Задание 8

Темы: классы, иерархия наследования, полиморфизм, виртуальные функции

Реализовать классы, образующие иерархию наследования. Классы упрощенно описывают геометрические фигуры. Иерархия наследования представлена на рис. 1.

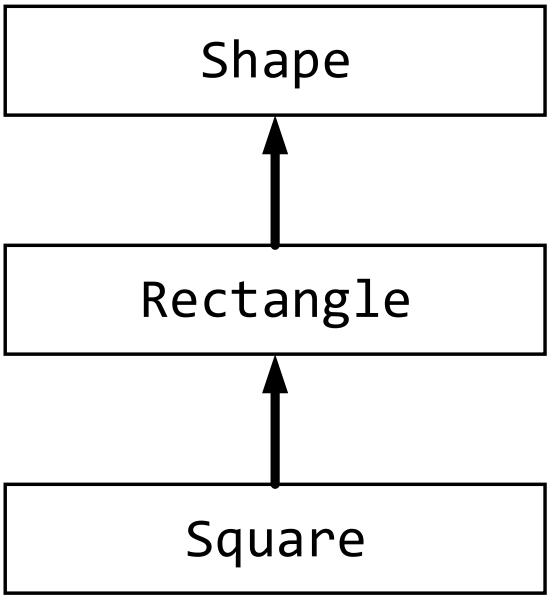

Рис. 1 – Иерархия наследования

Класс Shape

Родительский класс **Shape** содержит поле, хранящее площадь фигуры. Данное поле инициализируется при создании фигуры. Значение данного поля может быть получено путем вызова открытого метода **GetArea**.

```
class Shape
{
private:
    double area;
public:
    Shape();
    Shape(double Area);
    Shape(const Shape& Other);
    virtual ~Shape();
    virtual void draw();
    void print();
    double GetArea();
};
```

Функционал класса Shape:

Конструктор по умолчанию – создает объект и инициализирует поле area нулевым значением.

Конструктор с параметром – создает объект и инициализирует поле area значением, преданным в качестве аргумента.

Конструктор копирования – создает новый объект на основе существующего.

Деструктор – корректно освобождает ресурсы, занятые объектом (если нужно).

Метод GetArea – возвращает значение площади фигуры для данного объекта.

Метод draw – выводит на экран имя класса и имя метода.

Метод print – выводит на экран имя класса и имя метода.

Пример реализации метода **draw**:

```
void Shape::draw() { std::cout << “Shape::draw” << std::endl;}
```

Класс Rectangle

Данный класс также содержит длину (a) и ширину (b) прямоугольника. При создании экземпляров данного класса должна корректно инициализироваться базовая часть (**Shape**). Площадь фигуры, передаваемая в базовую часть, вычисляется как a*b.

```
class Rectangle : public Shape
{
private:
    double a;
    double b;
public:
    Rectangle();
    Rectangle(double A, double B);
    Rectangle(const Rectangle& Other);
    ~Rectangle();
    void draw();
    void print();
};
```

Функционал класса Rectangle:

Конструктор по умолчанию – создает объект и инициализирует длину и ширину нулевыми значениями. При этом должна корректно инициализироваться базовая часть класса.

Конструктор с параметром – создает объект и инициализирует длину и ширину прямоугольника значениями, преданными в качестве аргументов. При этом должна корректно инициализироваться базовая часть класса.

Конструктор копирования – создает новый объект на основе существующего. При этом должна корректно инициализироваться базовая часть класса. Базовая часть класса должна инициализироваться с использованием конструктора копирования базового класса.

Деструктор – корректно освобождает ресурсы, занятые объектом (если нужно).

Метод draw – выводит на экран имя класса и имя метода.

Метод print – выводит на экран имя класса и имя метода.

Класс Square

```
class Square : public Rectangle
{
public:
    Square();
    Square(double C);
    Square(const Square& Other);
    ~Square();
    void draw();
    void print();
};
```

Функционал класса Square:

Конструктор по умолчанию – создает объект и инициализирует его стороны нулевыми значениями. При этом должна корректно инициализироваться базовая часть класса.

Конструктор с параметром – создает объект и инициализирует его стороны значением, преданным в качестве аргумента. При этом должна корректно инициализироваться базовая часть класса (**Rectangle**).

Конструктор копирования – создает новый объект на основе существующего. При этом должна корректно инициализироваться базовая часть класса. Базовая часть класса должна инициализироваться с использованием конструктора копирования базового класса.

Деструктор – корректно освобождает ресурсы, занятые объектом (если нужно).

Метод draw – выводит на экран имя класса и имя метода.

Метод print – выводит на экран имя класса и имя метода.

Задание

Реализовать классы **Shape**, **Rectangle** и **Square**, входящие в иерархию наследования, представленную на рис. 1. Конструкторы и деструктор всех классов должны содержать отладочную печать с указанием имени класса и имени вызываемого метода.

Объявление каждого класса должно находиться в заголовочном файле, имя которого совпадает с именем класса. Реализация методов класса должна находиться в файле исходного кода, имя которого совпадает с именем класса. Например, объявление класса **Shape** должно находиться в заголовочном файле Shape.h, реализация – в Shape.cpp.

Изменение спецификаторов доступа для полей класса не допускается!

Реализовать три глобальных функции **Draw1**, **Draw2** и **Draw3**, принимающие экземпляры класса **Shape** по значению, указателю и ссылке. Внутри данных функций осуществить вызов методов **draw** и **print**. В функции **Draw2** также осуществить вызов метода **draw**, который независимо от типа переданного аргумента всегда будет выполняться для базового класса (**Shape**).

```
void Draw1(Shape S) { ... }

void Draw2(Shape* S) { ... }

void Draw3(Shape& S) { ... }
```

Передать в каждую из функций **Draw1**, **Draw2** и **Draw3** объекты типов **Shape**, **Rectangle** и **Square** и объяснить для какого типа вызывается виртуальная функция **draw** и невиртульная **print** в каждом случае и почему. Объекты, передаваемые в функции **Draw1**, **Draw2** и **Draw3**, должны быть инициализированы с помощью различных конструкторов (по умолчанию, с параметрами и копирования). Пример:

```
Rectangle r1(1,2);
Draw1(r1);
Draw2(&r1);
Draw3(r1);
//Аналогично для объектов типа Shape и Square
```

Создать объект одного из дочерних классов (**Rectangle** или **Square**) в динамической области памяти и сохранить его адрес в указателе на базовый класс (**Shape**). Затем осуществить удаление объекта из памяти. Объяснить порядок вызова деструкторов. Пронаблюдать за порядком вызова деструкторов, когда деструктор объявлен как невиртуальный. Пример:

```
Shape* p1 = new Rectangle(1, 2);
Shape* p2 = new Square(3);
...
delete p1;
delete p2;
```

Вопросы для самопроверки:

1. Дайте определение раннему связыванию.
2. Дайте определение позднему связыванию.
3. Какими достоинствами обладает раннее связывание?
4. Какими достоинствами обладает позднее связывание?
5. Какими недостатками обладает раннее связывание?
6. Какими недостатками обладает позднее связывание?
7. Дайте определение полиморфизму.
8. Для чего используются виртуальные функции?
9. Какие методы класса рекомендуется всегда делать виртуальными, при включении класса в иерархию наследования?
10. Для чего используются чисто виртуальные функции?
11. Какие классы называются абстрактными?
12. Для чего используются абстрактные классы?
13. Как связаны абстрактные классы и интерфейсы в C++?
14. Дайте определение интерфейсу.
15. Объясните механизм вызова виртуальных функций.
16. Какие методы класса не могут быть виртуальными?
17. При каких условиях будет работать полиморфизм?
18. В каком случае будет раннее, а в каком случае позднее связывание при вызове метода **draw**, объявленного с ключевым словом `virtual`? Поясните свой выбор.

```
Rectangle r(1,2);
```

a. `r.draw();`
b. `(&r)->draw();`

19. В каком случае не возникнет проблем с освобождением ресурсов, если деструктор объявлен как невиртуальный? Поясните свой выбор.

a. `Shape s = Rectangle(1, 2);`
b. `Shape* s = new Rectangle(1, 2);`
c. `Rectangle* r = new Rectangle(1, 2);`
d. `Square s = Rectangle(1, 2);`