MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY

“KHARKOV POLYTECHNICAL INSTITUTE”

LABORATORY WORK № 1

# “Creation and Use of C++ Classes”

Created by student of 1.КН.201.8г

Chukwu Irele Omike

Checked by

KHARKIV 2019

## 1 Training Assignment

### 1.1 Class for Representation of Simple Fractions

Create a class that represents a simple fraction. Implement constructors, a function for reduction of fractions, and overload operations +, -, \*, /, as well as stream input and output. Demonstrate class features in the main() function.

### 1.2 Class for Representation of Two-Dimensional Array

Develop a class to represent two-dimensional array of integers (matrix) of arbitrary size. Create constructors and destructors, overload operations of addition, subtraction and multiplication (according to the rules of working with matrices), getting items by index, as well as stream input and output. Create your own exception classes and generate relevant objects, if it is impossible to perform a particular operation.

Create a separate function that gets reference to an array and performs actions listed in the table. The function should not be friend or member function.

|  |  |  |  |
| --- | --- | --- | --- |
| **Index of variant  (from students list)** | **Rule of a source array transformation** | **Row count m** | **Column count n** |
| 1 | All elements with odd values should be doubled | 4 | 3 |
| 2 | All elements with even values should be replaced with their squares | 3 | 5 |
| 3 | All elements with null value should be replaced with ones | 3 | 4 |
| 4 | All elements with even values should be doubled | 4 | 5 |
| 5 | All elements should be replaced with their absolute magnitudes | 5 | 4 |
| 6 | All elements with even values should be tripled | 3 | 3 |
| 7 | All positive elements should be replaced with integer parts of their Briggs (base ten) logarithms | 4 | 5 |
| 8 | All negative elements should be replaced with their squares | 4 | 4 |
| 9 | All positive elements should be replaced with integer parts of their Napierian (natural) logarithms | 5 | 4 |
| 10 | All positive elements should be replaced with integer parts of their square roots | 3 | 5 |
| 11 | All positive elements with even values should be doubled | 5 | 4 |
| 12 | All negative elements with odd values should be tripled | 3 | 4 |
| 13 | All negative elements with odd values should be doubled | 4 | 3 |
| 14 | All positive elements with even values should be tripled | 3 | 5 |

Demonstrate class features in the main() function. Solve individual task with catching of possible exceptions.

### 1.3 Calculation of the Sum of Entered Values

Create a class with a private data member, getter, and constructor with one argument. This class also should contain a static private data member that stores the sum of all data members of previously created objects. Each call of the constructor includes adding of a new value to the static field. Public static function of the same class should return this sum.

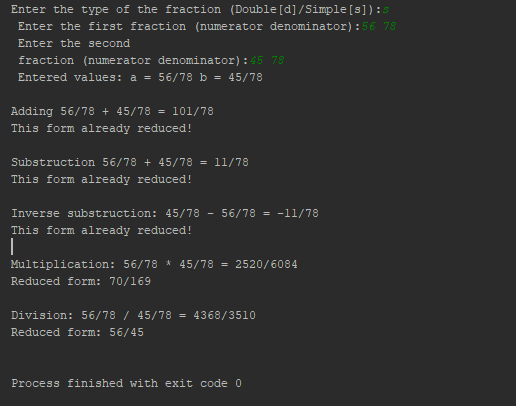
The main() function should contain creation of several objects and output of the calculated sum.

Task one :

The code :

#include <iostream>  
#include <string>  
  
using std::cin;  
using std::cout;  
using std::endl;  
using std::ostream;  
using std::istream;  
  
class Fraction  
{  
private:  
 long numerator;  
 long denominator;  
 long euclidean(long a, long b);  
public:  
 Fraction(void);  
 ~Fraction(void);  
  
 Fraction(int wholeNumber);  
 Fraction(double number);  
 Fraction(int numerator, int denominator);  
  
 Fraction operator+(Fraction fraction);  
 Fraction operator-(Fraction fraction);  
 Fraction operator\*(Fraction fraction);  
 Fraction operator/(Fraction fraction);  
  
 friend ostream& operator <<(ostream &out, Fraction &fraction);  
 friend istream& operator >>(istream &in, Fraction &fraction);  
  
 void setNumerator(int value);  
 void setDenominator(int value);  
  
 int getNumerator();  
 int getDenominator();  
 bool reduce(void);  
 void convertDoubleToFraction(double numerator);  
  
};  
  
ostream& operator <<(ostream &out, Fraction &fraction);  
istream& operator >>(istream &in, Fraction &fraction);  
  
void reducedValue(Fraction value)  
{  
 if (value.reduce())  
 {  
 cout << "Reduced form: " << value << endl << endl;  
 }  
 else  
 {  
 cout << "This form already reduced!" << endl << endl;  
 }  
}  
  
void outputResult(Fraction first, Fraction second)  
{  
 cout << "Entered values: " << "a = " << first << " b = " << second << endl << endl;  
  
 Fraction sum = first + second;  
 cout << "Adding " << first << " + " << second << " = " << sum << endl;  
 reducedValue(sum);  
  
 Fraction sub = first - second;  
 cout << "Substruction " << first << " + " << second << " = " << sub << endl;  
 reducedValue(sub);  
  
 Fraction inverseSub = second - first;  
 cout << "Inverse substruction: " << second << " - " << first << " = " << inverseSub << endl;  
 reducedValue(inverseSub);  
  
 Fraction f = first \* second;  
 cout << "Multiplication: " << first << " \* " << second << " = " << f << endl;  
 reducedValue(f);  
  
 Fraction g = first / second;  
 cout << "Division: " << first << " / " << second << " = " << g << endl;  
 reducedValue(g);  
}  
  
Fraction convertDoubleToFraction(Fraction fraction)  
{  
 double number;  
 cout << "Enter a floating point number: ";  
 cin >> number;  
 fraction.convertDoubleToFraction(number);  
 cout << "Converted value: " << fraction << endl;  
 return fraction;  
}  
  
  
int main()  
{  
 Fraction fraction1, fraction2;  
 cout << "Enter the type of the fraction (Double[d]/Simple[s]): ";  
 std::string inputType;  
 cin >> inputType;  
  
 if (inputType.size() == 1 && inputType == "d")  
 {  
 double number1, number2;  
  
 fraction1 = convertDoubleToFraction(fraction1);  
 fraction2 = convertDoubleToFraction(fraction2);  
  
 outputResult(fraction1, fraction2);  
  
 }  
 else if (inputType.size() == 1 && inputType == "s")  
 {  
 cout << "Enter the first fraction (numerator denominator): ";  
 cin >> fraction1;  
  
 cout << "Enter the second fraction (numerator denominator): ";  
 cin >> fraction2;  
  
 outputResult(fraction1, fraction2);  
 }  
 else  
 {  
 cout << "Error!";  
 }  
 cin.get();  
 return 0;  
}  
  
Fraction::Fraction(void) {  
 numerator = 0;  
 denominator = 0;  
}  
  
Fraction::Fraction(int whole\_number) {  
 numerator = whole\_number;  
 denominator = 1;  
}  
  
Fraction::Fraction(double Number)  
{  
 this->convertDoubleToFraction(Number);  
}  
  
Fraction::Fraction(int numerator, int denominator)  
{  
 this->numerator = numerator;  
 this->denominator = denominator;  
}  
Fraction::~Fraction(void)  
{  
}  
  
Fraction Fraction::operator+(Fraction fraction)  
{  
 Fraction resultFraction;  
  
 if (this->denominator == fraction.getDenominator()) {  
 resultFraction.setNumerator(this->numerator + fraction.getNumerator());  
 resultFraction.setDenominator(this->denominator);  
 }  
 else {  
 resultFraction.setNumerator((this->numerator \* fraction.getDenominator()) +  
 (fraction.getNumerator() \* this->denominator));  
 resultFraction.setDenominator(this->denominator \* fraction.getDenominator());  
 }  
  
 return resultFraction;  
}  
  
Fraction Fraction::operator-(Fraction fraction) {  
 Fraction resultFraction;  
  
 if (this->denominator == fraction.getDenominator()) {  
 resultFraction.setNumerator(this->numerator - fraction.getNumerator());  
 resultFraction.setDenominator(this->denominator);  
 }  
 else {  
 resultFraction.setNumerator((this->numerator \* fraction.getDenominator()) - (fraction.getNumerator() \* this->denominator));  
 resultFraction.setDenominator(this->denominator \* fraction.getDenominator());  
 }  
  
 return resultFraction;  
}  
  
Fraction Fraction::operator\*(Fraction fraction) {  
 Fraction resultFraction;  
  
 resultFraction.setNumerator(this->numerator \* fraction.getNumerator());  
 resultFraction.setDenominator(this->denominator \* fraction.getDenominator());  
  
 return resultFraction;  
}  
  
Fraction Fraction::operator/(Fraction fraction) {  
 Fraction resultFraction;  
  
 resultFraction.setDenominator(this->denominator \* fraction.getNumerator());  
 resultFraction.setNumerator(this->numerator \* fraction.getDenominator());  
  
 return resultFraction;  
}  
  
int Fraction::getNumerator()  
{  
 return numerator;  
}  
  
int Fraction::getDenominator()  
{  
 return denominator;  
}  
  
void Fraction::setNumerator(int value)  
{  
 numerator = value;  
}  
  
void Fraction::setDenominator(int value)  
{  
 denominator = value;  
}  
  
ostream& operator<<(ostream& out, Fraction& fraction)  
{  
 // output the frACTION N/D  
 out << fraction.getNumerator();  
 out << "/";  
 out << fraction.getDenominator();  
 return out;  
}  
  
istream& operator>>(istream & in, Fraction & fraction)  
{  
 int numerator, denominator;  
 //out << "Please enter numerator and denominator: \n";  
 in >> numerator >> denominator;  
 fraction.setNumerator(numerator);  
 fraction.setDenominator(denominator);  
 return in;  
}  
  
//Recursive euclidean func  
long Fraction::euclidean(long a, long b) {  
 return b == 0 ? a : this->euclidean(b, a % b);  
}  
  
//Reduce the fraction  
bool Fraction::reduce(void) {  
 long gcd(this->euclidean(this->numerator, this->denominator));  
  
 if (1 < gcd) {  
 this->numerator /= gcd;  
 this->denominator /= gcd;  
  
 return true;  
 }  
 else {  
 return false;  
 }  
}  
  
void Fraction::convertDoubleToFraction(double Number)  
{  
 this->denominator = 1;  
  
 while (((double)(int)Number) != Number) {  
 Number = Number \* 10;  
 this->denominator = this->denominator \* 10;  
 }  
  
 this->numerator = (long)Number;  
 this->reduce();  
}

Execution :

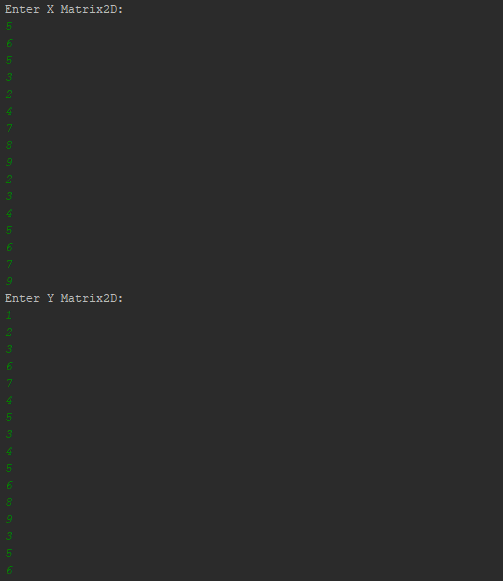


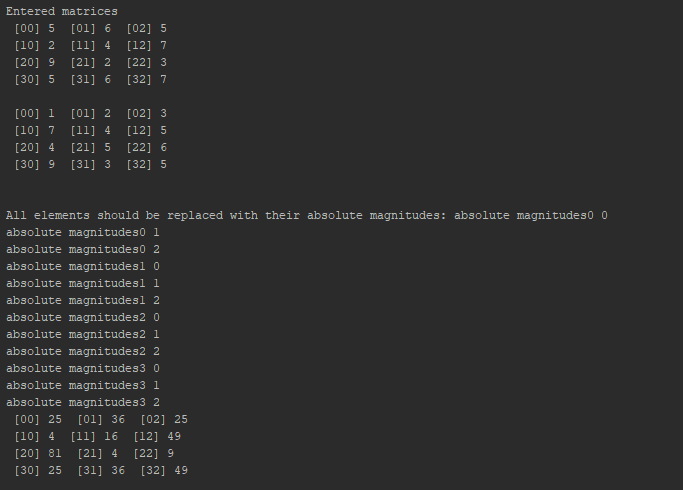
Task two:

The code :

#include <iostream>  
#include <cmath>  
  
using std::cout;  
using std::cin;  
using std::istream;  
using std::endl;  
using std::ostream;  
  
class Matrix2D  
{  
 friend ostream& operator<<(ostream& out, const Matrix2D& m)  
 {  
 for (int i = 0; i < m.rows; i++)  
 {  
 for (int j = 0; j < m.cols; j++)  
 {  
 out << " [" << i << j << "] " << m(i, j) << " ";  
 }  
 out << endl;  
 }  
 return out;  
 }  
  
 friend istream& operator >>(istream& in, Matrix2D& m)  
 {  
 for (int i = 0; i < m.rows; i++)  
 {  
 for (int j = 0; j < m.rows; j++)  
 {  
 in >> m.Matrix[i][j];  
 }  
 }  
 return in;  
 }  
  
 private:  
 int \*\*Matrix;  
 int rows;  
 int cols;  
 int next\_indexR;  
 int next\_indexC;  
 public:  
 class OutOfBounds  
 {  
 int outOfBounIndexRow;  
 int outOfBounIndexCol;  
  
 public:  
 OutOfBounds(int x, int y) : outOfBounIndexRow(x), outOfBounIndexCol(y) {}  
 int getOutOfBounIndexRow() const { return outOfBounIndexRow; }  
 int getOutOfBounIndexCol() const { return outOfBounIndexCol; }  
 };  
  
 Matrix2D()  
 {  
 Matrix = alloc(0, 0);  
 next\_indexR = 0;  
 next\_indexC = 0;  
 }  
  
 ~Matrix2D();  
  
 int& operator() (int i, int j)  
 {  
 if (i >= rows || j >= cols)  
 throw OutOfBounds(i, j);  
 return Matrix[i][j];  
 }  
 const int& operator() (int i, int j) const  
 {  
 if (i >= rows || j >= cols)  
 throw OutOfBounds(i, j);  
 return Matrix[i][j];  
 }  
  
 int \*getMatrix2D();  
 int getRows() { return rows; }  
 int getCols() { return cols; }  
 int getElementByIndex(int x, int y) { return Matrix[x][y]; }  
 int getNextIndexR() { return next\_indexR; }  
 int getNextIndexC() { return next\_indexC; }  
  
 int getValueAt(int i, int j)  
 {  
 return this->Matrix[i][j];  
 }  
  
 void addElement(int element);  
  
 Matrix2D operator+(const Matrix2D& m) const  
 {  
 // first, make sure matrices can be added. if not, return original matrix  
 if (rows != m.rows || cols != m.cols) {  
 cout << "Matrix sizes do not match. Mission impossible.";  
 return (\*this);  
 }  
 else {  
 //matrix new\_mat(row,col);  
 Matrix2D new\_mat;  
 new\_mat.alloc(rows, cols);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 new\_mat(i, j) = Matrix[i][j] + m(i, j);  
 }  
 }  
 return new\_mat;  
 }  
 }  
  
 /\*int\* operator[](const Matrix& m){  
 return m.matrix[i];  
 }\*/  
 Matrix2D operator-(const Matrix2D& m)  
 {  
 Matrix2D resultMatrix;  
  
 // first, make sure matrices can be added. if not, return original Matrix2D  
 if (rows != m.rows || cols != m.cols) {  
 cout << "Matrix2D sizes do not match. Mission impossible.";  
 return (\*this);  
 }  
  
 //Matrix2D new\_mat(row,col);  
 Matrix2D new\_mat;  
 new\_mat.alloc(rows, cols);  
  
 for (int i = 0; i < rows; i++)  
 {  
 for (int j = 0; j < cols; j++)  
 {  
 new\_mat.Matrix[i][j] = Matrix[i][j] - m.Matrix[i][j];  
 }  
 }  
 return new\_mat;  
 }  
  
 Matrix2D operator\*(const Matrix2D& m)  
 {  
 if (cols == m.rows)  
 {  
 Matrix2D new\_mat;  
 new\_mat.alloc(rows, m.cols);  
  
 for (int i = 0; i < rows; i++)  
 {  
 for (int j = 0; j < m.cols; j++)  
 {  
 //new\_mat.Matrix2D[i][j] = 0;  
 for (int k = 0; k < cols; k++)  
 new\_mat.Matrix[i][j] += Matrix[i][k] \* m.Matrix[k][j];  
 }  
 }  
 return new\_mat;  
 }  
  
 }  
  
 void addElByIndex(int i, int j, int element)  
 {  
 Matrix[i][j] = element;  
 }  
 int getMatrix2DVal(int i, int j) { return Matrix[i][j]; }  
 int \*\*alloc(int rows, int cols);  
 void dealloc();  
};  
  
Matrix2D::~Matrix2D()  
{  
 dealloc();  
}  
  
int \*\*Matrix2D::alloc(int rows, int cols)  
{  
 this->rows = rows;  
 this->cols = cols;  
 next\_indexC = 0;  
 next\_indexR = 0;  
  
 Matrix = new int \*[rows];  
 for (int row = 0; row < rows; row++)  
 {  
 Matrix[row] = new int[cols];  
 }  
 return Matrix;  
}  
void Matrix2D::dealloc() {  
 for (int row = 0; row < rows; row++)  
 {  
 delete[] Matrix[row];  
 }  
 delete[] Matrix;  
}  
  
int \*Matrix2D::getMatrix2D()  
{  
 int \*start = Matrix[0];  
  
 return start;  
}  
  
void outputInfo(int a, Matrix2D m);  
void Triple(Matrix2D& m);  
int main()  
{  
  
 Matrix2D X, Y;  
 X.alloc(4, 3);  
 Y.alloc(4, 3);  
  
 try {  
 cout << "Enter X Matrix2D: " << endl;  
 cin >> X;  
  
 cout << "Enter Y Matrix2D: " << endl;  
 cin >> Y;  
  
 }  
 catch (Matrix2D::OutOfBounds e) {  
 cout << "Bad row index: " << e.getOutOfBounIndexRow() << endl;  
 cout << "Bad col index: " << e.getOutOfBounIndexCol() << endl;  
  
 }  
  
 cout << "Entered matrices\n";  
 cout << X << endl;  
 cout << Y << endl;  
 cout << "\nAll elements should be replaced with their absolute magnitudes: ";  
  
 //Matrix2D A, all even values are tripled  
 Triple(X);  
 cout << X;  
 cout << endl;  
 system("pause");  
 return 0;  
}  
  
void outputInfo(int a, Matrix2D m) {  
 cout << "Element " << a << " was added as [" << m.getNextIndexR() << "][" << m.getNextIndexC() << "]" << endl;  
}  
  
void Triple(Matrix2D& m)  
{  
 for (int i = 0; i < m.getRows(); i++) {  
 for (int j = 0; j < m.getCols(); j++) {  
  
 if (m(i, j) / 2) {  
 cout << "absolute magnitudes" << i << " " << j << "\n";  
 m(i, j) = (m(i, j) \* (m(i, j)));  
 }  
 }  
 }  
}

Execution :





Task 3:

#include <iostream>  
using std::cout;  
using std::endl;  
  
class ObjectSum {  
private:  
 static int sum;  
public:  
 static int getSum() { return sum; }  
  
 ObjectSum() {  
 sum += 1;  
 }  
};  
  
int ObjectSum::sum = 0;  
  
int main()  
{  
 ObjectSum a;  
 cout << a.getSum() << endl;  
 ObjectSum b;  
 cout << b.getSum() << endl;  
 ObjectSum c;  
 cout << c.getSum() << endl;  
 ObjectSum d;  
 cout << d.getSum() << endl;  
 ObjectSum e;  
 cout << e.getSum() << endl;  
  
 //cin.get();  
  
 return 0;  
}

Execution:

