­­­Лабораторна робота 3

Побойний Артем, 304 група

Завдання 1:

main.cpp

#include <cstdlib>  
#include <iostream>  
#include "Circle.h"  
   
using namespace std;  
   
int main(int argc, char \*argv[])  
{  
     
           for(;;)  
           {  
            try  
             {  
               double radius;  
               string color;  
               cout<<"Enter the color: ";  
               cin>>color;  
               cout<<"Enter the radius: ";  
               cin>>radius;    
              Circle cir(radius);  
              cout<<"Radius: "<<cir.getRadius()<<endl;  
              cout<<"Area: "<<cir.getArea()<<endl;  
              cout<<"Color: "<<color<<endl;  
              cout<<"Circumference: "<<cir.getCircumference()<<endl;  
              break;        
              }  
              catch(double e)  
              {  
                    cout<<"The radius cannot be 0.0"<<endl<<endl;  
                    continue;  
              }  
              }  
     
     
    system("PAUSE");  
    return 0;

}

Circle.cpp

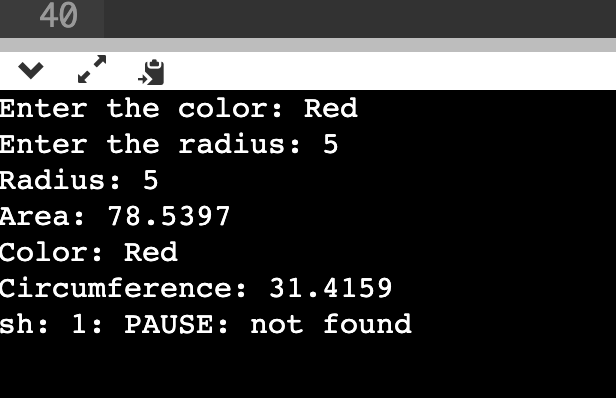
#include <iostream>  
#include <cstdlib>  
#include "Circle.h"  
#include "ZeroException.h"  
   
using namespace std;  
   
             Circle::Circle()  
             {  
                  radius = 1.0;  
                  color = "White";  
             }  
              Circle::Circle(double radi)  
             {  
                 if( radi == 0 || radi == 0.0)  
                 {  
                     ZeroException ex;  
                     ex.Checkradius();  
                 }  
                 else  
                 {  
                   radius = radi;    
                 }                    
                 
             }  
             
               
             void Circle::setRadius(double r)  
             {  
              if( r == 0 || r == 0.0)  
                 {  
                     ZeroException ex;  
                     ex.Checkradius();  
                 }  
                 else  
                 {  
                   radius = r;    
                 }  
             }  
               
             void Circle::setColor(string c)  
             {  
                 color = c;  
             }  
               
             double Circle::getRadius()  
             {  
                return radius;    
             }  
               
             string Circle::getColor()  
             {  
                 return color;  
             }  
               
             double Circle::getArea()  
             {  
                return pi \* radius \* radius;  
             }  
               
               
             double Circle::getCircumference()  
             {  
                  return 2 \* pi \* radius;    
             }  
               
               
             void  ZeroException::Checkradius()  
            {  
                    throw 0.0;                    
            }

Circle.h

#ifndef CIRCLEH  
#define CIRCLEH  
#include <iostream>  
#include <cstdlib>  
   
using namespace std;  
   
class Circle  
{  
      public:  
             Circle();  
               
             Circle(double radi);  
               
             void setRadius(double r);  
               
             void setColor(string c);  
               
             string getColor();  
               
             double getRadius();  
               
             double getArea();  
               
             double getCircumference();  
               
               
      private:  
              double radius;  
              string color;  
              const  double pi = 3.14159;      
};  
   
#endif

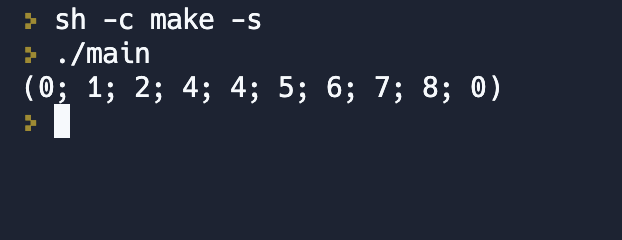
ZeroExepition.h

#ifndef ZeroExceptionH  
#define ZeroExceptionH  
#include <iostream>  
#include <cstdlib>  
   
using namespace std;  
   
class ZeroException  
{  
      public:  
            void  Checkradius();  
                 
};  
   
#endif



Завдання 2:

#include <iostream>  
#include <stdexcept>  
   
class VectorOfDouble {  
 public:  
  VectorOfDouble() : size(1), data(new double[size]()) {}  
  VectorOfDouble(size\_t size) : size(size), data(new double[size]) {  
    for (size\_t i = 0; i < size; ++i) {  
      data[i] = i;  
    }  
  }  
  VectorOfDouble(int size, double initializer) : size(size),  
    data(new double[size]) {  
    setAllTo(initializer);  
  }  
  VectorOfDouble(const VectorOfDouble &other) {  
    copyFrom(other);  
  }  
  VectorOfDouble &operator=(const VectorOfDouble &other) {  
    return copyFrom(other);  
  }  
  virtual ~VectorOfDouble() {  
    delete [] data;  
  }  
  double &get(size\_t position) {  
    if (position >= size) {  
      throw std::out\_of\_range("index went out of range");  
    }  
    return data[position];  
  }  
  const double &get(size\_t position) const {  
    if (position >= size) {  
      throw std::out\_of\_range("index went out of range");  
    }  
    return data[position];  
  }  
  void set(size\_t position) {  
    if (position >= size) {  
      throw std::out\_of\_range("index went out of range");  
    }  
    data[position] = 0;  
  }  
  void set(size\_t position, double value) {  
    if (position >= size) {  
      throw std::out\_of\_range("index went out of range");  
    }  
    data[position] = value;  
  }  
  size\_t getSize() const { return size; }  
  void setAllTo(double value) {  
    for (size\_t i = 0; i < size; ++i) {  
      data[i] = value;  
    }  
  }  
  VectorOfDouble &copyFrom(const VectorOfDouble &other) {  
    if (this != &other) {  
      size = other.size;  
      delete [] data;  
      data = new double[size];  
      for (size\_t i = 0; i < size; ++i) {  
        data[i] = other.data[i];  
      }  
    }  
    return \*this;  
  }  
  static size\_t getMaximumSize() { return static\_cast<size\_t>(-1); }  
 private:  
  size\_t size;  
  double \*data;  
};  
   
std::ostream &operator<<(std::ostream &stream, const VectorOfDouble &v) {  
  stream << "(";  
  for (size\_t i = 0; i < v.getSize(); ++i) {  
    stream << v.get(i);  
    if (i < v.getSize() - 1) stream << "; ";  
  }  
  return stream << ")";  
}  
   
int main(int argc, char \*argv[]) {  
  VectorOfDouble v(10);  
  v.set(9);  
  v.set(3, 4);  
  std::cout << v << std::endl;  
  return 0;  
}



Завдання 3:

#include <iostream>

#include <iomanip>

using namespace std;

class Matrix

{

public:

static int matrix\_count;

string error\_state = "";

Matrix(unsigned int \_n = 3)

{

matrix\_count++;

n = \_n;

m = \_n;

matrix = new int\* [n];

if (matrix == nullptr)

{

error\_state = "Недостатньо пам'яті";

return;

}

for (int i = 0; i < n; i++)

{

matrix[i] = new int[m];

if (matrix[i] == nullptr)

{

error\_state = "Недостатньо пам'яті";

return;

}

for (int j = 0; j < m; j++)

{

matrix[i][j] = 0;

}

}

}

Matrix(unsigned int \_n, unsigned int \_m, int value)

{

matrix\_count++;

n = \_n;

m = \_m;

matrix = new int\* [n];

if (matrix == nullptr)

{

error\_state = "Недостатньо пам'яті";

return;

}

for (int i = 0; i < n; i++)

{

matrix[i] = new int[m];

if (matrix[i] == nullptr)

{

error\_state = "Недостатньо пам'яті";

return;

}

for (int j = 0; j < m; j++)

{

matrix[i][j] = value;

}

}

}

Matrix(const Matrix& other)

{

matrix\_count++;

n = other.n;

m = other.m;

matrix = new int\* [n];

if (matrix == nullptr)

{

error\_state = "Недостатньо пам'яті";

return;

}

for (int i = 0; i < n; i++)

{

matrix[i] = new int[m];

if (matrix[i] == nullptr)

{

error\_state = "Недостатньо пам'яті";

return;

}

for (int j = 0; j < m; j++)

{

matrix[i][j] = other.matrix[i][j];

}

}

}

Matrix operator=(const Matrix& other)

{

n = other.n;

m = other.m;

if (matrix)

{

for (int i = 0; i < n; i++)

delete[]matrix[i];

delete[]matrix;

}

matrix = new int\* [n];

if (matrix == nullptr)

{

error\_state = "Недостатньо пам'яті";

return Matrix();

}

for (int i = 0; i < n; i++)

{

matrix[i] = new int[m];

if (matrix[i] == nullptr)

{

error\_state = "Недостатньо пам'яті";

return Matrix();

}

for (int j = 0; j < m; j++)

{

matrix[i][j] = other.matrix[i][j];

}

}

}

~Matrix()

{

matrix\_count--;

for (int i = 0; i < n; i++)

{

delete[]matrix[i];

}

delete[]matrix;

}

void set(unsigned int i, unsigned int j, int value)

{

if (i >= n || j >= m)

{

error\_state = "Індекс поза діапазоном";

return;

}

matrix[i][j] = value;

}

int get(unsigned int i, unsigned int j)

{

if (i >= n || j >= m)

{

error\_state = "Індекс поза діапазоном";

return 1;

}

return matrix[i][j];

}

unsigned int rows()

{

return n;

}

unsigned int columns()

{

return m;

}

void randomize(int a, int b)

{

a \*= 100; b \*= 100;

for (unsigned int i = 0; i < n; i++)

{

for (unsigned int j = 0; j < m; j++)

{

matrix[i][j] = ((rand() \* rand()) % (b - a) + a) / 100;

}

}

}

Matrix operator+(Matrix other)

{

Matrix result(n, m, 0);

if (n != other.n || m != other.m)

{

result.error\_state = "Розмір матриці недійсний";

return result;

}

for (unsigned int i = 0; i < n; i++)

{

for (unsigned int j = 0; j < m; j++)

{

result.matrix[i][j] = matrix[i][j] + other.matrix[i][j];

}

}

return result;

}

Matrix operator-(Matrix other)

{

Matrix result(n, m, 0);

if (n != other.n || m != other.m)

{

result.error\_state = "Розмір матриці недійсний";

return result;

}

for (unsigned int i = 0; i < n; i++)

{

for (unsigned int j = 0; j < m; j++)

{

result.matrix[i][j] = matrix[i][j] - other.matrix[i][j];

}

}

return result;

}

Matrix operator\*(int x)

{

Matrix result(n, m, 0);

for (unsigned int i = 0; i < n; i++)

{

for (unsigned int j = 0; j < m; j++)

{

result.matrix[i][j] = matrix[i][j] \* x;

}

}

return result;

}

bool operator==(Matrix other)

{

if (n != other.n || m != other.m)

return false;

for (unsigned int i = 0; i < n; i++)

{

for (unsigned int j = 0; j < m; j++)

{

if (matrix[i][j] != other.matrix[i][j])

return false;

}

}

return true;

}

bool operator!=(Matrix other)

{

return !(\*this == other);

}

bool operator>(Matrix other)

{

if (n != other.n || m != other.m)

return false;

int count = 0;

for (unsigned int i = 0; i < n; i++)

{

for (unsigned int j = 0; j < m; j++)

{

if (matrix[i][j] > other.matrix[i][j])

count++;

}

}

return (count == n \* m);

}

bool operator<(Matrix other)

{

if (n != other.n || m != other.m)

return false;

int count = 0;

for (unsigned int i = 0; i < n; i++)

{

for (unsigned int j = 0; j < m; j++)

{

if (matrix[i][j] < other.matrix[i][j])

count++;

}

}

return (count == n \* m);

}

friend ostream& operator<<(ostream& out, Matrix other)

{

for (int i = 0; i < other.n; i++)

{

for (int j = 0; j < other.m; j++)

{

out << fixed;

out << setprecision(2) << setw(8) << other.matrix[i][j];

}

out << endl;

}

return out;

}

private:

unsigned int n, m;

int\*\* matrix;

};

int Matrix::matrix\_count = 0;

int main()

{

srand(time(nullptr));

Matrix m1(3), m2(3);

Matrix\* m3 = new Matrix();

m1.randomize(1, 10);

m2.randomize(1, 10);

m3->randomize(1, 10);

cout << "Matrix1:\n" << m1 << endl;

cout << "Matrix2:\n" << m2 << endl;

cout << "Matrix3:\n" << \*m3 << endl;

cout << "Сума матриць Matrix1 і Matrix2:\n" << m1 + m2 << endl;

cout << "Різниця матриць Matrix1 і Matrix2:\n" << m1 - m2 << endl;

cout << "Matrix1 \* 2:\n" << m1 \* 2 << endl;

cout << "Matrix1 < (Matrix1 \* 2): " << boolalpha << (m1 < (m1 \* 2)) << endl;

cout << "Matrix1 > (Matrix1 \* 2): " << boolalpha << (m1 > (m1 \* 2)) << endl;

cout << "Matrix1 == Matrix1: " << boolalpha << (m1 == m1) << endl;

cout << "Matrix1 == Matrix2: " << boolalpha << (m1 == m2) << endl;

cout << "Matrix1 != Matrix1: " << boolalpha << (m1 != m1) << endl;

cout << "Matrix1 != Matrix2: " << boolalpha << (m1 != m2) << endl;

m1.set(0, 0, 100);

cout << "Встановіть Matrix1[0][0] як 100 і отримайте його: " << m1.get(0, 0) << endl;

cout << "Кількість створених матриць: " << Matrix::matrix\_count << endl;

cout << "Зняття матриці Matrix3" << endl;

delete m3;

cout << "Кількість створених матриць: " << Matrix::matrix\_count << endl;

}

