**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Изучение основ работы с шаблонами (template) в С++

Студент: Черемисинов Максим (староста)

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Написать программу с базовым классом Figure и производными классами квадрата, прямоугольника с трапеции, которые наследуются от класса Figure. Должна быть возможность вычисления центра фигуры и ее площади при помощи переопределения виртуальных функций родительского класса.

1. **Репозиторий github**

<https://github.com/devepodete/oop_exercise_04>

1. **Описание программы**

Программа состоит из 5 файлов:

1. main.cpp
2. ui.h - заголовочный файл для пользовательского интерфейса
3. ui.cpp - пользовательский интерфейс
4. figures.h - фигуры и их функции
5. cord\_exception.h - исключения

Пользователь вводит фигуру в формате “имя тип”, например, “square double”, после чего вводит координаты. Если координаты введены неверно, то выдается соответствующая ошибка. После задания фигуры можно вызвать для нее функции подсчета площади, центра или вывода координат.

1. **Набор testcases**

**test\_\_01.txt**

aoaoao

qp p [e [

square akak

square int

0 0 z 0

square double

0 0 0 0

square double

0 0 4.5 4.5

area center print

exit

exit

Проверка проверки ввода команд, задания координат квадрата, функций для него.

**test\_\_02.txt**

rectangle int

0 0 0 0 0 0

rectangle int

0 0 0 4 2 5

rectangle int

0 0 0 4.4 2.4 4.4

area

center

print

exit

exit

Проверка проверки координат прямоугольника, обработки нецелых чисел при задании координат фигуры с целочисленными координатами, вывод формул для фигуры.

**test\_\_03.txt**

trapezoid int

0 0 0 3 0 4 0 5

trapezoid double

0 0 1 3 3 3 4 0

area

center

print

o

exit

Проверка проверок для трапеции.

1. **Результаты выполнения тестов.**

Все тесты успешно пройдены, программа выдает правильные ответы.

1. **Листинг программы**

**main.cpp**

/\*

Написать программу по вычислению центра и площади квадрата, прямоугольника или трапеции, которые

задает пользователь, и по выводу их координат. Использовать вместо наследования шаблоны.

Сделать функции, которые принимают на вход кортежи.

Черемисинов Максим

М8О-201Б-18

\*/

#include "cord\_exception.h"

#include "figures.h"

#include "ui.h"

auto main() -> int{

uil::welcome();

uil::try\_tuples();

uil\_command cmd;

while(cmd != UIL\_COMMAND\_EXIT){

cmd = uil::main\_cmd();

}

}

**ui.h**

#ifndef \_\_UI\_H\_\_

#define \_\_UI\_H\_\_

#include <iostream>

using uil\_command = std::string;

const uil\_command UIL\_COMMAND\_EXIT = "exit";

namespace uil{

int check\_syntax(const uil\_command &cmd);

int check\_command(const uil\_command &cmd);

bool is\_number(const std::string &str);

int value\_type();

uil\_command main\_cmd();

template <class T>

void make\_action(const T &figure);

template <class T>

void get\_square\_cords(T &i1, T &i2);

template <class T>

void get\_rectangle\_cords(T &i1, T &i2, T &i3);

template <class T>

void get\_trapezoid\_cords(T &i1, T &i2, T &i3, T &i4);

void try\_tuples();

void welcome();

}

template <class T>

void uil::get\_square\_cords(std::pair<T, T> &i1, std::pair<T, T> &i2){

std::cout << "cords(2): ";

std::cin >> i1 >> i2;

}

template <class T>

void uil::get\_rectangle\_cords(std::pair<T, T> &i1, std::pair<T, T> &i2, std::pair<T, T> &i3){

std::cout << "cords(3): ";

std::cin >> i1 >> i2 >> i3;

}

template <class T>

void uil::get\_trapezoid\_cords(std::pair<T, T> &i1, std::pair<T, T> &i2, std::pair<T, T> &i3, std::pair<T, T> &i4){

std::cout << "cords(4): ";

std::cin >> i1 >> i2 >> i3 >> i4;

}

template <class T>

void uil::make\_action(const T &figure){

while(true){

std::cout << "action: ";

uil\_command command;

std::cin >> command;

int result = uil::check\_command(command);

switch(result){

case -1:

return;

case 0:

//area

std::cout << calculate\_area(figure) << std::endl;

break;

case 1:

//center

std::cout << calculate\_center(figure) << std::endl;

break;

case 2:

//print

print\_cords(figure);

break;

}

}

}

#endif

**figures.h**

#ifndef \_\_FIGURES\_H\_\_

#define \_\_FIGURES\_H\_\_

#include <exception>

#include <type\_traits>

#include <string>

#include <tuple>

#include <cmath>

#include "cord\_exception.h"

#include "ui.h"

template <class T>

bool collinear(const std::pair<T, T> &a, const std::pair<T, T> &b, const std::pair<T, T> &c){

return (a.second-b.second)\*(a.first-c.first) - (a.second-c.second)\*(a.first-b.first) <= 1e-9;

}

template <int>

bool collinear(std::pair<int, int> &a, std::pair<int, int> &b, std::pair<int, int> &c){

return (a.second-b.second)\*(a.first-c.first) == (a.second-c.second)\*(a.first-b.first);

}

template <class T>

bool perpendicular(const std::pair<T, T> &a, const std::pair<T, T> &b, const std::pair<T, T> &c){

using vect = std::pair<T, T>;

vect v1 = b-a;

vect v2 = b-c;

T scalarProduct = v1.first\*v2.first + v1.second\*v2.second;

if(scalarProduct <= 1e-9 && v1.first\*v2.first + v1.second\*v2.second >= -1e-9){

return true;

}else{

return false;

}

}

template <class T>

class square{

public:

using type = T;

using point = std::pair<T, T>;

point a, b;

square(){}

square(point &p1, point &p2){

if(p1 == p2){

throw WrongPointsException<T>(p1, p2);

}else{

if(p1.first > p2.first){

a = p2;

b = p1;

}else{

a = p1;

b = p2;

}

}

}

};

template <class T>

class rectangle{

public:

using type = T;

using point = std::pair<T, T>;

point a, b, c;

rectangle(){}

rectangle(point &p1, point &p2, point &p3){

if(p1 == p2 || p1 == p3 || p2 == p3 || collinear(p1, p2, p3) || !(perpendicular(p1, p2, p3))){

throw WrongPointsException<T>(p1, p2, p3);

}else{

a = p1;

b = p2;

c = p3;

}

}

};

template <class T>

class trapezoid{

public:

using type = T;

using point = std::pair<T, T>;

point a, b, c, d;

trapezoid(){}

trapezoid(point &p1, point &p2, point &p3, point &p4){

if(p1 == p2 || p1 == p3 || p1 == p4 || p2 == p3 || p2 == p4 || p3 == p4 ||

collinear(p1, p2, p3) || collinear(p1, p2, p4) || collinear(p1, p3, p4) ||

collinear(p2, p3, p4)){

throw WrongPointsException<T>(p1, p2, p3, p4);

}else{

a = p1;

b = p2;

c = p3;

d = p4;

}

}

};

template <class T>

struct is\_square{

static const bool value = false;

};

template <class T>

struct is\_square<square<T>>{

static const bool value = true;

};

template <class T>

struct is\_rectangle{

static const bool value = false;

};

template <class T>

struct is\_rectangle<rectangle<T>>{

static const bool value = true;

};

template <class T>

struct is\_trapezoid{

static const bool value = false;

};

template <class T>

struct is\_trapezoid<trapezoid<T>>{

static const bool value = true;

};

template <class T>

using square\_center = typename std::enable\_if<is\_square<T>::value, std::pair<typename T::type, typename T::type>>::type;

template <class T>

using rectangle\_center = typename std::enable\_if<is\_rectangle<T>::value, std::pair<typename T::type, typename T::type>>::type;

template <class T>

using trapezoid\_center = typename std::enable\_if<is\_trapezoid<T>::value, std::pair<typename T::type, typename T::type>>::type;

template <class T>

using square\_cords = typename std::enable\_if<is\_square<T>::value, void>::type;

template <class T>

using rectangle\_cords = typename std::enable\_if<is\_rectangle<T>::value, void>::type;

template <class T>

using trapezoid\_cords = typename std::enable\_if<is\_trapezoid<T>::value, void>::type;

template <class T>

using square\_area = typename std::enable\_if<is\_square<T>::value, typename T::type>::type;

template <class T>

using rectangle\_area = typename std::enable\_if<is\_rectangle<T>::value, typename T::type>::type;

template <class T>

using trapezoid\_area = typename std::enable\_if<is\_trapezoid<T>::value, typename T::type>::type;

template <class T>

square\_center<T> calculate\_center(const T &sq){

typename T::type x, y;

x = sq.a.first + (sq.b.first-sq.a.first)/2;

if(sq.a.second > sq.b.second){

y = sq.b.second + (sq.a.second-sq.b.second)/2;

}else{

y = sq.a.second + (sq.b.second-sq.a.second)/2;

}

return {x, y};

}

template <class T>

rectangle\_center<T> calculate\_center(const T &rec){

typename T::type x\_center, y\_center, x1, x3, y1, y3;

x1 = rec.a.first;

x3 = rec.c.first;

y1 = rec.a.second;

y3 = rec.c.second;

x\_center = x1 + (x3-x1)/2;

if(y1 > y3){

y\_center = y3 + (y1-y3)/2;

}else{

y\_center = y1 + (y3-y1)/2;

}

return {x\_center, y\_center};

}

template <class T>

trapezoid\_center<T> calculate\_center(const T &tr){

typename T::type x1, x2, x3, x4, y1, y2, y3, y4, x\_center, y\_center;

x1 = tr.a.first;

x2 = tr.b.first;

x3 = tr.c.first;

x4 = tr.d.first;

y1 = tr.a.second;

y2 = tr.b.second;

y3 = tr.c.second;

y4 = tr.d.second;

x\_center = (x1+x2+x3+x4)/4;

y\_center = (y1+y2+y3+y4)/4;

return {x\_center, y\_center};

}

template <class T>

std::pair<T, T> calculate\_center(const std::tuple<std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

square<T> sq(p1, p2);

return calculate\_center(sq);

}

template <class T>

std::pair<T, T> calculate\_center(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2, p3;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

p3 = std::get<2>(tp);

rectangle<T> rec(p1, p2, p3);

return calculate\_center(rec);

}

template <class T>

std::pair<T, T> calculate\_center(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2, p3, p4;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

p3 = std::get<2>(tp);

p4 = std::get<3>(tp);

trapezoid<T> tr(p1, p2, p3, p4);

return calculate\_center(tr);

}

template <class T>

square\_cords<T> print\_cords(const T &sq){

std::cout << "Cords of square: " << sq.a << ", " << sq.b << std::endl;

}

template <class T>

rectangle\_cords<T> print\_cords(const T &rec){

std::cout << "Cords of rectangle: " << rec.a << ", " << rec.b << ", " << rec.c << std::endl;

}

template <class T>

trapezoid\_cords<T> print\_cords(const T &tr){

std::cout << "Cords of trapezoid: " << tr.a << ", " << tr.b << ", " << tr.c << ", " << tr.d << std::endl;

}

template <class T>

void print\_cords(const std::tuple<std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

square<T> sq(p1, p2);

std::cout << "Tuple. ";

print\_cords(sq);

}

template <class T>

void print\_cords(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2, p3;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

p3 = std::get<2>(tp);

rectangle<T> rec(p1, p2, p3);

std::cout << "Tuple. ";

print\_cords(rec);

}

template <class T>

void print\_cords(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2, p3, p4;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

p3 = std::get<2>(tp);

p4 = std::get<3>(tp);

trapezoid<T> tr(p1, p2, p3, p4);

std::cout << "Tuple. ";

print\_cords(tr);

}

template <class T>

square\_area<T> calculate\_area(const T &sq){

typename T::type x1, x2, y1, y2;

x1 = sq.a.first;

x2 = sq.b.first;

y1 = sq.a.second;

y2 = sq.b.second;

return ((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2))/2;

}

template <class T>

rectangle\_area<T> calculate\_area(const T &rec){

typename T::type x1, x2, x3, y1, y2, y3;

x1 = rec.a.first;

x2 = rec.b.first;

x3 = rec.c.first;

y1 = rec.a.second;

y2 = rec.b.second;

y3 = rec.c.second;

auto side1 = std::sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));

auto side2 = std::sqrt((x2-x3)\*(x2-x3)+(y2-y3)\*(y2-y3));

return side1\*side2;

}

template <class T>

trapezoid\_area<T> calculate\_area(const T &tr){

typename T::type x1, x2, x3, x4, y1, y2, y3, y4;

x1 = tr.a.first;

x2 = tr.b.first;

x3 = tr.c.first;

x4 = tr.d.first;

y1 = tr.a.second;

y2 = tr.b.second;

y3 = tr.c.second;

y4 = tr.d.second;

auto area = (x1\*y2-x2\*y1)+(x2\*y3-x3\*y2)+(x3\*y4-x4\*y3);

area /= 2;

if(area < 0) return -area;

return area;

}

template <class T>

T calculate\_area(const std::tuple<std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

square<T> sq(p1, p2);

return calculate\_area(sq);

}

template <class T>

T calculate\_area(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2, p3;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

p3 = std::get<2>(tp);

rectangle<T> rec(p1, p2, p3);

return calculate\_area(rec);

}

template <class T>

T calculate\_area(const std::tuple<std::pair<T, T>, std::pair<T, T>, std::pair<T, T>, std::pair<T, T>> &tp){

std::pair<T,T> p1, p2, p3, p4;

p1 = std::get<0>(tp);

p2 = std::get<1>(tp);

p3 = std::get<2>(tp);

p4 = std::get<3>(tp);

trapezoid<T> tr(p1, p2, p3, p4);

return calculate\_area(tr);

}

template <class T>

std::istream& operator>>(std::istream &is, std::pair<T, T> &p){

std::string checker;

is >> checker;

if(uil::is\_number(checker) == false){

throw WrongValueException(checker);

}

p.first = static\_cast<T>(std::stod(checker));

is >> checker;

if(uil::is\_number(checker) == false){

throw WrongValueException(checker);

}

p.second = static\_cast<T>(std::stod(checker));

return is;

}

template <class T>

std::ostream& operator<<(std::ostream &o, const std::pair<T, T> &p){

o << "<" << p.first << ", " << p.second << ">";

return o;

}

template <class T>

std::ostream& operator<<(std::ostream &o, const std::pair<T, T> &&p){

o << "<" << p.first << ", " << p.second << ">";

return o;

}

template <class T>

bool operator==(const std::pair<T, T> &a, const std::pair<T, T> &b){

return (a.first == b.first) && (a.second == b.second);

}

template <class T>

std::pair<T, T> operator-(const std::pair<T, T> &p1, const std::pair<T, T> &p2){

return {p1.first-p2.first, p1.second-p2.second};

}

#endif

**cords\_exception.h**

#ifndef \_\_CORD\_EXCEPTION\_H\_\_

#define \_\_CORD\_EXCEPTION\_H\_\_

#include <exception>

#include <string>

template <class T>

class WrongPointsException: public std::exception{

private:

short sq = 0;

short rec = 1;

short trap = 2;

std::pair<T, T> a, b, c, d;

short type;

public:

WrongPointsException(const std::pair<T, T> &x, const std::pair<T,T> &y){

a = x;

b = y;

type = sq;

}

WrongPointsException(const std::pair<T, T> &x, const std::pair<T,T> &y,

const std::pair<T, T> &z){

a = x;

b = y;

c = z;

type = rec;

}

WrongPointsException(const std::pair<T, T> &x, const std::pair<T,T> &y,

const std::pair<T, T> &z, const std::pair<T, T> &w){

a = x;

b = y;

c = z;

d = w;

type = trap;

}

const char \*what()const noexcept override{

if(type == sq){

static std::string str;

str = "Wrong points: ";

str += "<";

str += std::to\_string(a.first);

str += ", ";

str += std::to\_string(a.second);

str += ">, ";

str += "<";

str += std::to\_string(b.first);

str += ", ";

str += std::to\_string(b.second);

str += ">";

return str.c\_str();

}else if (type == rec){

static std::string str("Wrong points: ");

str += "<";

str += std::to\_string(a.first);

str += ", ";

str += std::to\_string(a.second);

str += ">, ";

str += "<";

str += std::to\_string(b.first);

str += ", ";

str += std::to\_string(b.second);

str += ">, ";

str += "<";

str += std::to\_string(c.first);

str += ", ";

str += std::to\_string(c.second);

str += "> ";

return str.c\_str();

}else{

static std::string str;

str = "Wrong points: ";

str += "<";

str += std::to\_string(a.first);

str += ", ";

str += std::to\_string(a.second);

str += ">, ";

str += "<";

str += std::to\_string(b.first);

str += ", ";

str += std::to\_string(b.second);

str += ">, ";

str += "<";

str += std::to\_string(c.first);

str += ", ";

str += std::to\_string(c.second);

str += ">, ";

str += "<";

str += std::to\_string(d.first);

str += ", ";

str += std::to\_string(d.second);

str += ">";

return str.c\_str();

}

}

};

class WrongValueException: public std::exception{

private:

std::string str;

public:

WrongValueException(const std::string &a): str(a){}

const char \*what()const noexcept override{

static std::string error;

error = "Wrong value: ";

error += str;

return error.c\_str();

}

};

#endif

**ui.cpp**

#include <iostream>

#include <string>

#include <vector>

#include <tuple>

#include "figures.h"

#include "ui.h"

#include "cord\_exception.h"

uil\_command uil::main\_cmd(){

std::cout << "> ";

static uil\_command cmd;

std::cin >> cmd;

using pair\_int = std::pair<int, int>;

using pair\_double = std::pair<double, double>;

switch(uil::check\_syntax(cmd)){

case 0:

break;

case 1:{

//square

int type = uil::value\_type();

if(type == -1){

std::cout << "Error: incorrect value type\n";

break;

}

switch(type){

case 0:{

//int

try{

pair\_int i1, i2;

uil::get\_square\_cords(i1, i2);

square<int> sq\_int(i1, i2);

uil::make\_action(sq\_int);

}catch(WrongPointsException<int> &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}catch(WrongValueException &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}

break;

}

case 1:{

//double

try{

pair\_double i1, i2;

uil::get\_square\_cords(i1, i2);

square<double> sq\_int(i1, i2);

uil::make\_action(sq\_int);

}catch(WrongPointsException<double> &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}catch(WrongValueException &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}

break;

}

}

break;

}

case 2:{

//rectangle

int type = uil::value\_type();

if(type == -1){

std::cout << "Error: incorrect value type\n";

break;

}

switch(type){

case 0:{

//int

try{

pair\_int i1, i2, i3;

uil::get\_rectangle\_cords(i1, i2, i3);

rectangle<int> rec\_int(i1, i2, i3);

uil::make\_action(rec\_int);

}catch(WrongPointsException<int> &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}catch(WrongValueException &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}

break;

}

case 1:{

//double

try{

pair\_double i1, i2, i3;

uil::get\_rectangle\_cords(i1, i2, i3);

rectangle<double> rec\_double(i1, i2, i3);

uil::make\_action(rec\_double);

}catch(WrongPointsException<double> &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}catch(WrongValueException &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}

break;

}

}

break;

}

case 3:{

//trapezoid

int type = uil::value\_type();

if(type == -1){

std::cout << "Error: incorrect value type\n";

break;

}

switch(type){

case 0:{

//int

try{

pair\_int i1, i2, i3, i4;

uil::get\_trapezoid\_cords(i1, i2, i3, i4);

trapezoid<int> tr\_int(i1, i2, i3, i4);

uil::make\_action(tr\_int);

}catch(WrongPointsException<int> &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}catch(WrongValueException &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}

break;

}

case 1:{

//double

try{

pair\_double i1, i2, i3, i4;

uil::get\_trapezoid\_cords(i1, i2, i3, i4);

trapezoid<double> tr\_double(i1, i2, i3, i4);

uil::make\_action(tr\_double);

}catch(WrongPointsException<double> &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}catch(WrongValueException &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}

break;

}

}

break;

}

case -1:

std::cout << "Error: incorrect command\n";

break;

}

return cmd;

}

int uil::check\_syntax(const uil\_command &cmd){

std::vector<std::string> commands(0, std::string());

commands.push\_back("exit");

commands.push\_back("square");

commands.push\_back("rectangle");

commands.push\_back("trapezoid");

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

if(cmd == commands[i]) return i;

}

return -1;

}

int uil::check\_command(const uil\_command &cmd){

std::vector<std::string> commands(0, std::string());

commands.push\_back("area");

commands.push\_back("center");

commands.push\_back("print");

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

if(cmd == commands[i]) return i;

}

return -1;

}

int uil::value\_type(){

std::cout << "> ";

static std::string type;

std::vector<std::string> types(0, std::string());

types.push\_back("int");

types.push\_back("double");

std::cin >> type;

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

if(type == types[i]) return i;

}

return -1;

}

bool uil::is\_number(const std::string &str){

short points = 0;

if(str.empty()) return false;

if(str[0] == '.' || str[str.size()-1] == '.') return false;

for(auto ptr = str.begin(); ptr != str.end(); ptr++){

if(\*ptr == '.'){

points++;

continue;

}

if(\*ptr < '0' || \*ptr > '9') return false;

}

if(points >= 2) return false;

return true;

}

void uil::welcome(){

std::cout << "Welcome to the UI!\n";

std::cout << "------------------------------------------\n";

std::cout << "To create a figure type [figure name]\n";

std::cout << "where [figure name] could be square, rectangle or trapezoid.\n";

std::cout << "------------------------------------------\n";

std::cout << "Add the coordinates.\n";

std::cout << "------------------------------------------\n";

std::cout << "Available commands for figure: area, center, print\n";

std::cout << "To finish work with figure type anything except available commands\n";

std::cout << "------------------------------------------\n";

std::cout << "To exit the program type \"exit\"\n";

}

void uil::try\_tuples(){

std::cout << "\nExample with tuple usage:\n\n";

using int\_type = int;

using double\_type = double;

using int\_pair = std::pair<int\_type, int\_type>;

using double\_pair = std::pair<double\_type, double\_type>;

std::tuple<double\_pair, double\_pair> tuple\_double({0,0}, {3.5,3.5});

print\_cords(tuple\_double);

std::cout << calculate\_area(tuple\_double) << std::endl;

std::cout << calculate\_center(tuple\_double) << std::endl;

try{

std::tuple<int\_pair, int\_pair, int\_pair> tuple\_int({0,0}, {0, 5}, {1, 9});

calculate\_area(tuple\_int);

}catch(WrongPointsException<int\_type> &ex){

std::cout << "Exception in runtime: " << ex.what() << std::endl;

}

std::cout << "\n";

}

1. **Вывод**

Программа производит проверки для корректности ввода координат фигур, но не умеет генерировать другие точки по нескольким заданным. Вычисляет площади и центры, делает проверки ввода координат и команд.

**Список литературы**

1. Шилдт, Герберт. С++: базовый курс, 3-е изд. : Пер. с англ. - М. : ООО “И.Д. Вильямс”, 2018. - 624 с. : ил. - Парал. тит. англ.
2. Справочник по языку С++ [Электронный ресурс]. URL: [http://www.cplusplus.com/reference](http://www.cplusplus.com/reference/deque/) (дата обращения: 25.10.2019).
3. Справочник по языку С++ [Электронный ресурс]. URL:

<https://en.cppreference.com/w/> (дата обращения: 27.10.2019).