Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

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

Москва, 2019

1. **Тема**: Наследование, полиморфизм
2. **Цель работы**: Изучение механизмов работы с наследованием в C++
3. **Задание** (*вариант № 7*):

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

- Вычисление геометрического центра фигуры
- Вывод в стандартный поток std::cout координат вершин фигуры
- Вычисление площади фигуры

Создать программу, которая позволяет:

- Вводить из стандартного вода std::cin фигуры, согласно варианту задания
- Сохранять созданные фигуры в динамический массив std::vector<Figure*>
- Вызывыть для всего массива общие функции (1 – 3)
- Необходимо уметь вычислять общую площадь фигур в массиве
- Удалять из массива фигуру по индексу

Фигуры (Вариант 7):

Треугольник, 6-угольник, 8-угольник.

4. **Адрес репозитория на GitHub** https://github.com/DragonKeker/oop_exercise_03
5. **Код программы на C++**

main.cpp

```
#include <iostream>
#include "figure.h"
#include "triangle.h"
#include "octagon.h"
#include "hexagon.h"
#include <vector>
#include <string>

void read_figt(std::vector<Figure*>& fig)
{
	int figt;
	Figure* f = nullptr;

	std::cout << "Fig types: 1 - triangle; 2 - hexagon; 3 - octagon\n";
	std::cin >> figt;
	if (figt == 1) {
		f = new Triangle(std::cin);
	}
	else if (figt == 2) {
		f = new Hexagon(std::cin);
	}
	else if (figt == 3) {
	f = new Octagon(std::cin);

}else{
	std::cout << "Wrong. Try 1 - triangle, 2 - hexagon or 3 - octagon\n";
		}
	fig.push_back(dynamic_cast<Figure*>(f));
	}


int main() {
	unsigned int index;
```

```
	double Tarea = 0;
	std::string operation;
	std::vector<Figure*> fig;

	std::cout << "Operations: add / delete / out / quit\n";

	while (std::cin >> operation) {
		if (operation == "add") {
			read_figt(fig);
		}
		else if (operation == "delete") {
			std::cin >> index;
			delete fig[index];
			for (; index < fig.size() - 1; ++index) {
				fig[index] = fig[index + 1];
			}
			fig.pop_back();
		}
		else if (operation == "out") {
			Tarea = 0;
			for (unsigned int i = 0; i < fig.size(); i++) {
				std::cout << i << ":\n";
				std::cout << "Area: " << fig[i]->area() << std::endl;
				std::cout << "Center: " << fig[i]->center() << std::endl;
				std::cout << "Coordinates: ";
				fig[i]->print(std::cout);
				std::cout << std::endl;
				Tarea += fig[i]->area();
			}
			std::cout << "Total area: " << Tarea << std::endl;
		}
		else if (operation == "quit") {
			for (unsigned int i = 0; i < fig.size(); ++i) {
				delete fig[i];
			}
			return 0;
		}
		else {
			std::cout << "Wrong. Operations: add / delete / out / quit\n";
		}
	}
}
```

point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
	Point();
	Point(double x, double y);
	double X() const;
	double Y() const;
	friend std::ostream& operator<< (std::ostream& out, const Point& p);
	friend std::istream& operator>> (std::istream& in, Point& p);
private:
	double x;
	double y;
};
#endif
```

point.cpp

```cpp
#include "point.h"
#include <cmath>

Point::Point() : x{ 0 }, y{ 0 }
{}

Point::Point(double x, double y) : x{ x }, y{ y }
{}

double Point::X() const
{
	return x;
}

double Point::Y() const
{
	return y;
}

std::ostream& operator<< (std::ostream& out, const Point& p)
{
	out << "(" << p.X() << ";" << p.Y() << ")";
	return out;
}


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

figure.h

```cpp
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
#include "point.h"

class Figure {
public:
	virtual double area() const = 0;
	virtual Point center() const = 0;
	virtual std::ostream& print(std::ostream& out) const = 0;
	virtual ~Figure() = default;
};

#endif
```

octagon.h

```cpp
#ifndef OCTAGON_H
#define OCTAGON_H

#include "figure.h"
#include "point.h"

class Octagon : public Figure {
public:
	Octagon();
```

```
	Octagon(std::istream& in);
	double area() const override;
	Point center() const override;
	std::ostream& print(std::ostream& out) const override;
private:
	Point A;
	Point B;
	Point C;
	Point D;
	Point E;
	Point F;
	Point G;
	Point H;
};
#endif
```

octagon.cpp

```
#include "octagon.h"
#include <cmath>

Octagon::Octagon() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0, 0 }, E{ 0, 0 }, F{ 0, 0 }, G{
0, 0 }, H{ 0, 0 }  {}

Octagon::Octagon(std::istream& in) {
	in >> A >> B >> C >> D >> E >> F >> G >> H;
}


double Octagon::area() const {
	return 0.5* abs(A.X() * B.Y() + B.X() * C.Y() + C.X() * D.Y() + D.X() * E.Y() +
E.X() * F.Y() + F.X() * G.Y() + G.X() * H.Y() + H.X() * A.Y()
		- B.X() * A.Y() - C.X() * B.Y() - D.X() * C.Y() - E.X() * D.Y() - F.X() *
E.Y() - G.X() * F.Y() - H.X() * G.Y() - A.X() * H.Y());
}

Point Octagon::center() const {
	return Point{ (A.X() + B.X() + C.X() + D.X() + E.X() + F.X() + G.X() + H.X()) / 8,
(A.Y() + B.Y() + C.Y() + D.Y() + E.Y() + F.Y() + G.Y() + H.Y()) / 8 };
}

std::ostream& Octagon::print(std::ostream& out) const {
	out << A << " " << B << " " << C << " " << D << " " << E << " " << F << " " << G
<< " " << H;
	return out;
}
```

hexagon.h

```
#ifndef HEXAGON_H
#define HEXAGON_H

#include "figure.h"
#include "point.h"

class Hexagon : public Figure {
public:
	Hexagon();
	Hexagon(std::istream& in);
	double area() const override;
	Point center() const override;
	std::ostream& print(std::ostream& out) const override;
private:
```

```
        Point A;
        Point B;
        Point C;
        Point D;
        Point E;
        Point F;
};
#endif
```

hexagon.cpp

```
#include "hexagon.h"
#include <cmath>

Hexagon::Hexagon() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0, 0 }, E{ 0, 0 }, F{ 0, 0 } {}

Hexagon::Hexagon(std::istream& in) {
        in >> A >> B >> C >> D >> E >> F;
}


double Hexagon::area() const {
        return 0.5 * abs(A.X() * B.Y() + B.X() * C.Y() + C.X() * D.Y() + D.X() * E.Y() +
E.X() * F.Y() + F.X() * A.Y()
                - B.X() * A.Y() - C.X() * B.Y() - D.X() * C.Y() - E.X() * D.Y() - F.X() *
E.Y() - A.X() * F.Y());
}

Point Hexagon::center() const {
        return Point{ (A.X() + B.X() + C.X() + D.X() + E.X() + F.X()) / 6, (A.Y() + B.Y()
+ C.Y() + D.Y() + E.Y() + F.Y()) / 6 };
}

std::ostream& Hexagon::print(std::ostream& out) const {
        out << A << " " << B << " " << C << " " << D << " " << E << " " << F;
        return out;
}
```

triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H

#include "figure.h"
#include "point.h"

class Triangle : public Figure {
public:
        Triangle();
        Triangle(std::istream& in);
        double area() const override;
        Point center() const override;
        std::ostream& print(std::ostream& out) const override;
private:
        Point A;
        Point B;
        Point C;
};
#endif
```

triangle.cpp

```
#include "triangle.h"
```

```
#include <cmath>

Triangle::Triangle() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 } {}

Triangle::Triangle(std::istream& in) {
	in >> A >> B >> C;
}


double Triangle::area() const {
	return 0.5 * abs(A.X() * B.Y() + B.X() * C.Y() + C.X() * A.Y() - B.X() * A.Y() -
C.X() * B.Y() - A.X() * C.Y());
}

Point Triangle::center() const
{
	return Point{ (A.X() + B.X() + C.X()) / 3, (A.Y() + B.Y() + C.Y()) / 3 };
}

std::ostream& Triangle::print(std::ostream& out) const
{
	out << A << " " << B << " " << C;
	return out;
}
```

CMakeLists.txt

```
cmake_minimum_required (VERSION 3.5)

project(Lab3)

add_executable(oop_exercise_03
  main.cpp
  figure.cpp
  triangle.cpp
  point.cpp
  hexagon.cpp
  octagon.cpp)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")

set_target_properties(oop_exercise_03 PROPERTIES CXX_STANDART 14 CXX_STANDART_REQUIRED
ON)
```

6. Результаты выполнения тестов

ТЕСТ 1

Operations: add / delete / out / quit
add
Fig types: 1 - triangle; 2 - hexagon; 3 - octagon
1
0 0
2 2
0 1
add
Fig types: 1 - triangle; 2 - hexagon; 3 - octagon
2
0 0
1 1
2 2
3 3

4 4
5 5
add
Fig types: 1 - triangle; 2 - hexagon; 3 - octagon
3
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
out
0:
Area: 1
Center: (0.666667;1)
Coordinates: (0;0) (2;2) (0;1)
1:
Area: 0
Center: (2.5;2.5)
Coordinates: (0;0) (1;1) (2;2) (3;3) (4;4) (5;5)
2:
Area: 0
Center: (3.5;4.5)
Coordinates: (0;1) (1;2) (2;3) (3;4) (4;5) (5;6) (6;7) (7;8)
Total area: 1

ТЕСТ 2

Operations: add / delete / out / quit
add
Fig types: 1 - triangle; 2 - hexagon; 3 - octagon
1
0 0
2 2
0 1
add
Fig types: 1 - triangle; 2 - hexagon; 3 - octagon
2
0 0
1 1
2 2
3 3
4 4
5 5
add
Fig types: 1 - triangle; 2 - hexagon; 3 - octagon
3
0 1
1 2
2 3
3 4
4 5

5 6
6 7
7 8
out
0:
Area: 1
Center: (0.666667;1)
Coordinates: (0;0) (2;2) (0;1)
1:
Area: 0
Center: (2.5;2.5)
Coordinates: (0;0) (1;1) (2;2) (3;3) (4;4) (5;5)
2:
Area: 0
Center: (3.5;4.5)
Coordinates: (0;1) (1;2) (2;3) (3;4) (4;5) (5;6) (6;7) (7;8)
Total area: 1
quit
C:\Users\Андрей\source\repos\лаба 3.1\Debug\лаба 3.1.exe (процесс 11504) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу…

7. Объяснение результатов работы программы - вывод

Изначально создается базовый класс Figure задающий общий принцип структуры для классов — наследников(треугольника, 6-угольника и 8-угольника).
Наследование позволяет избежать дублирования лишнего кода при написании классов, т. к. класс может использовать переменные и методы другого класса как свои собственные.
В данном случае класс Figure является абстрактным — он определяет интерфейс для переопределения методов другими классами.
Также в данной лаборторной работе используется полиморфизм он осуществляется посредством виртуальных функций.
Эта лабораторная работа научила меня использовать наследование классов.