МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №1**

по курсу объектно-ориентированное программирование 3 семестр, 2021/22 уч. Год

Студент Карпов Даниил Антонович М8О-207Б-20

Преподаватель Дорохов Евгений Павлович

**Условие**

Задание:Вариант 12: Трапеция, Ромб, 5-угольник.

Необходимо спроектировать и запрограммировать на языке C++ классы трех фигур, согласно варианту задания. Классы должны удовлетворять следующим правилам:  Должны быть названы также, как в вариантах задания и расположенны в раздельных файлах: отдельно заголовки (имя\_класса\_с\_маленькой\_буквы.h), отдельно описания методов (имя\_класса\_с\_маленькой\_буквы.cpp); 

Иметь общий родительский класс Figure; 

Содержать конструктор по умолчанию;  Содержать конструктор, принимающий координаты вершин фигуры из стандартного потока std::cin, расположенных через пробел. Пример: 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0  Содержать набор общих методов:

◦ size\_t VertexesNumber() - метод, возвращающий количество вершин фигуры;

◦ double Area() - метод расчета площади фигуры;

◦ void Print(std::ostream& os) - метод печати типа фигуры и ее координат вершин в поток вывода os в формате: Pentagon: (+0.0, +0.0) (+0.0, +1.0) (+1.0, +0.5) (+2.0, +1.0) (+2.0б +0.0)\n

**Описание программы**

Исходный код лежит в 7 файликах:

1. main.cpp

2. pentagon.cpp

3. pentagon.h

4. rhombus.cpp

5. rhombus.h

6. trapezoid.cpp

7. trapezoid.h

**Дневник отладки**

Результат работы программы при тестовых данных:

Ромб: 0 0 1 2 2 0 1 -2

Трапеция: 0 0 10 0 7 1 2 1

5-угольник 0 0 0 1 1 0 2 1 2 0

Rhombus (0 0)(1 2)(2 0)(1 -2)

Trapezoid (0 0)(10 0)(7 1)(2 1)

Pentagon (0 0)(0 1)(1 0.5)(2 1)(2 0)

**Недочёты**

**Выводы**

Данная лабораторная работа позволила мне ознакомиться с абстрактными классами и наследованием. Я снова убедился в удобности и практичности такого подхода к программированию, как ООП.

Ссылка на гитхаб:

**Исходный код**

main.cpp

#include<iostream>

#include<cstdlib>

#include<vector>

#include"figure.h"

#include"rhombus.h"

#include"trapezoid.h"

#include"pentagon.h"

// Используется по умолчанию пространство имен "std"

using namespace std;

int main() {

vector<Figure\*> v;

int n;

cout << "Number of figures n=>";

cin >> n;

if ( n <= 0 ) {

cout << "n must be natural" << endl;

exit( 1 );

}

// Код, в котором может возникнуть исключение

try {

// Ввод фигур

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

int choice;

cout << "Enter: 1) rhombus; 2) trapezoid; 3) pentagon. Your choice =>";

cin >> choice;

switch ( choice ) {

case 1:

cout << "Enter coordinates of rhombus: " << endl;

v.push\_back( new Rhombus );

break;

case 2:

cout << "Enter coordinates of trapezoid: " << endl;

v.push\_back( new Trapezoid );

break;

case 3:

cout << "Enter coordinates of pentagon: " << endl;

v.push\_back( new Pentagon );

break;

default:

cout << "Bad choice" << endl;

break;

}

}

// Распечатка данных фигур

for ( size\_t i = 0; i < v.size(); i++ ) {

v[ i ]->Print( std::cout );

}

// Удаление фигур

for ( size\_t i = 0; i < v.size(); i++ )

delete v[ i ];

// Отлов исключения

} catch ( FigException e ) {

cout << "Caught an exception: " << e.message << endl;

}

return 0;

}

pentagon.cpp

#include<iomanip>

#include"pentagon.h"

Pentagon::Pentagon() : Figure() {

for ( int i = 0; i < 5; i++ ) {

cin >> x[ i ];

cin >> y[ i ];

}

if ( cin.fail() ) {

throw FigException( "Bad coordinate values in Pentagon constructor" );

}

// Проверка математической корректности пятиугольника

// Проверка равенства нулю сторон пятиугольника

double len0 = getLength( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ] );

double len1 = getLength( x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ] );

double len2 = getLength( x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ] );

double len3 = getLength( x[ 3 ], y[ 3 ], x[ 4 ], y[ 4 ] );

double len4 = getLength( x[ 4 ], y[ 4 ], x[ 0 ], y[ 0 ] );

if ( isDoubleZero( len0 ) ||

isDoubleZero( len1 ) ||

isDoubleZero( len2 ) ||

isDoubleZero( len3 ) ||

isDoubleZero( len4 ) ) {

throw FigException( "Error in Pentagon constructor: pentagon with zero side length" );

}

// Проверка непараллельности смежных отрезков

if ( isVectorsParallel( x[0]-x[1], y[0]-y[1], x[2]-x[1], y[2]-y[1] ) ||

isVectorsParallel( x[1]-x[2], y[1]-y[2], x[3]-x[2], y[3]-y[2] ) ||

isVectorsParallel( x[2]-x[3], y[2]-y[3], x[4]-x[3], y[4]-y[3] ) ||

isVectorsParallel( x[3]-x[4], y[3]-y[4], x[0]-x[4], y[0]-y[4] ) ||

isVectorsParallel( x[4]-x[0], y[4]-y[0], x[1]-x[0], y[1]-y[0] ) ) {

throw FigException( "Error in Pentagon constructor: pentagon with parallel adjacent sides" );

}

// Проверка переечения несмежных отрезков

if ( isIntersect( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ] ) ||

isIntersect( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ], x[ 3 ], y[ 3 ], x[ 4 ], y[ 4 ] ) ||

isIntersect( x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ], x[ 4 ], y[ 4 ] ) ||

isIntersect( x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ], x[ 0 ], y[ 0 ], x[ 4 ], y[ 4 ] ) ||

isIntersect( x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ], x[ 0 ], y[ 0 ], x[ 4 ], y[ 4 ] ) ) {

throw FigException( "Error in Pentagon constructor: pentagon with pair nonadjacent sides interect" );

}

}

size\_t Pentagon::VertexesNumber() const {

return 5;

}

double Pentagon::Area() const {

// Выражение площади математически корректного пятиугольника через

// ориентированные площади трех треугольников

return fabs( getOrArea( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ] ) +

getOrArea( x[ 0 ], y[ 0 ], x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ] ) +

getOrArea( x[ 0 ], y[ 0 ], x[ 3 ], y[ 3 ], x[ 4 ], y[ 4 ] ) );

}

void Pentagon::Print( std::ostream & os ) const {

os << "Pentagon: ";

os << showpos << fixed;

for ( int i = 0; i < 5; i++ ) {

os << "(" << setprecision( 1 ) << x[ i ] <<

", " << setprecision( 1 ) << y[ i ] <<

")";

if ( i < 4 )

os << " ";

}

os << "\n";

os << noshowpos;

}

Pentagon::~Pentagon() { }

pentagon.h

#ifndef PENTAGON\_H

#define PENTAGON\_H

#include"figure.h"

class Pentagon : public Figure {

double x[ 5 ];

double y[ 5 ];

public:

Pentagon();

virtual ~Pentagon();

virtual size\_t VertexesNumber() const;

virtual double Area() const;

virtual void Print( std::ostream & os ) const;

};

#endif /\* PENTAGON\_H \*/

rhombus.cpp

#include<iomanip>

#include"rhombus.h"

Rhombus::Rhombus() : Figure() {

for ( int i = 0; i < 4; i++ ) {

cin >> x[ i ];

cin >> y[ i ];

}

if ( cin.fail() ) {

throw FigException( "Bad coordinate values in Rhombus constructor" );

}

// Проверка математической корректности ромба

// Проверка равенства сторон ромба

double len0 = getLength( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ] );

double len1 = getLength( x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ] );

double len2 = getLength( x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ] );

double len3 = getLength( x[ 3 ], y[ 3 ], x[ 0 ], y[ 0 ] );

if ( ! isDoublesEqual( len0, len1 ) ||

! isDoublesEqual( len1, len2 ) ||

! isDoublesEqual( len2, len3 ) ) {

throw FigException( "Error in Rhombus constructor: rhomb with different side lengths" );

}

// Проверка равенства нулю сторон ромба

if ( isDoubleZero( len0 ) ) {

throw FigException( "Error in Rhombus constructor: rhomb with zero side lengths" );

}

// Проверка несовпадаения противоположных вершин

if ( ( isDoublesEqual( x[ 0 ], x[ 2 ] ) && isDoublesEqual( y[ 0 ], y[ 2 ] ) ) ||

( isDoublesEqual( x[ 1 ], x[ 3 ] ) && isDoublesEqual( y[ 1 ], y[ 3 ] ) ) ) {

throw FigException( "Error in Rhombus constructor: rhomb with coincided vertexes" );

}

}

size\_t Rhombus::VertexesNumber() const {

return 4;

}

double Rhombus::Area() const {

return fabs( getOrArea( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ] ) ) \* 2.0;

}

void Rhombus::Print( std::ostream & os ) const {

os << "Rhombus: ";

os << showpos << fixed;

for ( int i = 0; i < 4; i++ ) {

os << "(" << setprecision( 1 ) << x[ i ] <<

", " << setprecision( 1 ) << y[ i ] <<

")";

if ( i < 3 )

os << " ";

}

os << "\n";

os << noshowpos;

}

Rhombus::~Rhombus() { }

rhombus.h

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include"figure.h"

class Rhombus : public Figure {

double x[ 4 ];

double y[ 4 ];

public:

Rhombus();

virtual ~Rhombus();

virtual size\_t VertexesNumber() const;

virtual double Area() const;

virtual void Print( std::ostream & os ) const;

};

#endif /\* RHOMBUS\_H \*/

trapezoid.cpp

#include<iomanip>

#include"trapezoid.h"

Trapezoid::Trapezoid() : Figure() {

for ( int i = 0; i < 4; i++ ) {

cin >> x[ i ];

cin >> y[ i ];

}

if ( cin.fail() ) {

throw FigException( "Bad coordinate values in Trapezoid constructor" );

}

// Проверка математической корректности трапеции

// Стороны трапеции

double len0 = getLength( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ] );

double len1 = getLength( x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ] );

double len2 = getLength( x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ] );

double len3 = getLength( x[ 3 ], y[ 3 ], x[ 0 ], y[ 0 ] );

// Если есть сторона нулевой длины

if ( isDoubleZero( len0 ) ||

isDoubleZero( len1 ) ||

isDoubleZero( len2 ) ||

isDoubleZero( len3 ) ) {

throw FigException( "Error in Trapezoid constructor: trapezoid with zero side length" );

}

// Если обе пары противоположных сторон не являются параллельными

if ( ! isVectorsParallel( x[ 0 ]-x[ 1 ], y[ 0 ]-y[ 1 ], x[ 3 ]-x[ 2 ], y[ 3 ]-y[ 2 ] ) &&

! isVectorsParallel( x[ 1 ]-x[ 2 ], y[ 1 ]-y[ 2 ], x[ 3 ]-x[ 0 ], y[ 3 ]-y[ 0 ] ) ) {

throw FigException( "Error in Trapezoid constructor: trapeziod without parallel sides" );

}

}

size\_t Trapezoid::VertexesNumber() const {

return 4;

}

double Trapezoid::Area() const {

return fabs( getOrArea( x[ 0 ], y[ 0 ], x[ 1 ], y[ 1 ], x[ 2 ], y[ 2 ] ) +

getOrArea( x[ 0 ], y[ 0 ], x[ 2 ], y[ 2 ], x[ 3 ], y[ 3 ] ) );

}

void Trapezoid::Print( std::ostream & os ) const {

os << "Trapezoid: ";

os << showpos << fixed;

for ( int i = 0; i < 4; i++ ) {

os << "(" << setprecision( 1 ) << x[ i ] <<

", " << setprecision( 1 ) << y[ i ] <<

")";

if ( i < 3 )

os << " ";

}

os << "\n";

os << noshowpos;

}

Trapezoid::~Trapezoid() { }

trapezoid.h

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include"figure.h"

class Trapezoid : public Figure {

double x[ 4 ];

double y[ 4 ];

public:

Trapezoid();

virtual ~Trapezoid();

virtual size\_t VertexesNumber() const;

virtual double Area() const;

virtual void Print( std::ostream & os ) const;

};

#endif /\* TRAPEZOID\_H \*/