**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Основы метапрограммирования

Студент: Айрапетова Евгения Ашотовна

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Создать программу, которая позволяет:

* Создает набор фигур согласно варианту задания (как минимум по одной фигуре каждого типа с координатами типа int и координатами типа double).
* Сохраняет фигуры в std::tuple
* Печатает на экран содержимое std::tuple с помощью шаблонной функции print.
* Вычисляет суммарную площадь фигур в std::tuple и выводит значение на экран.

При реализации шаблонных функций допускается использование вспомогательных шаблонов std::enable\_if, std::tuple\_size, std::is\_same.

Вариант 34: квадрат, прямоугольник, трапеция.

.

1. Описание программы

На вход программе поступают координаты фигур типа int и double. Программой проверяется правильность введенных данных и фигуры сохраняются в кортеж. Если с пользователь ввел все координаты верно, то на экран выводятся координаты вершин всех фигур в кортеже, а также общая площадь этих фигур.

1. Набор тестов

Тест 1:  
Квадрат с координатами типа int:

0 0

1 0

1 1

0 1

Квадрат с координатами типа double:

0 0

1.2 0

1.2 1.2

0 1.2

Прямоугольник с координатами типа int:

0 0

2 0

2 1

0 1

Прямоугольник с координатами типа double:

0.2 0.2

2.4 0.2

2.4 2.3

0.2 2.3

Трапеция с координатами типа int:

4 0

2 2

-2 2

-4 0

Трапеция с координатами типа double:

4.1 0.1

2.1 2.1

-2.1 2.1

-4.1 0.1

Тест 2:  
Квадрат с координатами типа int:

3 4

6 4

6 7

3 7

Квадрат с координатами типа double:

0 0

2.3 0

2.3 2.3

0 2.3

Прямоугольник с координатами типа int:

3 4

7 4

7 6

3 6

Прямоугольник с координатами типа double:

2 2

3.1 2

3.1 5.6

2 5.6

Трапеция с координатами типа int:

5 0

2 2

-2 2

-5 0

Трапеция с координатами типа double:

5.1 1.1

2 2

-2 2

-5.1 1.1

1. Результаты выполнения тестов

Тест 1:

Your figures:

(0,0)

(1,0)

(1,1)

(0,1)

(0,0)

(1.2,0)

(1.2,1.2)

(0,1.2)

(0,0)

(2,0)

(2,1)

(0,1)

(0.2,0.2)

(2.4,0.2)

(2.4,2.3)

(0.2,2.3)

(4,0)

(2,2)

(-2,2)

(-4,0)

(4.1,0.1)

(2.1,2.1)

(-2.1,2.1)

(-4.1,0.1)

Тест 2:

Your figures:

(3,4)

(6,4)

(6,7)

(3,7)

(0,0)

(2.3,0)

(2.3,2.3)

(0,2.3)

(3,4)

(7,4)

(7,6)

(3,6)

(2,2)

(3.1,2)

(3.1,5.6)

(2,5.6)

(5,0)

(2,2)

(-2,2)

(-5,0)

(5.1,1.1)

(2,2)

(-2,2)

(-5.1,1.1)

Total area: 46.64

1. Листинг программы

// Айрапетова Евгения М8о-206Б-19

#include <iostream>

#include <tuple>

using namespace std;

template <class T>

using vertex\_t = pair<T, T>;

template <class T>

istream& operator>> (istream& input, vertex\_t<T>& v) {

input >> v.first >> v.second;

return input;

}

template<class T>

ostream& operator<< (ostream& output, const vertex\_t<T> v) {

output << "(" << v.first << "," << v.second << ")" << '\n';

return output;

}

template <class T>

class Square {

public:

vertex\_t<T> a, b, c, d;

};

template <class T>

class Rectangle {

public:

vertex\_t<T> a, b, c, d;

};

template <class T>

class Trapeze {

public:

vertex\_t<T> a, b, c, d;

};

template <class T>

double area(const Square<T>& sq) {

return abs(sq.a.first \* sq.b.second + sq.b.first \* sq.c.second + sq.c.first \* sq.d.second + sq.d.first \* sq.a.second

- sq.b.first \* sq.a.second - sq.c.first \* sq.b.second - sq.d.first \* sq.c.second - sq.a.first \* sq.d.second) / 2;

}

template <class T>

double area(const Rectangle<T>& re) {

return abs(re.a.first \* re.b.second + re.b.first \* re.c.second + re.c.first \* re.d.second + re.d.first \* re.a.second

- re.b.first \* re.a.second - re.c.first \* re.b.second - re.d.first \* re.c.second - re.a.first \* re.d.second) / 2;

}

template <class T>

double area(const Trapeze<T>& tr) {

return abs(tr.a.first \* tr.b.second + tr.b.first \* tr.c.second + tr.c.first \* tr.d.second + tr.d.first \* tr.a.second

- tr.b.first \* tr.a.second - tr.c.first \* tr.b.second - tr.d.first \* tr.c.second - tr.a.first \* tr.d.second) / 2;

}

template <class T>

istream& operator>> (istream& input, Square<T>& sq) {

input >> sq.a >> sq.b >> sq.c >> sq.d;

return input;

}

template <class T>

istream& operator>> (istream& input, Rectangle<T>& re) {

input >> re.a >> re.b >> re.c >> re.d;

return input;

}

template <class T>

istream& operator>> (istream& input, Trapeze<T>& tr) {

input >> tr.a >> tr.b >> tr.c >> tr.d;

return input;

}

template <class T>

void print(const Square<T>& sq) {

cout << sq.a << ' ' << sq.b << ' ' << sq.c << ' ' << sq.d << '\n';

return;

}

template <class T>

void print(const Rectangle<T>& re) {

cout << re.a << ' ' << re.b << ' ' << re.c << ' ' << re.d << '\n';

return;

}

template <class T>

void print(const Trapeze<T>& tr) {

cout << tr.a << ' ' << tr.b << ' ' << tr.c << ' ' << tr.d << '\n';

return;

}

template <class... Elements>

void print(const tuple<Elements...> t) {

if (tuple\_size<tuple<Elements...>>::value != 6) {

cerr << "Incorrect number of figures entered.\n";

return;

}

print(get<0>(t));

print(get<1>(t));

print(get<2>(t));

print(get<3>(t));

print(get<4>(t));

print(get<5>(t));

return;

}

template<class... Elements>

void area(const tuple<Elements...> t) {

if (tuple\_size<tuple<Elements...>>::value != 6) {

cerr << "Incorrect number of figures entered.\n";

return;

}

double sum\_area = 0;

sum\_area += area(get<0>(t));

sum\_area += area(get<1>(t));

sum\_area += area(get<2>(t));

sum\_area += area(get<3>(t));

sum\_area += area(get<4>(t));

sum\_area += area(get<5>(t));

cout << "\nTotal area: " << sum\_area;

return;

}

template<class T>

double lenght(vertex\_t<T>& a, vertex\_t<T>& b) {

return sqrt((b.first - a.first) \* (b.first - a.first) + ((b.second - a.second) \* (b.second - a.second)));

}

template<class T>

int CorrectInput(pair<T, T>& a, pair<T, T>& b, pair<T, T>& c, pair<T, T>& d, int f) {

if ((a.first == c.first && a.second == c.second) || (b.first == d.first && b.second == d.second)) {

return 1;

}

double len\_ab = lenght(a, b);

double len\_bc = lenght(b, c);

double len\_cd = lenght(c, d);

double len\_da = lenght(d, a);

double len\_ac = lenght(a, c);

double len\_bd = lenght(b, d);

if (f == 1) {

if (abs(len\_cd - len\_ab) < numeric\_limits<double>::epsilon() &&

abs(len\_cd - len\_bc) < numeric\_limits<double>::epsilon() &&

abs(len\_cd - len\_da) < numeric\_limits<double>::epsilon() &&

abs(len\_bd - len\_ac) < numeric\_limits<double>::epsilon()) {

return 0;

}

return 1;

}

if (f == 2) {

if (abs(len\_ab - len\_cd) < numeric\_limits<double>::epsilon() &&

abs(len\_bc - len\_da) < numeric\_limits<double>::epsilon() &&

abs(len\_ac - len\_bd) < numeric\_limits<double>::epsilon()) {

return 0;

}

return 1;

}

if (f == 3) {

if (abs(len\_ab - len\_cd) < numeric\_limits<double>::epsilon() &&

abs(len\_ac - len\_bd) < numeric\_limits<double>::epsilon()) {

return 0;

}

return 1;

}

return 0;

}

int main() {

pair<int, int> A\_int, B\_int, C\_int, D\_int;

pair<double, double> A\_dob, B\_dob, C\_dob, D\_dob;

cout << "Enter squares of integer:" << '\n';

cin >> A\_int >> B\_int >> C\_int >> D\_int;

int f = CorrectInput(A\_int, B\_int, C\_int, D\_int, 1);

if (f == 1) { cerr << "Error input: square"; return -1; }

Square<int> s\_i{ A\_int,B\_int,C\_int,D\_int };

cout << "Enter squares of double:" << '\n';

cin >> A\_dob >> B\_dob >> C\_dob >> D\_dob;

f = CorrectInput(A\_dob, B\_dob, C\_dob, D\_dob, 1);

if (f == 1) { cerr << "Error input: square"; return -1; }

Square<double> s\_d{ A\_dob,B\_dob,C\_dob,D\_dob };

cout << "Enter rectangle of integer:" << '\n';

cin >> A\_int >> B\_int >> C\_int >> D\_int;

f = CorrectInput(A\_int, B\_int, C\_int, D\_int, 2);

if (f == 1) { cerr << "Error input: rectangle"; return -1; }

Rectangle<int> r\_i{ A\_int,B\_int,C\_int,D\_int };

cout << "Enter rectangle of double:" << '\n';

cin >> A\_dob >> B\_dob >> C\_dob >> D\_dob;

f = CorrectInput(A\_dob, B\_dob, C\_dob, D\_dob, 2);

if (f == 1) { cerr << "Error input: rectangle"; return -1; }

Rectangle<double> r\_d{ A\_dob,B\_dob,C\_dob,D\_dob };

cout << "Enter trapeze of integer:" << '\n';

cin >> A\_int >> B\_int >> C\_int >> D\_int;

f = CorrectInput(A\_int, B\_int, C\_int, D\_int, 3);

if (f == 1) { cerr << "Error input: trapeze"; return -1; }

Trapeze<int> t\_i{ A\_int,B\_int,C\_int,D\_int };

cout << "Enter trapeze of double:" << '\n';

cin >> A\_dob >> B\_dob >> C\_dob >> D\_dob;

f = CorrectInput(A\_dob, B\_dob, C\_dob, D\_dob, 3);

if (f == 1) { cerr << "Error input: trapeze"; return -1; }

Trapeze<double> t\_d{ A\_dob,B\_dob,C\_dob,D\_dob };

auto tup = make\_tuple(s\_i, s\_d, r\_i, r\_d, t\_i, t\_d);

cout << "\nYour figures:\n";

print(tup);

area(tup);

return 0;

}

1. Вывод:

В процессе выполнения данной лабораторной работы я научилась создавать и использовать кортежи, а также работать с шаблонными функциями и классами.

Список литературы:

1. Кортежи std::tuple в C++ [Электронный ресурс]. URL: <https://qosys.info/tuple-cpp> (дата обращения: 4.11.2020)
2. Работа с кортежами С++ (std::tuple). Функции foreach, map и call [Электронный ресурс]. URL: <https://habr.com/ru/post/318236/> (дата обращения: 5.11.2020)
3. Шаблоны функций в C++ [Электронный ресурс]. URL: <http://cppstudio.com/post/5165/> (дата обращения: 5.11.2020)