Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 3

Тема: Наследование и Полиморфизм C++

Студент: Короткевич Л. В.
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

- Вычисление геометрического центра фигуры;
- Вывод в стандартный поток вывода std::cout координат вершин фигуры;
- 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 16:

1. Восьмиугольник
2. Треугольник
3. Квадрат

2. Описание программы

Figure.hpp	
virtual Point Barycenter() const = 0;	Вычисление геом. центра фигуры
virtual void PrintCoordinates() const = 0;	Вывод координат вершин фигуры
virtual double Area() const = 0;	Вычисление площади фигуры
virtual int GetType() const = 0;	Геттер типа фигуры (1 - 8-миугольник, 2 - треугольник, 3 - квадрат)
virtual std::ostream &PrintInfo(std::ostream &out) const = 0;	Вывод всей информации о фигуре
virtual std::istream &ReadInfo(std::istream &in) = 0;	Считывание информации (коорд. вершин)
virtual ~Figure() {}	Стандартный деструктор
Square.hpp	
Square();	Стандартный конструктор
Square(const Point &a, const Point &b, const Point &c, const Point &d);	Конструктор с 4 параметрами: точки квадрата
Point Barycenter() const;	Вычисление геом. центра фигуры
void PrintCoordinates() const;	Вывод координат вершин фигуры
double Area() const;	Вычисление площади фигуры
std::istream &ReadInfo(std::istream &in);	Считывание информации (коорд. вершин)
std::ostream &PrintInfo(std::ostream &out) const;	Вывод всей информации о фигуре
int GetType() const;	Геттер типа фигуры (1 - 8-миугольник, 2 - треугольник, 3 - квадрат)
~Square();	Стандартный деструктор
Octagon.hpp	
Octagon();	Стандартный конструктор
Octagon(const std::vector<Point> &pts)	Конструктор с 1 параметром: вектор точек 8-миугольника
Point Barycenter() const;	Вычисление геом. центра фигуры
void PrintCoordinates() const;	Вывод координат вершин фигуры
double Area() const;	Вычисление площади фигуры
std::istream &ReadInfo(std::istream &in);	Считывание информации (коорд. вершин)
std::ostream &PrintInfo(std::ostream &out) const;	Вывод всей информации о фигуре
int GetType() const;	Геттер типа фигуры (1 - 8-миугольник, 2 - треугольник, 3 - квадрат)
~Octagon();	Стандартный деструктор
Triangle.hpp	
Triangle();	Стандартный конструктор

Triangle(const Point &_A, const Point &_B, const Point &_C)	Конструктор с 3 параметрами: точки треуголльника
Point Barycenter() const;	Вычисление геом. центра фигуры
void PrintCoordinates() const;	Вывод координат вершин фигуры
double Area() const;	Вычисление площади фигуры
std::istream &ReadInfo(std::istream &in);	Считывание информации (коорд. вершин)
std::ostream &PrintInfo(std::ostream &out) const;	Вывод информации о фигуре
int GetType() const;	Геттер типа фигуры (1 - 8-миугольник, 2 - треугольник, 3 - квадрат)
~Triangle();	Стандартный деструктор
Point.hpp	
std::istream &operator>>(std::istream &in, Point &pt);	Считывание координат точки
std::ostream &operator<<(std::ostream &out, const Point &pt);	Вывод координат точки
inline double DistanceSquared(Point p, Point q);	Расстояние между точками в квадрате
inline double Cross(Point p, Point q);	Векторное произведение
int ret[2][2] = {{0, 3}, {1, 2}};	Массив номеров четвертей в декартовой системе координат
inline int Quad(Point p);	Определение четверти точки в декартовой системе координат
bool byAngle(const Point &a, const Point &b);	Функция для сортировки вектора точек по полярному углу

3. Набор тестов

Test #1	Пояснение	Test #2	Пояснение	Test #3	Пояснение
1 1	Добавить 8-миугольник	1 2	Добавить треуг.	1 2	Добавить треуг.
2 0	Координаты	1 0	Координаты	1 0	Координаты
4 0		0 1		0 1	
6 2		1 1		2 3	
6 4		1 2	Добавить треуг.	1 3	Добавить квадрат
0 4		1 1	Координаты	-1 -1	Координаты
4 6		2 1		-1 1	
2 6		1.5 -1		1 1	
0 2		1 2	Добавить треуг.	1 -1	
1 2	Добавить треугольник	0 0	Координаты	1 3	Добавить квадрат
0 0	Координаты	1 1		1 1	Координаты
1 1		2 0		2 0	
2 3		3	Вывести таблицу	3 1	

1 3	Добавить квадрат	4 0	Вывод ГЦ 0-й фигуры	2 2	
0 100	Кооординаты	4 1	Вывод ГЦ 1-й фигуры	3	Вывод таблицы
100 0		4 2	Вывод ГЦ 2-й фигуры	7 0	Информация о 0-й фигуре
100 100		5 0	Вывод коорд. вершин 0-й фигуры	7 1	Информация о 1-й фигуре
0 0		5 1	Вывод коорд. вершин 1-й фигуры	7 2	Информация о 2-й фигуре
1 1	Добавить 8-миугольник	5 2	Вывод коорд. вершин 2-й фигуры	8	Суммарная площадь фигур
1 0	Координаты	6 0	Вывод площади 0-й фигуры	9	Выход
2 0		6 1	Вывод площади 1-й фигуры		
3 1		6 2	Вывод площади 2-й фигуры		
3 2		9	Выход		
2 3					
1 3					
0 2					
0 1					
3	Вывести таблицу				
2 0	Удалить 0-й элемент				
3	Вывести таблицу				
9	Выход				

4. Результаты выполнения тестов

[leo@pc ЛР3]$ cat out1

1) Add figure (type <1 1> to add Octagon, <1 2> - Triangle, <1 3> - Square)
2) Remove figure
3) Display elements
4) Display barycenter of figure
5) Display points of figure
6) Display area of figure
7) Display all information about the figure
8) Display total area
9) Exit
Octagon succesfully added at index 0
Triangle succesfully added at index 1
Square succesfully added at index 2
Octagon succesfully added at index 3

```
Index   Type            Area    Coordinates
0       Octagon         28      (2, 0), (4, 0), (6, 2), (6, 4), (4, 6), (2, 6), (0, 4), (0, 2)
1       Triangle        0.5     (0, 0), (1, 1), (2, 3)
2       Square          10000   (0, 100), (100, 0), (100, 100), (0, 0)
3       Octagon         7       (1, 0), (2, 0), (3, 1), (3, 2), (2, 3), (1, 3), (0, 2), (0, 1)
Octagon successfully removed
Index   Type            Area    Coordinates
0       Triangle        0.5     (0, 0), (1, 1), (2, 3)
1       Square          10000   (0, 100), (100, 0), (100, 100), (0, 0)
2       Octagon         7       (1, 0), (2, 0), (3, 1), (3, 2), (2, 3), (1, 3), (0, 2), (0, 1)
```
[leo@pc ЛР3]$ cat out2
```
1) Add figure (type <1 1> to add Octagon, <1 2> - Triangle, <1 3> - Square)
2) Remove figure
3) Display elements
4) Display barycenter of figure
5) Display points of figure
6) Display area of figure
7) Display all information about the figure
8) Display total area
9) Exit
Triangle succesfully added at index 0
Triangle succesfully added at index 1
Triangle succesfully added at index 2
Index   Type            Area    Coordinates
0       Triangle        0.5     (1, 0), (0, 1), (1, 1)
1       Triangle        1       (1, 1), (2, 1), (1.5, -1)
2       Triangle        1       (0, 0), (1, 1), (2, 0)
Triangle's barycenter: (0.666667, 0.666667)
Triangle's barycenter: (1.5, 0.333333)
Triangle's barycenter: (1, 0.333333)
Triangle's points:      (1, 0), (0, 1), (1, 1)
Triangle's points:      (1, 1), (2, 1), (1.5, -1)
Triangle's points:      (0, 0), (1, 1), (2, 0)
Triangle's area: 0.5
Triangle's area: 1
Triangle's area: 1
```
[leo@pc ЛР3]$ cat out3
```
1) Add figure (type <1 1> to add Octagon, <1 2> - Triangle, <1 3> - Square)
2) Remove figure
3) Display elements
4) Display barycenter of figure
5) Display points of figure
6) Display area of figure
7) Display all information about the figure
8) Display total area
9) Exit
Triangle succesfully added at index 0
Square succesfully added at index 1
Square succesfully added at index 2
Index   Type            Area    Coordinates
0       Triangle        2       (1, 0), (0, 1), (2, 3)
1       Square          4       (-1, -1), (-1, 1), (1, 1), (1, -1)
2       Square          2       (1, 1), (2, 0), (3, 1), (2, 2)
Triangle info:
* Points:      (1, 0), (0, 1), (2, 3)
* Sides:       1.41421, 3.16228, 2.82843
* Area:        2
```

```
* Barycenter:   ((1, 1.33333)
Square info:
* Points:      (-1, -1), (-1, 1), (1, 1), (1, -1)
* Sides:       2
* Area:        4
* Barycenter:   (0, 0)
Square info:
* Points:      (1, 1), (2, 0), (3, 1), (2, 2)
* Sides:       1.41421
* Area:        2
* Barycenter:   (2.29289, 1.29289)
Total area: 8
```

5. Листинг программы

main.cpp

```
/*
Короткевич Л. В.
М8О-208Б-19
github.com/anxieuse/oop_exercise_03
Вариант 16:
   8-угольник
   Треугольник
   Квадрат
*/

#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <string>

#include "Point.hpp"
#include "Figure.hpp"
#include "Square.hpp"
#include "Triangle.hpp"
#include "Octagon.hpp"

void printHelp()
{
   std::cout << "1) Add figure (type <1 1> to add Octagon, <1 2> - Triangle, <1 3> - Square)\n";
   std::cout << "2) Remove figure\n";
   std::cout << "3) Display elements\n";
   std::cout << "4) Display barycenter of figure\n";
   std::cout << "5) Display points of figure\n";
   std::cout << "6) Display area of figure\n";
   std::cout << "7) Display all information about the figure\n";
   std::cout << "8) Display total area\n";
   std::cout << "9) Exit\n";
}

int main()
{
```

```
printHelp();

std::vector<Figure *> figures;
unsigned long long totalArea = 0;

bool input = true;
int cmdID, figureType, ID;

Square *sampleSq;
Octagon *sampleOc;
Triangle *sampleTr;

while (input)
{
   std::cin >> cmdID;
   switch (cmdID)
   {
   case 1: // add
   {
      std::cin >> figureType;
      if (figureType == 1)
      {
         sampleOc = new Octagon;
         std::cin >> *sampleOc;
         figures.push_back(sampleOc);
      }
      if (figureType == 2)
      {
         sampleTr = new Triangle;
         std::cin >> *sampleTr;
         figures.push_back(sampleTr);
      }
      if (figureType == 3)
      {
         sampleSq = new Square;
         std::cin >> *sampleSq;
         figures.push_back(sampleSq);
      }
      totalArea += figures.back()->Area();
      std::cout << nameByType[figureType - 1] << " succesfully added at index " << figures.size() - 1 << '\n';
      break;
   }
   case 2: // remove
   {
      std::cin >> ID;
      if (ID > figures.size())
      {
         std::cout << "ID is too big. There are no such many elements!\n";
      }
      else
      {
         std::cout << nameByType[figures[ID]->GetType() - 1] << " successfully removed\n";
```

```
            delete figures[ID];
            figures.erase(figures.begin() + ID);
        }
        break;
    }
    case 3: // display elements
    {
        std::cout << "Index\tType\t\tArea\tCoordinates\n";
        for (int i = 0; i < figures.size(); ++i)
        {
            std::cout << i << '\t' << nameByType[figures[i]->GetType() - 1] << "\t";
            if (nameByType[figures[i]->GetType() - 1] != "Triangle")
                std::cout << "\t";
            std::cout << figures[i]->Area() << '\t';
            figures[i]->PrintCoordinates();
        }
        break;
    }
    case 4: // barycenter
    {
        std::cin >> ID;
        if (ID > figures.size())
        {
            std::cout << "ID is too big. There are no such many elements!\n";
        }
        else
        {
                std::cout << nameByType[figures[ID]->GetType() - 1] << "'s barycenter: " << figures[ID]-
>Barycenter() << '\n';
        }
        break;
    }
    case 5: // coordinates
    {
        std::cin >> ID;
        if (ID > figures.size())
        {
            std::cout << "ID is too big. There are no such many elements!\n";
        }
        else
        {
            std::cout << nameByType[figures[ID]->GetType() - 1] << "'s points:\t";
            figures[ID]->PrintCoordinates();
        }
        break;
    }
    case 6: // area
    {
        std::cin >> ID;
        if (ID > figures.size())
        {
            std::cout << "ID is too big. There are no such many elements!\n";
```

```
            }
            else
            {
                std::cout << nameByType[figures[ID]->GetType() - 1]
                        << "'s area: " << figures[ID]->Area() << '\n';
            }
            break;
        }
        case 7: // info
        {
            std::cin >> ID;
            if (ID > figures.size())
            {
                std::cout << "ID is too big. There are no such many elements!\n";
            }
            else
            {
                std::cout << *figures[ID];
            }
            break;
        }
        case 8: // total area
        {
            std::cout << "Total area: " << totalArea << '\n';
            break;
        }
        case 9: // end
        {
            input = false;
            break;
        }
        }
    }
    for(auto &x : figures)
        delete x;
    return 0;
}
```

Point.hpp:

```
#ifndef POINT_HPP
#define POINT_HPP

struct Point
{
    double x, y;
};

std::istream &operator>>(std::istream &in, Point &pt)
{
    in >> pt.x >> pt.y;
    return in;
}
```

```
std::ostream &operator<<(std::ostream &out, const Point &pt)
{
   out << "(" << pt.x << ", " << pt.y << ")";
   return out;
}

inline double DistanceSquared(Point p, Point q)
{
   return (p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.y - q.y);
}

inline double Cross(Point p, Point q)
{
   return p.x * q.y - q.x * p.y;
}

int ret[2][2] = {{0, 3}, {1, 2}};

inline int Quad(Point p)
{
   return ret[p.x >= 0][p.y >= 0];
}

bool byAngle(const Point &a, const Point &b)
{
   return Quad(a) == Quad(b) ? Cross(a, b) > 0 : Quad(a) < Quad(b);
}

#endif
```

Figure.hpp:

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

std::vector<std::string> nameByType = {"Octagon", "Triangle", "Square"};

class Figure
{
public:
   virtual Point Barycenter() const = 0;
   virtual void PrintCoordinates() const = 0;
   virtual double Area() const = 0;
   virtual int GetType() const = 0;
   virtual std::ostream &PrintInfo(std::ostream &out) const = 0;
   virtual std::istream &ReadInfo(std::istream &in) = 0;
   virtual ~Figure() {}
};

std::ostream &operator<<(std::ostream &out, const Figure &fg)
{
   return fg.PrintInfo(out);
```

```
}

std::istream &operator>>(std::istream &in, Figure &fg)
{
   return fg.ReadInfo(in);
}

#endif
```

Square.hpp:

```
#ifndef SQUARE_HPP
#define SQUARE_HPP

#include "Figure.hpp"

class Square : public Figure
{
public:
   Square()
   {
      A = B = C = D = {0, 0};
      side = 0;
   }
   Square(const Point &a, const Point &b, const Point &c, const Point &d)
   {
      A = a, B = b, C = c, D = d;
      side = sqrt(std::min(DistanceSquared(A, B), DistanceSquared(A, C)));
   }
   Point Barycenter() const
   {
      double max_x = std::max({A.x, B.x, C.x, D.x});
      double max_y = std::max({A.y, B.y, C.y, D.y});
      return {max_x - side / 2.0, max_y - side / 2.0};
   }
   void PrintCoordinates() const {
      std::cout << A << ", " << B << ", " << C << ", " << D << '\n';
   }
   double Area() const
   {
      return side * side;
   }
   std::istream &ReadInfo(std::istream &in)
   {
      in >> A >> B >> C >> D;
      (*this) = Square(A, B, C, D);
      return in;
   }
   std::ostream &PrintInfo(std::ostream &out) const
   {
      out << "Square info: \n";
      out << "* Points:\t" << A << ", " << B << ", " << C << ", " << D << "\n";
      out << "* Sides:\t" << side << "\n";
```

```
        out << "* Area:\t\t" << Area() << '\n';
        out << "* Barycenter:\t" << Barycenter() << '\n';
        return out;
    }
    int GetType() const {
        return type;
    }
    ~Square() {}

private:
    Point A, B, C, D;
    double side;
    int type = 3;
};

#endif
```

Triangle.hpp:

```
#ifndef TRIANGLE_HPP
#define TRIANGLE_HPP

#include "Figure.hpp"

class Triangle : public Figure
{
public:
    Triangle()
    {
        A = B = C = {0, 0};
        a = b = c = 0;
    }
    Triangle(const Point &_A, const Point &_B, const Point &_C)
    {
        A = _A, B = _B, C = _C;
        a = sqrt(DistanceSquared(A, B));
        b = sqrt(DistanceSquared(A, C));
        c = sqrt(DistanceSquared(B, C));
    }
    Point Barycenter() const
    {
        double cx = (A.x + B.x + C.x) / 3.0;
        double cy = (A.y + B.y + C.y) / 3.0;
        return {cx, cy};
    }
    void PrintCoordinates() const {
        std::cout << A << ", " << B << ", " << C << '\n';
    }
    double Area() const
    {
        double p = 0.5 * (a + b + c);
        return sqrt(p * (p - a) * (p - b) * (p - c));
    }
```

```
    std::istream &ReadInfo(std::istream &in)
    {
        in >> A >> B >> C;
        (*this) = Triangle(A, B, C);
        return in;
    }
    std::ostream &PrintInfo(std::ostream &out) const {
        out << "Triangle info: \n";
        out << "* Points:\t" << A << ", " << B << ", " << C << '\n';
        out << "* Sides:\t" << a << ", " << b << ", " << c << '\n';
        out << "* Area:\t\t" << Area() << '\n';
        out << "* Barycenter:\t(" << Barycenter() << '\n';
        return out;
    }
    int GetType() const {
        return type;
    }
    ~Triangle() {}

private:
    Point A, B, C;
    double a, b, c;
    int type = 2;
};

#endif
```

Octagon.hpp:

```
#ifndef OCTAGON_HPP
#define OCTAGON_HPP

#include "Figure.hpp"

class Octagon : public Figure
{
public:
    Octagon()
    {
        points.resize(8);
        std::fill(points.begin(), points.end(), Point{0, 0});
        sides.resize(8);
        std::fill(sides.begin(), sides.end(), 0.0);
    }
    Octagon(const std::vector<Point> &pts)
    {
        points.resize(8);
        points = pts;
        std::sort(points.begin(), points.end(), byAngle);

        sides.resize(8);
        for (int i = 1; i < 8; ++i)
        {
```

```
        sides[i] = sqrt(DistanceSquared(points[i], points[i - 1]));
    }
    sides[0] = sqrt(DistanceSquared(points[7], points[0]));
}
Point Barycenter() const
{
    double sumx = 0, sumy = 0;
    for (auto &pt : points)
    {
        sumx += pt.x;
        sumy += pt.y;
    }
    return {sumx / 8.0, sumy / 8.0};
}
void PrintCoordinates() const
{
    for (int i = 0; i < 8; ++i)
    {
        std::cout << points[i];
        if(i != 7) std::cout << ", ";
    }
    std::cout << '\n';
}
double Area() const
{
    // Вычисление площади с помощью векторного произведения
    double area = 0;
    for (int i = 1; i < 8; ++i)
    {
        area += Cross(points[i - 1], points[i]);
    }
    area += Cross(points[7], points[0]);
    return fabs(area) / 2.0;
}
std::istream &ReadInfo(std::istream &in)
{
    for (auto &pt : points)
    {
        in >> pt;
    }
    (*this) = Octagon(points);
    return in;
}
std::ostream &PrintInfo(std::ostream &out) const
{
    out << "Octagon info:\n";
    out << "* Points:\t";
    for (int i = 0; i < 8; ++i)
    {
        out << points[i];
        if (i != 7)
            out << ", ";
```

```
        }
        out << '\n';
        out << "* Sides:\t";
        for (int i = 0; i < 8; ++i)
        {
            out << sides[i];
            if (i != 7)
                out << ", ";
        }
        out << '\n';
        out << "* Area:\t\t" << Area() << '\n';
        out << "* Barycenter:\t" << Barycenter() << '\n';
        return out;
    }
    int GetType() const
    {
        return type;
    }
    ~Octagon() {}

private:
    std::vector<Point> points;
    std::vector<double> sides;
    double side;
    int type = 1;
};

#endif
```