**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: Основы метапрограммирования

Студент: Хренов Геннадий

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения. Для хранения координат фигур необходимо использовать шаблон std::pair. Создать набор шаблонов, создающих функции, реализующие: вычисление геометрического центра фигуры,вывод в стандартный поток вывода std::cout координат вершин фигуры, вычисление площади фигуры.

1. Описание программы

Шаблоны фигур состоят из 4 точек, содержащих координаты по x и y. Шаблоны функций используются для того, чтобы работать с данными разных типов. Для каждой фигуры определены функции вывода координат, нахождения площади, нахождения геометрического центра. Шаблоны позволяют нам вводить фигуры с координатами любого типа, при этом все функции также будут работать корректно. Также была предусмотрена работа с tuples. Tuple заполняется 4 координатами, а используя свойства фигур, можно определить, это за фигура, выполнить функции, предназначенные для неё. Кроме этого, в программе есть контроль за входными данными, который сообщает о некорректно введенных данных, если такие имеются.

1. Набор testcases

а)SI 1 1 4 4

б)RD 1 1 10 3

в)TI 1 1 3 3 5 3 7 1

г)TUBD 1.5 1.5 1.5 4.5 3.0 4.5 3.0 1.5

д)TUBI 1 1 2 2 3 3 4 4

1. Результаты выполнения тестов.

а) создание квадрата

SI 1 1 4 4

Square: A(1; 1) B(1; 4) C(4; 4) D(4; 1)

center is (2; 2)

S = 9

б) создание прямоугольника

RD 1 1 10 3

Rectange: A(1; 1) B(1; 3) C(10; 3) D(10; 1)

center is (5.5; 2)

S = 18

в) создание трапеции

TI 1 1 3 3 5 3 7 1

Trape: A(1; 1) B(3; 3) C(5; 3) D(7; 1)

center is (4; 2)

S = 8

г) tuple должен интерпретироваться как прямоугольник

TUBD 1.5 1.5 1.5 4.5 3.0 4.5 3.0 1.5

tuple is

Rectange: A(1.5; 1.5) B(1.5; 4.5) C(3; 4.5) D(3; 1.5)

center is (2.25; 3)

S = 4.5

д) неверные данные

TUBI 1 1 2 2 3 3 4 4

wrong input

1. Листинг программы

//Хренов Геннадий М8О-207Б ЛР №4

// Шаблоны классов квадрат прямоугольник трапеция

// Для каждой фигуры определены функции : площадь, геометрический центр, координаты

// предусмотрена работа с tuples

#include<iostream>

#include <math.h>

#include <utility>

#include <tuple>

template<class T>

struct Square {

using type = T;

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

vertex A, B, C, D;

Square(T x1, T y1, T x2, T y2) :

A(x1, y1), B(x1, y2), C(x2, y2), D(x2, y1)

{

}

};

template<class T>

struct Rectan {

using type = T;

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

vertex A, B, C, D;

Rectan(T x1, T y1, T x2, T y2) :

A(x1, y1), B(x1, y2), C(x2, y2), D(x2, y1)

{

}

};

template<class T>

struct Trape {

using type = T;

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

vertex A, B, C, D;

Trape(T x1, T y1, T x2, T y2, T x3, T y3, T x4, T y4) :

A(x1, y1), B(x2, y2), C(x3, y3), D(x4, y4)

{

}

};

template <class T>

struct IsSquare {

static constexpr bool value = false;

};

template <template <class...> class F, class T>

struct IsSquare<F<T>> {

static constexpr bool value = std::is\_same<F<T>, Square<T>>::value;

};

template <class T>

struct IsRectan {

static constexpr bool value = false;

};

template <template <class...> class F, class T>

struct IsRectan<F<T>> {

static constexpr bool value = std::is\_same<F<T>, Rectan<T>>::value;

};

template <class T>

struct IsTrape {

static constexpr bool value = false;

};

template <template <class...> class F, class T>

struct IsTrape<F<T>> {

static constexpr bool value = std::is\_same<F<T>, Trape<T>>::value;

};

template <class T>

typename std::enable\_if<IsSquare<T>::value, typename T::type>::type

Sq(const T& r) {

typename T::type AB = static\_cast<typename T::type>(

sqrt(pow(r.B.first - r.A.first,2) + pow(r.B.second - r.A.second,2))

);

typename T::type BC = static\_cast<typename T::type>(

sqrt(pow(r.C.first - r.B.first,2) + pow(r.C.second - r.B.second,2))

);

return AB \* BC;

}

template <class T>

typename std::enable\_if<IsSquare<T>::value, std::pair<typename T::type, typename T::type>>::type

GCF(const T& r) {

typename T::type x = (r.A.first + r.B.first + r.C.first + r.D.first) / 4;

typename T::type y = (r.A.second + r.B.second + r.C.second + r.D.second) / 4;

return std::make\_pair(x,y);

}

template <class T>

void Print(const Square<T>& r) {

std::cout << "Square: A(" << r.A.first << "; " << r.A.second

<< ") B(" << r.B.first << "; " << r.B.second

<< ") C(" << r.C.first << "; " << r.C.second

<< ") D(" << r.D.first << "; " << r.D.second << ")\n";

}

template <class T>

typename std::enable\_if<IsRectan<T>::value, typename T::type>::type

Sq(const T& r) {

typename T::type AB = static\_cast<typename T::type>(

sqrt(pow(r.B.first - r.A.first,2) + pow(r.B.second - r.A.second,2))

);

typename T::type BC = static\_cast<typename T::type>(

sqrt(pow(r.C.first - r.B.first,2) + pow(r.C.second - r.C.second,2))

);

return AB \* BC;

}

template <class T>

typename std::enable\_if<IsRectan<T>::value, std::pair<typename T::type, typename T::type>>::type

GCF(const T& r) {

typename T::type x = (r.A.first + r.B.first + r.C.first + r.D.first) / 4;

typename T::type y = (r.A.second + r.B.second + r.C.second + r.D.second) / 4;

return std::make\_pair(x,y);

}

template <class T>

void Print(const Rectan<T>& r) {

std::cout << "Rectange: A(" << r.A.first << "; " << r.A.second

<< ") B(" << r.B.first << "; " << r.B.second

<< ") C(" << r.C.first << "; " << r.C.second

<< ") D(" << r.D.first << "; " << r.D.second << ")\n";

}

template <class T>

typename std::enable\_if<IsTrape<T>::value, typename T::type>::type

Sq(const T& r) {

typename T::type AD = static\_cast<typename T::type>(

sqrt(pow(r.D.first - r.A.first,2) + pow(r.D.second - r.A.second,2))

);

typename T::type BC = static\_cast<typename T::type>(

sqrt(pow(r.C.first - r.B.first,2) + pow(r.C.second - r.B.second,2))

);

double AB = sqrt(pow(r.B.first - r.A.first,2) + pow(r.B.second - r.A.second,2));

double H = sqrt(pow(AB,2) - pow((AD - BC)/2,2));

return (AD + BC) / 2 \* H;

}

template <class T>

typename std::enable\_if<IsTrape<T>::value, std::pair<typename T::type, typename T::type>>::type

GCF(const T& r) {

typename T::type x = (r.A.first + r.B.first + r.C.first + r.D.first) / 4;

typename T::type y = (r.A.second + r.B.second + r.C.second + r.D.second) / 4;

return std::make\_pair(x,y);

}

template <class T>

void Print(const Trape<T>& r) {

std::cout << "Trape: A(" << r.A.first << "; " << r.A.second

<< ") B(" << r.B.first << "; " << r.B.second

<< ") C(" << r.C.first << "; " << r.C.second

<< ") D(" << r.D.first << "; " << r.D.second << ")\n";

}

template <class T>

T tub(std::tuple<std::pair<T, T>,std::pair<T, T>,std::pair<T, T>,std::pair<T, T>> &tub){

if(pow(std::get<2>(tub).first - std::get<1>(tub).first,2) + pow(std::get<2>(tub).second - std::get<1>(tub).second,2) ==

pow(std::get<3>(tub).first - std::get<0>(tub).first,2) + pow(std::get<3>(tub).second - std::get<0>(tub).second,2) ) {

if (pow(std::get<1>(tub).first - std::get<0>(tub).first,2) + pow(std::get<1>(tub).second - std::get<0>(tub).second,2) !=

pow(std::get<2>(tub).first - std::get<1>(tub).first,2) + pow(std::get<2>(tub).second - std::get<1>(tub).second,2) ) {

Rectan<T> tu(std::get<0>(tub).first,std::get<0>(tub).second, std::get<2>(tub).first, std::get<2>(tub).second);

std::cout << "tuple is " << std::endl;

Print(tu);

std::cout << "center is (" << GCF(tu).first << "; " << GCF(tu).second << ")" << std::endl;

std::cout << "S = " << Sq(tu) << std::endl;

} else if (pow(std::get<1>(tub).first - std::get<0>(tub).first,2) + pow(std::get<1>(tub).second - std::get<0>(tub).second,2) ==

pow(std::get<2>(tub).first - std::get<1>(tub).first,2) + pow(std::get<2>(tub).second - std::get<1>(tub).second,2) ) {

Square<T> tu (std::get<0>(tub).first,std::get<0>(tub).second, std::get<2>(tub).first, std::get<2>(tub).second);

std::cout << "tuple is " << std::endl;

Print(tu);

std::cout << "center is (" << GCF(tu).first << "; " << GCF(tu).second << ")" << std::endl;

std::cout << "S = " << Sq(tu) << std::endl;

}

}else {

Trape<T> tu (std::get<0>(tub).first, std::get<0>(tub).second, std::get<1>(tub).first, std::get<1>(tub).second,

std::get<2>(tub).first,std::get<2>(tub).second, std::get<3>(tub).first, std::get<3>(tub).second);

std::cout << "tuple is " << std::endl;

Print(tu);

std::cout << "center is (" << GCF(tu).first << "; " << GCF(tu).second << ")" << std::endl;

std::cout << "S = " << Sq(tu) << std::endl;

}

}

template<class T>

bool oksq(T x1, T y1, T x3, T y3){

if((x1 != x3) && (y1 != y3)){

return true;

} else return false;

}

template<class T>

bool oktr(T x1, T y1, T x2, T y2, T x3, T y3, T x4, T y4){

if((y1 == y4) && (y2 == y3)){

if((pow(x2 - x1,2) + pow(y2 - y1,2)) == (pow(x3 - x4,2) + pow(y3 - y4,2)))

return true;

}

return false;

}

int main() {

std:: string str;

std::cout << "Press any bottom for istruction\n";

while(std::cin >> str){

if (str == "SI") {

int x1, y1, x2, y2;

std::cin >> x1 >> y1 >> x2 >> y2;

if (oksq(x1,y1,x2,y2)){

Square<int> s(x1,y1,x2,y2);

Print(s);

std::cout << "center is (" << GCF(s).first << "; " << GCF(s).second << ")" << std::endl;

std::cout << "S = " << Sq(s) << std::endl;

} else std::cout << "wrong input\n";

} else if (str == "SD") {

double x1, y1, x2, y2;

std::cin >> x1 >> y1 >> x2 >> y2;

if (oksq(x1,y1,x2,y2)){

Square<double> s(x1,y1,x2,y2);

Print(s);

std::cout << "center is (" << GCF(s).first << "; " << GCF(s).second << ")" << std::endl;

std::cout << "S = " << Sq(s) << std::endl;

} else std::cout << "wrong input\n";

} else if (str == "RI") {

int x1, y1, x2, y2;

std::cin >> x1 >> y1 >> x2 >> y2;

if (oksq(x1,y1,x2,y2)){

Rectan<int> s(x1,y1,x2,y2);

Print(s);

std::cout << "center is (" << GCF(s).first << "; " << GCF(s).second << ")" << std::endl;

std::cout << "S = " << Sq(s) << std::endl;

} else std::cout << "wrong input\n";

} else if (str == "RD") {

double x1, y1, x2, y2;

std::cin >> x1 >> y1 >> x2 >> y2;

if (oksq(x1,y1,x2,y2)){

Rectan<double> s(x1,y1,x2,y2);

Print(s);

std::cout << "center is (" << GCF(s).first << "; " << GCF(s).second << ")" << std::endl;

std::cout << "S = " << Sq(s) << std::endl;

} else std::cout << "wrong input\n";

} else if (str == "TI") {

int x1, y1, x2, y2, x3, y3, x4, y4;

std::cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;

if (oktr(x1,y1,x2,y2,x3,y3,x4,y4)){

Trape<int> t(x1,y1,x2,y2, x3, y3, x4, y4);

Print(t);

std::cout << "center is (" << GCF(t).first << "; " << GCF(t).second << ")" << std::endl;

std::cout << "S = " << Sq(t) << std::endl;

} else std::cout << "wrong input\n";

} else if (str == "TD") {

double x1, y1, x2, y2, x3, y3, x4, y4;

std::cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;

if (oktr(x1,y1,x2,y2,x3,y3,x4,y4)){

Trape<double> t(x1,y1,x2,y2,x3,y3,x4,y4);

Print(t);

std::cout << "center is (" << GCF(t).first << "; " << GCF(t).second << ")" << std::endl;

std::cout << "S = " << Sq(t) << std::endl;

} else std::cout << "wrong input\n";

} else if (str == "TUBI") {

int x1, y1, x2, y2, x3, y3, x4, y4;

std::cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;

if (oktr(x1,y1,x2,y2,x3,y3,x4,y4) && oksq(x1,y1,x3,y3)){

std::tuple<std::pair<int, int>, std::pair<int, int>, std::pair<int, int>, std::pair<int, int>> rtub(std::make\_pair(x1,y1),std::make\_pair(x2,y2),std::make\_pair(x3,y3),std::make\_pair(x4,y4));

tub(rtub);

} else std::cout << "wrong input\n";

} else if (str == "TUBD") {

double x1, y1, x2, y2, x3, y3, x4, y4;

std::cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;

if (oktr(x1,y1,x2,y2,x3,y3,x4,y4) && oksq(x1,y1,x3,y3)){

std::tuple<std::pair<double, double>, std::pair<double, double>, std::pair<double, double>, std::pair<double, double>> rtub(std::make\_pair(x1,y1),std::make\_pair(x2,y2),std::make\_pair(x3,y3),std::make\_pair(x4,y4));

tub(rtub);

} else std::cout << "wrong input\n";

} else if (str == "q"){

return 0;

}else{

std::cout << "USE:" << std::endl

<< "'SI or SD + x1 y1 x3 y3' to to exec Square with type int or double" << std::endl

<< "'RI or RD + x1 y1 x3 y3' to to exec Rectange with type int or double" << std::endl

<< "'TI or TD + x1 y1 x2 y2 x3 y3 x4 y4' to to exec Trape with type int or double"<< std:: endl

<< "'TUBI or TUBD + x1 y1 x2 y2 x3 y3 x4 y4' to to exec Tuple with type int or double" << std::endl

<< "'q' to quit" << std::endl;

}

}

}

6. Выводы

Шаблоны - средство языка C++, предназначенное для составления обобщенных алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию). Шаблоны позволяют избежать множества повторений в коде. Кортеж - коллекция элементов с фиксированным размером. Любая связанная пара, тройка, четверка и т.д элементов является кортежем. В качестве элементов кортежа могут выступать переменные произвольного типа. Во многих случаях удобно использовать кортежи. Например, кортежи позволяют легко определять и работать с функциями, возвращающими одно или более значений.

СПИСОК ЛИТЕРАТУРЫ

1. Шаблоны и шаблонные функции С++ [Электронный ресурс]. URL:<https://code-live.ru/post/cpp-template-functions/>

(дата обращения: 20.10.2019).

2. Кортежи в С++ [Электронный ресурс]. URL:

<https://qosys.info/tuple-cpp>

(дата обращения: 20.10.2019).