**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: Чернобаев Андрей Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата: 09.11.2020

Оценка:

Москва, 2020

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 только публичные поля. В классах не должно быть методов, только поля. Фигуры являются фигурами вращения (равнобедренными), за исключением трапеции и прямоугольника. Для хранения координат фигур необходимо использовать шаблон std::pair.

Например:

template <class T>

struct Square{

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

vertex\_t a,b,c,d;

};

Необходимо реализовать две шаблонных функции:

1. Функция **print** печати фигур на экран std::cout (печататься должны координаты вершин фигур). Функция должна принимать на вход std::tuple с фигурами, согласно варианту задания (минимум по одной каждого класса).

2. Функция **square** вычисления суммарной площади фигур. Функция должна принимать на вход std::tuple с фигурами, согласно варианту задания (минимум по одной каждого класса).

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.

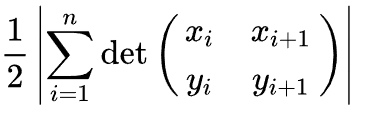
· Вычисляет суммарную площадь фигур в std::tuple и выводит значение на экран.

**Варианты заданий (выпуклые равносторонние фигуры вращения):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Фигура №1** | **Фигура №2** | **Фигура №3** |
| 14. | 5-угольник | 6-угольник | 8-угольник |

1. Описание программы.

Площадь многоугольников вычисляется с помощью формулы площади Гаусса:



Программа состоит из двух файлов:

accessory.h - Заголовочный файл, содержит дополнительные функции программы, облегчающие ввод-вывод.

main.cpp - основная часть программы.

Файл accessory.h состоит из:

* 3-х лямбдно-функций.
* 2-ух функций, реализующих split.
* функция help.

Файл main.cpp состоит из:

* Из трёх классов Pentagon, Hexagon и Octagon
* функции print\_array, выводящей на печать массив точек
* функции get\_det для вычисления детерминанта второго порядка
* функции gauss\_area для вычисления площади многоугольника по формуле Гаусса
* функции assign для присвоения значений из одного массива в другой
* функции print, соответствующей заданию
* функции square, соответствующей заданию

1. Набор тестов.

**Из файла test\_01.txt:**

9 9 3 6 1 0 4 2 3 8 pentagon

7 2 8 4 2 5 6 8 6 0 9 7 hexagon

3 6 8 1 6 9 3 0 6 9 0 0 6 1 0 7 octagon

9.459 9.637 3.641 6.85 1.375 0.045 4.541 2.825 3.777 8.048 pentagon

7.991 2.683 8.738 4.904 2.531 5.495 6.077 8.981 6.288 0.363 9.728 7.885 hexagon

3.337 6.088 8.338 1.906 6.313 9.73 3.742 0.787 6.408 9.553 0.424 0.048 6.402 1.798 0.092 7.295 octagon

area

print

exit

**Из файла test\_02.txt:**

0 0 7 5 5 1 6 7 5 5 pentagon

7 0 1 9 5 5 9 8 2 1 6 7 hexagon

9 6 5 2 5 5 3 4 0 4 0 4 6 6 6 9 octagon

0.682 0.451 7.163 5.888 5.607 1.545 6.3 7.823 5.858 5.509 pentagon

7.401 0.04 1.901 9.114 5.3 5.438 9.063 8.389 2.7 1.094 6.446 7.477 hexagon

9.714 6.623 5.698 2.354 5.21 5.611 3.563 4.066 0.997 4.244 0.868 4.511 6.483 6.474 6.055 9.134 octagon

area

print

exit

1. Результаты выполнения тестов.

|  |  |
| --- | --- |
| input | output |
| 9 9 3 6 1 0 4 2 3 8 pentagon  7 2 8 4 2 5 6 8 6 0 9 7 hexagon  3 6 8 1 6 9 3 0 6 9 0 0 6 1 0 7 octagon  9.459 9.637 3.641 6.85 1.375 0.045 4.541 2.825 3.777 8.048 pentagon  7.991 2.683 8.738 4.904 2.531 5.495 6.077 8.981 6.288 0.363 9.728 7.885 hexagon  3.337 6.088 8.338 1.906 6.313 9.73 3.742 0.787 6.408 9.553 0.424 0.048 6.402 1.798 0.092 7.295 octagon  area  print  exit | write exit for exit  write help for help  write area to calculate sum of areas  write index delete to remove figure by index  write go to show common function 1-3 for all array  input format x\_1 y\_1 x\_2 y\_2 ... x\_n y\_n figure  query 6:  area is 55.5722  query 7:  Pentagon:  9 9  3 6  1 0  4 2  3 8  Hexagon:  7 2  8 4  2 5  6 8  6 0  9 7  Octagon:  3 6  8 1  6 9  3 0  6 9  0 0  6 1  0 7  Pentagon:  9.459 9.637  3.641 6.85  1.375 0.045  4.541 2.825  3.777 8.048  Hexagon:  7.991 2.683  8.738 4.904  2.531 5.495  6.077 8.981  6.288 0.363  9.728 7.885  Octagon:  3.337 6.088  8.338 1.906  6.313 9.73  3.742 0.787  6.408 9.553  0.424 0.048  6.402 1.798  0.092 7.295 |
| 0 0 7 5 5 1 6 7 5 5 pentagon  7 0 1 9 5 5 9 8 2 1 6 7 hexagon  9 6 5 2 5 5 3 4 0 4 0 4 6 6 6 9 octagon  0.682 0.451 7.163 5.888 5.607 1.545 6.3 7.823 5.858 5.509 pentagon  7.401 0.04 1.901 9.114 5.3 5.438 9.063 8.389 2.7 1.094 6.446 7.477 hexagon  9.714 6.623 5.698 2.354 5.21 5.611 3.563 4.066 0.997 4.244 0.868 4.511 6.483 6.474 6.055 9.134 octagon  area  print  exit | write exit for exit  write help for help  write area to calculate sum of areas  write index delete to remove figure by index  write go to show common function 1-3 for all array  input format x\_1 y\_1 x\_2 y\_2 ... x\_n y\_n figure  query 6:  area is 66.075  query 7:  Pentagon:  0 0  7 5  5 1  6 7  5 5  Hexagon:  7 0  1 9  5 5  9 8  2 1  6 7  Octagon:  9 6  5 2  5 5  3 4  0 4  0 4  6 6  6 9  Pentagon:  0.682 0.451  7.163 5.888  5.607 1.545  6.3 7.823  5.858 5.509  Hexagon:  7.401 0.04  1.901 9.114  5.3 5.438  9.063 8.389  2.7 1.094  6.446 7.477  Octagon:  9.714 6.623  5.698 2.354  5.21 5.611  3.563 4.066  0.997 4.244  0.868 4.511  6.483 6.474  6.055 9.134 |

1. Листинг программы.

Файл accessory.h:

#ifndef TASK3\_ACCESSORY\_H

#define TASK3\_ACCESSORY\_H

#include <string>

#include <vector>

#include <sstream>

auto string\_to\_int = [](const std::string& s) {return std::stoi(s);};

auto string\_to\_double = [](const std::string& s) {return std::stod(s);};

auto int\_to\_string = [](const int i) {return std::to\_string(i);};

template <typename Out>

void split(const std::string &s, char delim, Out result) {

std::istringstream iss(s);

std::string item;

while (std::getline(iss, item, delim)) {

\*result++ = item;

}

}

std::vector<std::string> split(const std::string &s, char delim) {

std::vector<std::string> elems;

split(s, delim, std::back\_inserter(elems));

return elems;

}

void help(){

std::cout << "write exit for exit" << std::endl;

std::cout << "write help for help" << std::endl;

std::cout << "write area to calculate sum of areas" << std::endl;

std::cout << "write index delete to remove figure by index" << std::endl;

std::cout << "write go to show common function 1-3 for all array" << std::endl;

std::cout << "input format x\_1 y\_1 x\_2 y\_2 ... x\_n y\_n figure" << std::endl;

std::cout << "" << std::endl;

}

#endif //TASK3\_ACCESSORY\_H

Файл main.cpp

#include <iostream>

#include <tuple>

#include <vector>

#include <cmath>

#include "accessory.h"

/\*

Необходимо реализовать две шаблонных функции:

1. Функция print печати фигур на экран std::cout

(печататься должны координаты вершин фигур). Функция должна принимать на вход std::tuple с фигурами,

согласно варианту задания (минимум по одной каждого класса).

2. Функция square вычисления суммарной площади фигур. Функция должна принимать на вход

std::tuple с фигурами, согласно варианту задания (минимум по одной каждого класса).

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.

вариант 14

\*/

template <class T>

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

template <class T>

class Pentagon {

public:

vertex\_t<T> vertexes[5];

};

template <class T>

class Hexagon {

public:

vertex\_t<T> vertexes[6];

};

template <class T>

class Octagon {

public:

vertex\_t<T> vertexes[8];

};

template <class T>

void print\_array(std::stringstream& buffer, const vertex\_t<T>\* array, size\_t size, const char\* name){

buffer << name << ": " << std::endl;

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

buffer << array[i].first << " " << array[i].second << std::endl;

}

}

double get\_det(const double x1, const double y1,

const double x2, const double y2){

return x1 \* y2 - x2 \* y1;

}

template <class T>

double gauss\_area(const vertex\_t<T>\* nodes, size\_t size){

double out = 0.0;

auto count = size;

for(size\_t i = 0; i < (count - 1); i++){

out += get\_det(static\_cast<double>(nodes[i].first),static\_cast<double>(nodes[i].second),

static\_cast<double>(nodes[i+1].first),static\_cast<double>(nodes[i+1].second));

}

out += get\_det(static\_cast<double>(nodes[count-1].first),static\_cast<double>(nodes[count-1].second),

static\_cast<double>(nodes[0].first),static\_cast<double>(nodes[0].second));

return fabs(0.5 \* out);

}

template <class T>

void assign(vertex\_t<T>\* nodes,std::vector<vertex\_t<T>>& numbers, size\_t size){

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

nodes[i] = numbers[i];

}

}

template <class T,class K>

void print(std::stringstream& buffer, const std::tuple<Pentagon<T>,Hexagon<T>,Octagon<T>,Pentagon<K>,Hexagon<K>,Octagon<K>>& array){

Pentagon<T> pentagon1 = std::get<0>(array);

Hexagon<T> hexagon1 = std::get<1>(array);

Octagon<T> octagon1 = std::get<2>(array);

Pentagon<K> pentagon2 = std::get<3>(array);

Hexagon<K> hexagon2 = std::get<4>(array);

Octagon<K> octagon2 = std::get<5>(array);

print\_array<T>(buffer, pentagon1.vertexes,5, "Pentagon");

print\_array<T>(buffer, hexagon1.vertexes,6, "Hexagon");

print\_array<T>(buffer, octagon1.vertexes,8, "Octagon");

print\_array<K>(buffer, pentagon2.vertexes,5, "Pentagon");

print\_array<K>(buffer, hexagon2.vertexes,6, "Hexagon");

print\_array<K>(buffer, octagon2.vertexes,8, "Octagon");

}

template <class T,class K>

double square(const std::tuple<Pentagon<T>,Hexagon<T>,Octagon<T>,Pentagon<K>,Hexagon<K>,Octagon<K>>& array){

double pentagon\_area1 = gauss\_area(std::get<0>(array).vertexes,5);

double hexagon\_area1 = gauss\_area(std::get<1>(array).vertexes,6);

double octagon\_area1 = gauss\_area(std::get<2>(array).vertexes,8);

double pentagon\_area2 = gauss\_area(std::get<3>(array).vertexes,5);

double hexagon\_area2 = gauss\_area(std::get<4>(array).vertexes,6);

double octagon\_area2 = gauss\_area(std::get<5>(array).vertexes,8);

return pentagon\_area1 + hexagon\_area1 + octagon\_area1 +

pentagon\_area2 + hexagon\_area2 + octagon\_area2;

}

int main(){

std::string cmd;

std::vector<std::string> out;

std::stringstream buffer;

int count = 0;

std::tuple<Pentagon<int>,Hexagon<int>,Octagon<int>,Pentagon<double>,Hexagon<double>,Octagon<double>> gtuple;

help();

while (true){

getline(std::cin, cmd);

std::string prefix = std::string("query ") + int\_to\_string(count) + ": ";

if (cmd == "exit"){

break;

}

else if (cmd == "help"){

help();

}

else if (cmd == "area"){

buffer << prefix << std::endl;

buffer << "area is " << square(gtuple) << std::endl;

}

else if (cmd == "print"){

buffer << prefix << std::endl;

print<int,double>(buffer,gtuple);

}

else {

std::vector<std::string> array = split(cmd,' ');

std::vector<vertex\_t<int>> numbers\_int;

numbers\_int.resize((array.size() - 1) / 2);

std::vector<vertex\_t<double>> numbers\_double;

numbers\_double.resize((array.size() - 1) / 2);

int j = 0;

for(size\_t i = 0;i < numbers\_int.size() \* 2 ; i += 2){

//std::cout << "i is " << i << std::endl;

//std::cout << "i+1 is " << i + 1 << std::endl;

if (count >= 0 && count <= 2){

numbers\_int[j] = {string\_to\_int(array[i]),string\_to\_int(array[i+1])};

}

else if (count >= 3 && count <= 5){

numbers\_double[j] = {string\_to\_double(array[i]),string\_to\_double(array[i+1])};

}

else {

std::perror("count error");

}

j++;

}

//std::cout << "\*\*\*\*\*\*\*" << std::endl;

if (count >= 0 && count <= 2){

if (array.back() == "pentagon"){

assign<int>(std::get<0>(gtuple).vertexes,numbers\_int,5);

}

else if (array.back() == "hexagon"){

assign<int>(std::get<1>(gtuple).vertexes,numbers\_int,6);

}

else if (array.back() == "octagon"){

assign<int>(std::get<2>(gtuple).vertexes,numbers\_int,8);

}

}

else if (count >= 3 && count <= 5){

if (array.back() == "pentagon"){

assign<double>(std::get<3>(gtuple).vertexes,numbers\_double,5);

}

else if (array.back() == "hexagon"){

assign<double>(std::get<4>(gtuple).vertexes,numbers\_double,6);

}

else if (array.back() == "octagon"){

assign<double>(std::get<5>(gtuple).vertexes,numbers\_double,8);

}

}

}

count++;

}

std::cout << buffer.str() << std::endl;

}

1. Выводы.

Я научился работать с шаблонами и с std::tuple в С++.

1. Список использованных источников.

1 C++ Inheritance [Электронный ресурс].

URL: <https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm>

2 Templates [Электронный ресурс].

URL: https://www.cplusplus.com/doc/oldtutorial/templates/

3 std::tuple [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/utility/tuple

4 Shoelace formula [Электронный ресурс].

URL: <https://en.wikipedia.org/wiki/Shoelace_formula>

5 static\_cast conversion [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/language/static\_cast