**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:

Группа:

Преподаватель:

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Вариант 16:

|  |  |  |  |
| --- | --- | --- | --- |
| 16 | 8-угольник | Треугольник | Квадрат |

1. **Описание программы**

Программа содержит вектор указателей на родительский класс геометрических фигур. Это позволяет хранить в одном векторе объекты различных классов. Пользователь может добавлять фигуры в конец вектора, удалять фигуры по индексу, выводить геометрический центр, площадь, список вершин для конкретной фигуры и для всех фигур в векторе, а также посчитать сумму площадей для всех фигур. Также пользователь может получить всю информацию об одной фигуре, но введённой как кортеж.

1. **Набор testcases**

|  |  |  |
| --- | --- | --- |
| № | Описание | Ввод |
| 1 | Демонстрация основных функций программы | 8  3  0 0  0 2  2 0  8  2  1 0  2 1  1 2  0 1  8  1  1 0  2 0  3 1  3 2  2 3  1 3  0 2  0 1  0 |

1. **Результаты выполнения тестов.**

*test\_01.txt*:

1. Show commands

2. Add figure

3. Delete figure

4. Center point

5. Print points

6. Size of figure

7. Total size

8. Tuple figure

0. Exit

> 8

Choose size of tuple:

1 - Octagon (8 points)

2 - Square (4 points)

3 - Triangle (3 points)

Type: 3

Enter coordinates of points:

0 0

0 2

2 0

Center point: (0.666667, 0.666667)

Points: (0, 0), (0, 2), (2, 0)

Size: 2

> 8

Choose size of tuple:

1 - Octagon (8 points)

2 - Square (4 points)

3 - Triangle (3 points)

Type: 2

Enter coordinates of points:

1 0

2 1

1 2

0 1

Center point: (1, 1)

Points: (1, 0), (2, 1), (1, 2), (0, 1)

Size: 2

> 8

Choose size of tuple:

1 - Octagon (8 points)

2 - Square (4 points)

3 - Triangle (3 points)

Type: 1

Enter coordinates of points:

1 0

2 0

3 1

3 2

2 3

1 3

0 2

0 1

Center point: (1.5, 1.5)

Points: (1, 0), (2, 0), (3, 1), (3, 2), (2, 3), (1, 3), (0, 2), (0, 1)

Size: 7

> 0

1. **Листинг программы**

#include <iostream>

#include <vector>

#include <cmath>

#include <tuple>

#define PI 3.14159265f

//родительский класс для всех фигур

template<class T>

class Figure

{

public:

//точка многоугольника

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

//вычисление геометрического центра фигуры

point getCenter()

{

point center = std::make\_pair((T)0, (T)0);

for (point p : m\_points)

{

center.first += p.first;

center.second += p.second;

}

center.first /= m\_points.size();

center.second /= m\_points.size();

return center;

}

//вывод координат вершин фигуры

void print()

{

bool comma = false; //печатать запятую перед точкой или нет

for (point p : m\_points)

{

if (comma) std::cout << ", ";

comma = true;

std::cout << "(" << p.first << ", " << p.second << ")";

}

}

//вычисление площади

T size()

{

T S = (T)0;

for (int i = 0; i < m\_points.size() - 1; i++)

S += triag(m\_points[0], m\_points[i], m\_points[i + 1]);

return S;

}

protected:

//точки многоугольника

std::vector<point> m\_points;

//площадь треугольника по координатам вершин

//S = 1/2 \* abs(det(x1 - x3, y1 - y3; x2 - x3, y2 - y3))

T triag(point& a, point& b, point& c)

{

return 0.5 \* abs((a.first - c.first) \* (b.second - c.second) -

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

}

};

//Любую фигуру вращения можно задать координатами центра, радиусом описанной окружности и углом поворота

//8-угольник

template<class T>

class Octagon : public Figure<T>

{

public:

//заполняем вектор вершин

Octagon(T x, T y, T r, T a)

{

for (int i = 0; i < 8; i++)

{

T phi = a + i \* PI / 4.0;

m\_points.push\_back(std::make\_pair(r \* cos(phi) + x, r \* sin(phi) + y));

}

}

};

//Квадрат

template<class T>

class Square : public Figure<T>

{

public:

//заполняем вектор вершин

Square(T x, T y, T r, T a)

{

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

{

T phi = a + i \* PI / 2.0;

m\_points.push\_back(std::make\_pair(r \* cos(phi) + x, r \* sin(phi) + y));

}

}

};

//Треугольник

template<class T>

class Triangle : public Figure<T>

{

public:

//заполняем вектор вершин

Triangle(T x, T y, T r, T a)

{

for (int i = 0; i < 3; i++)

{

T phi = a + i \* PI / 1.5;

m\_points.push\_back(std::make\_pair(r \* cos(phi) + x, r \* sin(phi) + y));

}

}

};

//Преобразование tuple в вектор для более удобной работы с ним

template<class T>

std::vector<T> toVector(std::tuple<> t)

{

return std::vector<T>();

}

template<class T, class... T1>

std::vector<T> toVector(std::tuple<T, T1...> t)

{

//приведение tuple к родительскому классу даст нам хвост tuple

std::tuple<T1...> tail = \*((std::tuple<T1...>\*) & t);

std::vector<T> vec = toVector<T>(tail);

vec.insert(vec.begin(), std::get<0>(t));

return vec;

}

//Функции для tuple

template<class T, class... T1>

std::pair<T, T> tuple\_getCenter(std::tuple<T1...> t)

{

std::pair<T, T> center = std::make\_pair((T)0, (T)0);

std::vector<std::pair<T, T>> points = toVector(t);

for (std::pair<T, T> p : points)

{

center.first += p.first;

center.second += p.second;

}

center.first /= points.size();

center.second /= points.size();

return center;

}

template<class T, class... T1>

void tuple\_print(std::tuple<T1...> t)

{

bool comma = false; //печатать запятую перед точкой или нет

std::vector<std::pair<T, T>> points = toVector(t);

for (std::pair<T, T> p : points)

{

if (comma) std::cout << ", ";

comma = true;

std::cout << "(" << p.first << ", " << p.second << ")";

}

}

template<class T, class... T1>

T tuple\_size(std::tuple<T1...> t)

{

T S = (T)0;

std::vector<std::pair<T, T>> points = toVector(t);

for (int i = 0; i < points.size() - 1; i++)

{

S += 0.5 \* abs(

(points[0].first - points[i + 1].first) \* (points[i].second - points[i + 1].second) -

(points[i].first - points[i + 1].first) \* (points[0].second - points[i + 1].second));

}

return S;

}

//список команд

void showCommands()

{

std::cout <<

"1. Show commands" << std::endl <<

"2. Add figure" << std::endl <<

"3. Delete figure" << std::endl <<

"4. Center point" << std::endl <<

"5. Print points" << std::endl <<

"6. Size of figure" << std::endl <<

"7. Total size" << std::endl <<

"8. Tuple figure" << std::endl <<

"0. Exit" << std::endl;

}

int main()

{

//вектор фигур

std::vector<Figure<float>\*> figures;

showCommands();

//цикл программы

bool loop = true;

while (loop)

{

//читаем введённую команду

std::cout << "> ";

int command;

std::cin >> command;

switch (command)

{

case 0:

loop = false;

break;

case 1:

showCommands();

break;

case 2:

{

std::cout << "Choose type:" << std::endl <<

"1 - Octagon" << std::endl <<

"2 - Square" << std::endl <<

"3 - Triangle" << std::endl <<

"Type: ";

int type;

std::cin >> type;

if (type < 1 || type > 3)

std::cout << "Unknown type" << std::endl;

else

{

float x, y, r, a;

std::cout << "Coordinates of center: ";

std::cin >> x >> y;

std::cout << "Radius: ";

std::cin >> r;

std::cout << "Angle: ";

std::cin >> a;

switch (type)

{

case 1:

figures.push\_back(new Octagon<float>(x, y, r, a));

break;

case 2:

figures.push\_back(new Square<float>(x, y, r, a));

break;

case 3:

figures.push\_back(new Triangle<float>(x, y, r, a));

break;

}

}

break;

}

case 3:

{

std::cout << "Index: ";

int index;

std::cin >> index;

if (index < 0 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

delete figures[index];

figures.erase(figures.begin() + index);

}

break;

}

case 4:

{

std::cout << "Index (-1 to call for all figures): ";

int index;

std::cin >> index;

if (index == -1)

for (int i = 0; i < figures.size(); i++)

{

auto center = figures[i]->getCenter();

std::cout << i << ": (" << center.first << ", " << center.second << ")" << std::endl;

}

else if (index < -1 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

auto center = figures[index]->getCenter();

std::cout << index << ": (" << center.first << ", " << center.second << ")" << std::endl;

}

break;

}

case 5:

{

std::cout << "Index (-1 to call for all figures): ";

int index;

std::cin >> index;

if (index == -1)

for (int i = 0; i < figures.size(); i++)

{

std::cout << i << ": ";

figures[i]->print();

std::cout << std::endl;

}

else if (index < -1 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

std::cout << index << ": ";

figures[index]->print();

std::cout << std::endl;

}

break;

}

case 6:

{

std::cout << "Index (-1 to call for all figures): ";

int index;

std::cin >> index;

if (index == -1)

for (int i = 0; i < figures.size(); i++)

{

std::cout << i << ": " << figures[i]->size() << std::endl;

}

else if (index < -1 || index >= figures.size())

std::cout << "Index out of bounds" << std::endl;

else

{

std::cout << index << ": " << figures[index]->size() << std::endl;

}

break;

}

case 7:

{

float total = 0.0f;

for (auto\* f : figures)

total += f->size();

std::cout << "Total size of all figures: " << total << std::endl;

break;

}

case 8:

{

std::cout << "Choose size of tuple:" << std::endl <<

"1 - Octagon (8 points)" << std::endl <<

"2 - Square (4 points)" << std::endl <<

"3 - Triangle (3 points)" << std::endl <<

"Type: ";

int type;

std::cin >> type;

switch (type)

{

case 1:

{

std::cout << "Enter coordinates of points:" << std::endl;

float ax, ay, bx, by, cx, cy, dx, dy, ex, ey, fx, fy, gx, gy, hx, hy;

std::cin >> ax >> ay >> bx >> by >> cx >> cy >> dx >> dy >> ex >> ey >> fx >> fy >> gx >> gy >> hx >> hy;

std::tuple<std::pair<float, float>, std::pair<float, float>,

std::pair<float, float>, std::pair<float, float>, std::pair<float, float>,

std::pair<float, float>, std::pair<float, float>, std::pair<float, float>> fig =

std::make\_tuple(std::make\_pair(ax, ay), std::make\_pair(bx, by),

std::make\_pair(cx, cy), std::make\_pair(dx, dy), std::make\_pair(ex, ey),

std::make\_pair(fx, fy), std::make\_pair(gx, gy), std::make\_pair(hx, hy));

std::pair<float, float> center = tuple\_getCenter<float>(fig);

std::cout << "Center point: (" << center.first << ", " << center.second << ")" << std::endl;

std::cout << "Points: ";

tuple\_print<float>(fig);

std::cout << std::endl;

std::cout << "Size: " << tuple\_size<float>(fig) << std::endl;

break;

}

case 2:

{

std::cout << "Enter coordinates of points:" << std::endl;

float ax, ay, bx, by, cx, cy, dx, dy;

std::cin >> ax >> ay >> bx >> by >> cx >> cy >> dx >> dy;

std::tuple<std::pair<float, float>, std::pair<float, float>,

std::pair<float, float>, std::pair<float, float>> fig =

std::make\_tuple(std::make\_pair(ax, ay), std::make\_pair(bx, by),

std::make\_pair(cx, cy), std::make\_pair(dx, dy));

std::pair<float, float> center = tuple\_getCenter<float>(fig);

std::cout << "Center point: (" << center.first << ", " << center.second << ")" << std::endl;

std::cout << "Points: ";

tuple\_print<float>(fig);

std::cout << std::endl;

std::cout << "Size: " << tuple\_size<float>(fig) << std::endl;

break;

}

case 3:

{

std::cout << "Enter coordinates of points:" << std::endl;

float ax, ay, bx, by, cx, cy;

std::cin >> ax >> ay >> bx >> by >> cx >> cy;

std::tuple<std::pair<float, float>, std::pair<float, float>, std::pair<float, float>> fig =

std::make\_tuple(std::make\_pair(ax, ay), std::make\_pair(bx, by), std::make\_pair(cx, cy));

std::pair<float, float> center = tuple\_getCenter<float>(fig);

std::cout << "Center point: (" << center.first << ", " << center.second << ")" << std::endl;

std::cout << "Points: ";

tuple\_print<float>(fig);

std::cout << std::endl;

std::cout << "Size: " << tuple\_size<float>(fig) << std::endl;

break;

}

default:

std::cout << "Unknown type" << std::endl;

break;

}

break;

}

default:

std::cout << "Unknown command" << std::endl;

break;

}

}

return 0;

}

1. **Выводы:**

Изучены основы работы с шаблонами и кортежами на языке C++. Разработана программа на языке C++, позволяющая работать как с шаблонными фигурами, так и с кортежами.

**СПИСОК ЛИТЕРАТУРЫ**

1. Классы-шаблоны [электронный ресурс]. URL: <http://www.c-cpp.ru/books/klassy-shablony>
2. std::tuple [электронный ресурс]. URL: <https://ru.cppreference.com/w/cpp/utility/tuple>