Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 3: «Наследование, полиморфизм»

Группа:	М8О-208Б-18, №9
Студент:	Игитова Александра Андреевна
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	30.11.2019

Москва, 2019

1. Задание

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры

являются фигурами вращения. Все классы должны поддерживать набор общих методов:

1. Вычисление геометрического центра фигуры;

2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;

3. Вычисление площади фигуры;

Создать программу, которая позволяет:

• Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.

• Сохранять созданные фигуры в динамический массив std::vector<Figure*>

• Вызывать для всего массива общие функции (1-3 см. выше).Т.е. распечатывать для каждой фигуры в массиве

геометрический центр, координаты вершин и площадь.

• Необходимо уметь вычислять общую площадь фигур в массиве.

• Удалять из массива фигуру по индексу;

Вариант 9: треугольник, квадрат, прямоугольник

2. Адрес репозитория на GitHub

https://github.com/SandraIgitova/oop_exercise_03

3. Код программы на C++

main.cpp

```
#include "Figure.h"
#include <cstdio>
#include <stdlib.h>
#include <cmath>
#include <vector>
#include <iostream>
#include <cassert>

void MainMenuOutput(uint32_t& x) {
    std::cout << "\
1. Добавить фигуру в массив\n\
2. Удалить фигуру из массива\n\
3. Распечатать для каждой фигуры в массиве геом центр, координаты вершин, площадь\n\
```

```
4. Вычислить общую площадь фигур в массиве\n\
Ctrl-D Выход из программы\n\n" << std::endl;
    std::cin >> x;
    std::cout << std::endl;
}

void FigureMenuOutput(uint32_t& x) {
    std::cout << "\
1. Добавить треугольник\n\
2. Добавить четырехугольник\n\n" << std::endl;
    std::cin >> x;
}

void TriangleInput(std::vector<Figure*>& figures) {
    _2D A, B, C;
    std::cout << "Введите 6 чисел координат сторон треугольника, чередуя X
и Y: \n";
    std::cin >> A.x >> A.y >> B.x >> B.y >> C.x >> C.y;
    figures.push_back(new Triangle(A, B, C));
}

void RectangleInput(std::vector<Figure*>& figures) {
    _2D A, B, C, D;
    std::cout << "Введите 8 чисел координат сторон четырехугольника,
чередуя X и Y: \n";
    std::cin >> A.x >> A.y >> B.x >> B.y >> C.x >> C.y >> D.x >> D.y;
    if (IsRectangle(A, B, C, D)) {
        figures.push_back(new Rectangle(A, B, C, D));
    }
    else std::cout << "Это не прямоугольник\n";
    if (IsSquare(A, B, C, D)) {
        std::cout << "Данный прямоугольник является квадратом:" <<
std::endl;
    }
}

void IndexInput(std::vector<Figure*>& figures) {
    uint index;
    std::cout << "Введите индекс элемента в массиве\n" << std::endl;
    std::cin >> index;
    if (index > figures.size() - 1) {
        std::cout << "Недопустимый индекс\n" << std::endl;
    }
    else figures.erase(figures.begin() + index);
    std::cout << std::endl;
```

```
}

void FiguresOutput(std::vector<Figure*>& figures) {
    std::cout << "Вывод с всех фигур" << std::endl;
    for (uint n = 0; n < figures.size(); n++) {
        std::cout << "Фигура № " << n << std::endl;
        if (figures[n]->m_t == 3) {                         // обращаемся в векторе
            ((Triangle*)figures[n])->Output();     // к фигурам разных классов
        }                                                                   //
(треугольник или четырехугольник)
        if (figures[n]->m_t == 4) {
            ((Rectangle*)figures[n])->Output();
        }
        std::cout << "------------------\n" << std::endl;
    }
}

void Area(std::vector<Figure*>& figures) {
    std::cout << "Общая площадь фигур в массиве\n" << std::endl;
    std::cout << TotalArea(figures) << std::endl;
    std::cout << std::endl;
}
// --------------------

int main()
{
    uint32_t x = 0;
    std::vector<Figure*> figures;
    while (std::cin)
    {
        MainMenuOutput(x);
        if (x == 1) {
            FigureMenuOutput(x);
            if (x == 1) {
                TriangleInput(figures);
            }
            if (x == 2) {
                RectangleInput(figures);
            }
            continue;
        }
        if (x == 2) {
            IndexInput(figures);
            continue;
        }
```

```
        if (x == 3) {
                FiguresOutput(figures);
                continue;
        }
        if (x == 4) {
                Area(figures);
        }
    }
    return 0;
}
```

Figure.h

```
#pragma once

#ifndef FIGURE_HPP
#define FIGURE_HPP
#include <vector>


struct _2D
{
   double x;
   double y;
};


class Figure
{
public:
   unsigned int m_t; // количество вершин в фигуре, необходимо
для доступа к методам дочерних классов
                  // при извлечении из вектора (преобразовываем
Figure* в Triangle* или Rectangle* )

   virtual _2D Center() = 0;
   virtual double Area() = 0;
   virtual void Output() = 0;
};

class Triangle: public Figure
{
public:
   _2D m_tops[3];
   Triangle(_2D A, _2D B, _2D C);

   virtual double Area();
   virtual _2D Center();
   virtual void Output();
};

class Square: public Figure
{
public:
   _2D m_tops[4];
   Square(_2D A, _2D B, _2D C, _2D D);
   virtual double Area();
```

```
   virtual _2D Center();
   virtual void Output();
};

class Rectangle: public Figure
{
public:
   _2D m_tops[4];
   Rectangle(_2D A, _2D B, _2D C, _2D D);

   virtual double Area();
   virtual _2D Center();
   virtual void Output();
};

int comp(const void * a, const void * b);
bool IsRectangle(_2D A, _2D B, _2D C, _2D D);
bool IsSquare(_2D A, _2D B, _2D C, _2D D);
double TotalArea(std::vector<Figure*> &figures);
#endif
```

Figure.cpp

```
#include "Figure.h"
#include <cstdio>
#include <stdlib.h>
#include <cmath>
#include <iostream>
#include <cassert>


//------------------- математика------------------------------
// площадь прямоугольника
int comp(const void * a, const void * b) {
   return ((_2D*)a)->x - ((_2D*)b)->x;
}

bool IsRectangle(_2D A, _2D B, _2D C, _2D D) {
   _2D mas[4] = { A, B, C, D };
   qsort(mas, 4, sizeof(_2D), comp); // для введения точек в
произвольном порядке
   if (mas[0].x == mas[1].x) {
      if (mas[0].y > mas[1].y) { _2D hlp = mas[1]; mas[1] =
mas[0]; mas[0] = hlp; }
      if (mas[2].y < mas[3].y) { _2D hlp = mas[2]; mas[2] =
mas[3]; mas[3] = hlp; }
   }
   else if (mas[1].y < mas[3].y) { _2D hlp = mas[1]; mas[1] =
mas[3]; mas[3] = hlp; }
   _2D vector1, vector2, vector3, vector4;
   vector1.x = mas[1].x - mas[0].x; vector1.y = mas[1].y -
mas[0].y;
   vector2.x = mas[2].x - mas[1].x; vector2.y = mas[2].y -
mas[1].y;
   vector3.x = mas[3].x - mas[2].x; vector3.y = mas[3].y -
mas[2].y;
   vector4.x = mas[0].x - mas[3].x; vector4.y = mas[0].y -
mas[3].y;
   // проверяем три угла скалярными произведениями
```

```
   if (((vector1.x * vector2.x + vector1.y * vector2.y) == 0) &&
((vector3.x * vector2.x + vector3.y * vector2.y) == 0) &&
((vector4.x * vector3.x + vector4.y * vector3.y) == 0)) {
      return true;
   }
   else { return false; }

}

bool IsSquare(_2D A, _2D B, _2D C, _2D D) {
   _2D mas[4] = { A, B, C, D };
   qsort(mas, 4, sizeof(_2D), comp); // для введения точек в
произвольном порядке
   if (mas[0].x == mas[1].x) {
      if (mas[0].y > mas[1].y) { _2D hlp = mas[1]; mas[1] =
mas[0]; mas[0] = hlp; }
      if (mas[2].y < mas[3].y) { _2D hlp = mas[2]; mas[2] =
mas[3]; mas[3] = hlp; }
   }
   else if (mas[1].y < mas[3].y) { _2D hlp = mas[1]; mas[1] =
mas[3]; mas[3] = hlp; }
   double d1 = sqrt(pow(mas[1].x - mas[0].x, 2) + pow(mas[1].y -
mas[0].y, 2));
   double d2 = sqrt(pow(mas[2].x - mas[1].x, 2) + pow(mas[1].y -
mas[2].y, 2));
   _2D vector1, vector2, vector3, vector4;
   vector1.x = mas[1].x - mas[0].x; vector1.y = mas[1].y -
mas[0].y;
   vector2.x = mas[2].x - mas[1].x; vector2.y = mas[2].y -
mas[1].y;
   vector3.x = mas[3].x - mas[2].x; vector3.y = mas[3].y -
mas[2].y;
   // проверяем два угла скалярными произведениями и равенство
сторон
   if (((vector1.x * vector2.x + vector1.y * vector2.y) == 0) &&
((vector3.x * vector2.x + vector3.y * vector2.y) == 0) &&
((vector4.x * vector3.x + vector4.y * vector3.y) == 0) && (d1 ==
d2)){
      return true;
   }
   else { return false; }
}


double RectangleArea(_2D A, _2D B, _2D C, _2D D)
{
   _2D mas[4] = { A, B, C, D };
   qsort(mas, 4, sizeof(_2D), comp); // для введения точек в
произвольном порядке
   if (mas[0].x == mas[1].x) {
      if (mas[0].y > mas[1].y) { _2D hlp = mas[1]; mas[1] =
mas[0]; mas[0] = hlp; }
      if (mas[2].y < mas[3].y) { _2D hlp = mas[2]; mas[2] =
mas[3]; mas[3] = hlp; }
   }
   else if (mas[1].y < mas[3].y) {_2D hlp = mas[1]; mas[1] =
mas[3]; mas[3] = hlp;}
   double d1 = sqrt(pow(mas[1].x - mas[0].x, 2) + pow(mas[1].y -
mas[0].y, 2));
```

```
   double d2 = sqrt(pow(mas[2].x - mas[1].x, 2) + pow(mas[1].y -
mas[2].y, 2));
   return d1 * d2;
};
// площадь треугольника по формуле Герона
double TriangleArea(_2D A, _2D B, _2D C)
{
   _2D mas[3] = { A, B, C };
   qsort(mas, 3, sizeof(_2D), comp); // для введения точек в
произвольном порядке
   if (mas[0].x == mas[1].x)
     if (mas[0].y > mas[1].y) { _2D hlp = mas[1]; mas[1] =
mas[0]; mas[0] = hlp; }
   if (mas[1].x == mas[2].x)
     if (mas[2].y > mas[1].y) { _2D hlp = mas[1]; mas[1] =
mas[2]; mas[2] = hlp; }
   else if (mas[1].y < mas[2].y) { _2D hlp = mas[1]; mas[1] =
mas[2]; mas[2] = hlp; }
   double a = sqrt(pow(mas[1].x - mas[0].x, 2) + pow(mas[1].y -
mas[0].y, 2));
   double b = sqrt(pow(mas[1].x - mas[2].x, 2) + pow(mas[2].y -
mas[1].y, 2));
   double c = sqrt(pow(mas[2].x - mas[0].x, 2) + pow(mas[0].y -
mas[2].y, 2));
   double p = (a + b + c)/2;
   return  sqrt(p*(p - a)*(p - b)*(p - c));
};
// центр прямоугольника
_2D RectangleCenter(_2D A, _2D B, _2D C, _2D D) {
   _2D Center;
   Center.x = (A.x + B.x + C.x + D.x) / 4;
   Center.y = (A.y + B.y + C.y + D.y) / 4;
   return Center;
}
// центр треугольника
_2D TriangleCenter(_2D A, _2D B, _2D C) {
   _2D Center;
   Center.x = (A.x + B.x + C.x) / 3;
   Center.y = (A.y + B.y + C.y) / 3;
   return Center;
}

// ---------------- конец математики ----------------------

// ---------------- классы --------------------------------

//Triangle kek

Triangle::Triangle(_2D A, _2D B, _2D C){
     m_tops[0].x = A.x;           m_tops[0].y = A.y;
     m_tops[1].x = B.x;           m_tops[1].y = B.y;
     m_tops[2].x = C.x;           m_tops[2].y = C.y;
     m_t = 3;
   }

   double Triangle::Area() {
     return TriangleArea(m_tops[0], m_tops[1], m_tops[2]);
   }
    _2D Triangle::Center() {
     return TriangleCenter(m_tops[0], m_tops[1], m_tops[2]);
```

```
   }
   void Triangle::Output() {
      std::cout << "Координаты вершин:" << std::endl;
      std::cout << "\t" << "X" << "\t" << "Y" << std::endl;
      for (int number = 0; number < m_t; number++) {
           std::cout << "\t" << m_tops[number].x << "\t" <<
m_tops[number].y << std::endl;
      }
      std::cout << "Координаты центра:" << std::endl;
      std::cout << "\t" << "X" << "\t" << "Y" << std::endl;
      _2D m_center = Center();
      std::cout << "\t" << m_center.x << "\t" << m_center.y <<
std::endl;

      std::cout << "Площадь треугольника:" << std::endl;
      std::cout << "\t" << Area() << std::endl;
   }


//Square kek

Square::Square(_2D A, _2D B, _2D C, _2D D){
     m_tops[0].x = A.x;            m_tops[0].y = A.y;
     m_tops[1].x = B.x;            m_tops[1].y = B.y;
     m_tops[2].x = C.x;            m_tops[2].y = C.y;
     m_tops[3].x = D.x;            m_tops[3].y = D.y;
     m_t = 4;
   }
   double Square::Area() {
      return RectangleArea(m_tops[0], m_tops[1], m_tops[2],
m_tops[3]);
   }
   _2D Square::Center() {
      return RectangleCenter(m_tops[0], m_tops[1], m_tops[2],
m_tops[3]);
   }
   void Square::Output() {
      std::cout << "Координаты вершин:" << std::endl;
      std::cout << "\t" << "X" << "\t" << "Y" << std::endl;
      for (int number = 0; number < m_t; number++) {
           std::cout << "\t" << m_tops[number].x << "\t" <<
m_tops[number].y << std::endl;
      }
      std::cout << "Координаты центра:" << std::endl;
      std::cout << "\t" << "X" << "\t" << "Y" << std::endl;
      _2D m_center = Center();
      std::cout << "\t" << m_center.x << "\t" << m_center.y <<
std::endl;

      std::cout << "Площадь квадрата:" << std::endl;
      std::cout << "\t" << Area() << std::endl;
   }

//Rectangle kek

Rectangle::Rectangle(_2D A, _2D B, _2D C, _2D D){
     m_tops[0].x = A.x;            m_tops[0].y = A.y;
     m_tops[1].x = B.x;            m_tops[1].y = B.y;
     m_tops[2].x = C.x;            m_tops[2].y = C.y;
     m_tops[3].x = D.x;            m_tops[3].y = D.y;
```

```
    m_t = 4;
  }

  double Rectangle::Area() {
    return RectangleArea(m_tops[0], m_tops[1], m_tops[2],
m_tops[3]);
  }

  _2D Rectangle::Center() {
    return RectangleCenter(m_tops[0], m_tops[1], m_tops[2],
m_tops[3]);
  }

  void Rectangle::Output() {
    std::cout << "Координаты вершин:" << std::endl;
    std::cout << "\t" << "X" << "\t" << "Y" << std::endl;
    for (int number = 0; number < m_t; number++) {
        std::cout << "\t" << m_tops[number].x << "\t" <<
m_tops[number].y << std::endl;
    }
    std::cout << "Координаты центра:" << std::endl;
    std::cout << "\t" << "X" << "\t" << "Y" << std::endl;
    _2D m_center = Center();
    std::cout << "\t" << m_center.x << "\t" << m_center.y <<
std::endl;

    std::cout << "Площадь прямоугольника:" << std::endl;
    std::cout << "\t" << Area() << std::endl;

    if (IsSquare(m_tops[0], m_tops[1], m_tops[2], m_tops[3])) {
        std::cout << "Данный прямоугольник является
квадратом:" << std::endl;
    }
  }

  double TotalArea(std::vector<Figure*> &figures) {  // площадь
всех фигур из вектора
    double TotalArea = 0;
    for (uint n = 0; n < figures.size(); n++) { // обращаемся в
векторе
        if (figures[n]->m_t == 3) {                 // к
фигурам разных классов
            TotalArea += ((Triangle*)figures[n])->Area();//
(треугольник или четырехугольник)
        }
        if (figures[n]->m_t == 4) {
            TotalArea += ((Rectangle*)figures[n])->Area();
        }
    }
    return TotalArea;
  }
```

4. Объяснение результатов работы программы

Программа печатает в консоль меню, в которой описан весь возможный функционал: ввод различных фигур: треугольника, квадрата и прямоугольника по координатам, запись и хранение фигур в векторе указателей на фигуры, подсчет центров и площадей фигур, а также суммарной площади. Для решения данного задания было разработано 3 класса: класс вершин, фигур и фигур по заданию, которые наследуются от базового класса Figure, для каждого такого класса были переопределены функции нахождения центра, площади, а также вывод координат, при чем способ вычисления площади фигур находится по разному, в зависимости от типа фигуры.

5. Вывод

С помощью наследования программист может использовать универсальные классы и подстраивать их под себя, добавляя или изменяя функционал субкласса, для этого у программиста есть целый ряд функций и возможностей, например программист может переопределить virtual-методы субкласса так, как того требует задание, использовать данные и информацию уже описанного субкласса и добавлять к нему свои данные и методы.