bool flag = true;  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 if (matrix[i][j] != m.matrix[i][j]) {  
 flag = false;  
 break;  
 }  
 }  
 }  
  
 return flag;  
}  
  
//overloading comparison operator - inequality  
bool Matrix::operator!=(Matrix &m) {  
 if (determinant != m.determinant) {  
 return true;  
 }  
  
 bool flag = false;  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 if (matrix[i][j] != m.matrix[i][j]) {  
 flag = true;  
 break;  
 }  
 }  
 }  
  
 return flag;  
}  
  
//overloading comparison operator - matrix 1 is more than matrix 2  
bool Matrix::operator>(Matrix &m) {  
 bool flag = true;  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 if (matrix[i][j] <= m.matrix[i][j]) {  
 flag = false;  
 break;  
 }  
 }  
 }  
  
 return flag;  
}  
  
//overloading comparison operator - matrix 1 is less than matrix 2  
bool Matrix::operator<(Matrix &m) {  
 bool flag = true;  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 if (matrix[i][j] >= m.matrix[i][j]) {  
 flag = false;  
 break;  
 }  
 }  
 }  
  
 return flag;  
}  
  
//static function to calculate determinant for 3x3 matrix  
float Matrix::countDeterminant(Matrix &m) {  
 float plus = m.matrix[0][0] \* m.matrix[1][1] \* m.matrix[2][2] + m.matrix[0][1] \* m.matrix[1][2] \* m.matrix[2][0] +  
 m.matrix[0][2] \* m.matrix[1][0] \* m.matrix[1][2];  
 float minus = m.matrix[0][2] \* m.matrix[1][1] \* m.matrix[2][0] + m.matrix[0][1] \* m.matrix[1][0] \* m.matrix[2][2] +  
 m.matrix[0][0] \* m.matrix[1][2] \* m.matrix[2][1];  
 float ans = plus - minus;  
  
 return ans;  
}  
  
//print the matrix  
void Matrix::printMatrix() {  
 cout << "Matrix:" << endl;  
 for (int i = 0; i < dim; i++) {  
 for (int j = 0; j < dim; j++) {  
 cout << matrix[i][j] << " ";  
 }  
 cout << endl;  
 }  
 cout << endl;  
}  
  
#endif //LAB3\_MATRIX\_C

testmatrix.cpp – тесты для класса матрицы

#include "matrix.h"  
  
#include <iostream>  
  
using std::cin;  
using std::cout;  
using std::endl;  
  
inline Matrix &createMatrix(int number) {  
 static const int dimension = 3;  
 cout << "Enter matrix " << number << ":" << endl;  
  
 //make a 3x3 array in dynamic memory  
 float \*\*testarr = new float \*[dimension];  
 for (int i = 0; i < dimension; i++) {  
 testarr[i] = new float[dimension] {};  
 for (int j = 0; j < dimension; j++) {  
 cin >> testarr[i][j];  
 }  
 }  
  
 Matrix \*testmatrix = new Matrix(testarr);  
  
 //free the memory  
 for (int i = 0; i < dimension; i++) {  
 delete [] testarr[i];  
 }  
 delete [] testarr;  
  
 return \*testmatrix;  
}  
  
inline void testingMatrix() {  
 cout << "Testing matrix. First, enter 2 matrices and a real number: " << endl;  
  
 Matrix testmatrix1 = createMatrix(1);  
 Matrix testmatrix2 = createMatrix(2);  
  
 cout << "Now, enter a float number" << endl;  
  
 float num;  
 cin >> num;  
  
 cout << "Results:" << endl << endl;  
  
 Matrix \*temp = new Matrix();  
  
 cout << "Matrix 1 \* Matrix 2: " << endl;  
 \*temp = testmatrix1 \* testmatrix2;  
 temp->printMatrix();  
  
 cout << "Matrix 1 \* number: " << endl;  
 \*temp = testmatrix1 \* num;  
 temp->printMatrix();  
  
 cout << "Matrix 1 + Matrix 2: " << endl;  
 \*temp = testmatrix1 + testmatrix2;  
 temp->printMatrix();  
  
 cout << "Matrix 1 - Matrix 2: " << endl;  
 \*temp = testmatrix1 - testmatrix2;  
 temp->printMatrix();  
  
 if (testmatrix1 == testmatrix2) {  
 cout << "Matrix 1 is equal to Matrix 2" << endl;  
 } else if (testmatrix1 != testmatrix2) {  
 cout << "Matrix 1 is not equal to Matrix 2" << endl;  
 }  
  
 if (testmatrix1 > testmatrix2) {  
 cout << "Matrix 1 is more than Matrix 2" << endl;  
 } else if (testmatrix1 < testmatrix2) {  
 cout << "Matrix 1 is less than Matrix 2" << endl;  
 } else {  
 cout << "Can't compare Matrix 1 and Matrix 2" << endl;  
 }  
  
 cout << endl;  
 delete temp;  
}

main.cpp

#include "customqueue/testqueue.cpp"  
#include "matrix/testmatrix.cpp"  
  
int main () {  
 cout << "Hello" << endl;  
  
 testingMatrix();  
 testingQueue();  
  
 cout << "Bye!" << endl;  
  
 return 0;  
}

Вывод: в ходе данной лабораторной работы были реализованы классы, у которых были перегружены арифметические операторы.