Лабораторная работа № **25**

«Разработка алгоритмов и программ с использованием виртуальных функций (виртуальных методов)»

Полиморфизм – (от греч. poli – много, morpheo – форма, поведение) – один из трех основных принципов ООП, заключающийся в возможности определения единого по имени действия (поведения), применимого ко всем объектам иерархии наследования (иерархии классов), причем каждый объект может реализовать это действие (поведение) собственным способом (в зависимости от того, к какому именно классу из данной иерархии этот объект относится).

Виртуальные функции

Если мы говорим о виртуальных функциях внутри классов, то логично их называть **виртуальные методы** класса. Виртуальными могут быть методы и деструкторы, но **не** конструкторы классов.

Виртуальные методы – это методы, объявляемые в родительском классе с ключевым словом virtual и переопределяемые (перегружаемые) в дочерних классах. При вызове виртуального метода в момент выполнения программы определяется тип (имя класса) конкретного объекта, у которого вызван виртуальный метод и вызывается для работы метод именно данного определенного класса (в данной иерархии все классы имеют метод с таким именем и сигнатурой, но тела у методов разные). Это происходит не при компиляции программы, при исполнении программы, поскольку только тогда будет точно известно, объект какого именно класса вызывает свой виртуальный метод. Это возможно благодаря механизму позднего связывания в программе – окончательная проверка типов будет проводиться уже на этапе выполнения программы, поскольку на этапе компиляции может быть не известно, на какой объект будет указывать указатель родительского класса (это может определяться логикой работы программы или даже пользователем через меню, которое можно реализовать, например, структурой switch-case. Виртуальный метод позволяет в коде программы работать с объектами разных классов, но обязательно из одной иерархии наследования, так, как будто это объекты одного класса.

Можно создать массив типа родительского класса и заполнять его объектами как родительского, так и его дочерних классов. Указатель типа родительского класса может указывать на объекты как собственно своего типа, так и на объекты дочерних классов, то есть указатель типа родительского класса может брать адреса объектов типа дочерних классов и вызывать их члены, например, методы, и если эти методы перегруженные, то для каждого объекта вызовется именно метод его класса, хотя сам указатель только типа родительского класса.

Итак, если у нас есть наследование, то есть у класса есть хотя бы один дочерний класс, который ему наследует, то можно использовать виртуальные методы. Для этого в родительском классе пишем метод и перед его определением указываем ключевое слово **virtual**. В дочерних классах мы должны написать метод с таким же именем и сигнатурой, слово virtual перед ними уже писать нет смысла (он виртуальный ввиду факта наследования), а вот после сигнатуры для переопределяемых виртуальных методов написать ключевое слово **override** (переопределенный) нужно, поскольку оно требует от компилятора провести проверку: действительно ли в иерархии наследования перед этим классом *есть* родительский класс с *виртуальным* методом, имеющим *точно такое же имя и сигнатуру* – если это действительно так, то все скомпилируется правильно, если нет – то будет сообщение об ошибке на этапе компиляции и мы сразу узнаем, что на самом деле не смогли правильно написать виртуальный метод или не смогли правильно переопределить его в дочернем классе или нескольких дочерних классах.

```
class Parent
{
	protected:
		int a;
	public:
		virtual void Show()
		{
			cout << a << endl;
		}
};
class Child : public Parent
{
	protected:
		double b;
	public:
		void Show() override
		{
			cout << “This is a child: ”;
			Parent::Show();		//аналогично cout << Parent::a << endl;, но зачем повторять уже
			//написанный код, когда его можно сразу вызвать по имени (метод имеет имя)
			cout << b << endl;
		}
};
```

Если нужно из тела дочернего класса вызвать метод родительского класса, который **переопределен** в дочернем классе с таким же именем и сигнатурой (входными параметрами), то пишем:

ИмяРодительскогоКласса :: ИмяМетода(входные параметры);

, причем мы вызываем готовый метод из родительского класса, поэтому передаем в него фактические параметры (имена переменных, а не описание «тип имяПараметра, тип2 имяПараметра2». Если написать просто ИмяМетода(входные параметры), то мы вызовем метод дочернего класса с таким же именем и сигнатурой, а потому для вызова «старого» переопределенного метода из родительского класса нужно явно указать имя родительского класса, поскольку в дочернем классе у него имеется метод-тезка с таким же именем и сигнатурой, и у объекта этого дочернего класса вызывается именно написанный в нем метод, как его собственный, «более близкий ему» метод. Это верно и для виртуальных и переопределенных методов.

Дочерние классы наследуют **не**private методы родительских классов, то есть при вызове у объекта дочернего класса унаследованного им метода будет вызван метод родительского класса. Но если у объекта дочернего класса нужно вызвать метод с именем и сигнатурой как у родительского класса, но со своим собственным поведением, то надо этот нестатический метод в родительском классе объявить виртуальным virtual, а в дочернем классе объявить переопределенный override метод с таким же именем и сигнатурой, но другим поведением (другим телом). К механизму виртуальных методов обращаются в тех случаях, когда в каждом производном (дочернем) классе требуется свой вариант некоторого метода, сильно напоминающий метод родительского класса, но всё же отличающийся от него в деталях (тело кода метода дочернего класса по задумке программиста отличается от наследуемого метода родительского класса хотя бы одним выражением). Классы, содержащие такие члены (методы), называются **полиморфными** и играют особую роль в ООП, реализуя третье правило ООП – полиморфизм. Классы с виртуальными методами называют **полиморфными**.

Виртуальные методы предоставляют **механизм позднего (отложенного)** или **динамического связывания**, то есть решение о том, виртуальный метод какого именно класса вызвать, принимается не при компиляции, а при выполнении программы: будет вызвана виртуальная функция именно того объекта, на который указывает указатель типа родительского класса. Несмотря на то, что сам указатель выгодно писать типа родительского класса, поскольку он может указывать на любой объект в своей иерархии наследования (объект своего типа или объект любого своего дочернего класса), но вызовется именно виртуальный метод того объекта (того дочернего или родительского класса), на который сейчас во время выполнения программы указывает данный указатель. Любой **не**статический (не имеющий слова **static** перед своим определением и **не** объявленный в статическом классе) метод базового класса может быть сделан виртуальным, для чего перед его определением используется ключевое слово virtual.

Раннее связывание – противоположность позднему связыванию. Раннее связывание возможно в случаях, когда нет виртуальных методов и иного кода, который не позволяет заранее на этапе компиляции вычислить точные типы всех данных в программе, а потому компилятор в момент компиляции (создания исполнимого .exe-файла) определяет точные типы переменных, массивов, объектов и их вызываемые методы заранее до момента фактической работы программы.

Таким образом, интерпретация каждого вызова виртуального метода через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого реально фактически выполняется вызов метода.

Выбор того, какой виртуальный метод вызвать, будет зависеть от типа объекта, на который фактически (в момент выполнения программы) направлен указатель, а не от типа указателя.

Виртуальными могут быть только нестатические члены класса (методы).

Виртуальность наследуется. После того как метод был определен как виртуальный, его повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию, причем спецификатор virtual присваивается ему компилятором неявно и писать его в коде не надо, а вот слово **override** (**переопределенный** виртуальный метод) написать в грамотном коде надо. Если от дочернего класса унаследовать «внучатый» класс, то в нем тоже будет унаследован виртуальный метод и его можно будет тоже переопределить.

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальный метод, должен иметь виртуальный деструктор.

Задание 1

Наберите приведенный ниже код и на его основе освойте использование виртуальных методов класса:

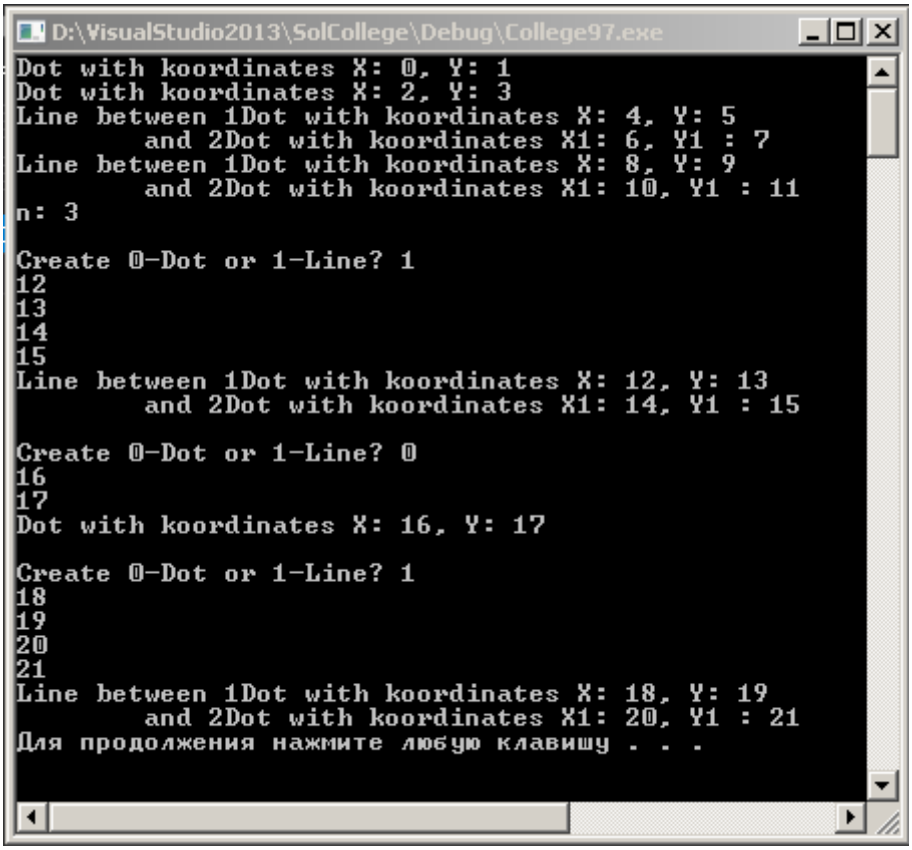
```cpp
#include <iostream>
using namespace std;

class Dot
{
protected:
    int x, y;
public:
    virtual void show()//этот метод можно вызвать у экземпляра данного класса, но еще он указан как виртуальный,
    {//т.е. он может быть переопределен в дочерних классах и пригоден для позднего связывания
        cout << "Dot with koordinates X: " << x << ", Y: " << y << endl;
    }
    Dot(int x0, int y0) :x(x0), y(y0)//параметризованный конструктор с прямой инициализацией
    {//его тело пустое, поскольку код в строке выше все 2 поля проинициализировал
    }
    Dot()//нужен конструктор без параметров, чтобы создать массив из указателей класса Точка
    {//поскольку выше мы написали конструктор с параметрами, код которого "скрыл" автоматический конструктор без параметров,
    }//то надо самим написать конструктор без параметров (тело у него пустое)
};

class Line: public Dot//наследование открытое (public), а если сделать его закрытым (private) или защищенным (protected),
{//то метод show() получит соответствующий спецификатор и станет недоступным из функции main()
protected:
    int x1, y1;//поля x и y в этом классе есть благодаря наследованию, нужно только досоздать недостающие поля
public:
    void show() override//override указывает компилятору проверить, действительно ли в родительском классе есть виртуальный метод
    {//с таким же именем и сигнатурой (списком входных параметров), и после этого переопределить (заместить) его тут новой реализацией
        cout << "Line between 1";
        Dot::show();//вызываем метод родительского класса, используя его код повторно
        cout << "\tand 2Dot with koordinates X1: " << x1 << ", Y1 : " << y1 << endl;//и дописываем недостающий код для ДАННОГО класса
    }
    Line(int x0, int y0, int x2, int y2) :Dot(x0, y0), x1(x2), y1(y2)//параметризованный конструктор, вызывающий родительский конструктор и доинициализирующий остальные поля
    {//код в строке выше делает всю работу данного конструктора, поэтому его тело пустое
    }
};

int main()
{
    Dot a(0, 1);//"статический" объект типа класса Точка с именем a принимает 2 аргумента в круглых скобках - т.е. здесь вызван конструктор с двумя параметрами
    Dot* b = new Dot(2, 3);//создается "динамический" объект типа класса Точка, для которого вызывается конструктор
    Line c(4, 5, 6, 7);//аналогично, но входных параметров 4, поскольку у этого класса 4 поля
    Line* d = new Line(8, 9, 10, 11);
    a.show();//вызываем метод у "статического" объекта
    b->show();//вызываем метод у "динамического" объекта
    c.show();
    d->show();
    int n = 3, f = 0, xA, xB, yA, yB;
    cout << "n: ";//сколько экземпляров помещать в массив?
    cin >> n;
    Dot** mas = new Dot*[n];//создаем массив указателей, а не массив точек Dot, поскольку нам надо ПОЗДНЕЕ связывание
    for (int i = 0; i < n; i++)
    {
        cout << "\nCreate 0-Dot or 1-Line? ";//создать Точку (0) или Линию (1)
        cin >> f;
        cin >> xA >> yA;//пользователь вводит значения первой точки
        if (f == 0)//if(!f)
        {
            mas[i] = new Dot(xA, yA);//создаем объект класса Точка в массиве в цикле
        }
        else
        {
            cin >> xB >> yB;//для Линии надо еще значения для второй точки
            mas[i] = new Line(xA, yA, xB, yB);//создаем объект класса Линия в массиве в цикле
        }
        mas[i]->show();//метод show() вызывается у конкретного объекта (это Точка или Линия), на который указывает указатель, хотя сам указатель типа Точка*
    }//до запуска программы пользователем неизвестно, объекты с какими индексами станут Точками, а какие - Линиями. Может, они все будут Линиями по решению пользователя
    system("pause");
    return 0;
}
```

Тестируем работу программы:



Задание 2

Написать **1** программу с классами, заданными в вашем варианте в лабораторной работе № **24**, реализовав подходящее наследование. Базовый (родительский) класс должен содержать виртуальный метод Show() и виртуальные методы по расчету периметра, площади и объема (если у экземпляра конкретного класса нет периметра, площади или объема, то соответствующие методы возвращают значение ноль (0)). В дочерних классах виртуальные методы родительских классов должны переопределяться с использованием зарезервированного слова override. В переопределенных методах Show() вызовите метод Show() родительского класса для того, чтобы использовать его работу и далее допечатать на консоль только недостающие данные. В переопределенных методах дочерних классов по расчету периметра, площади и объема постарайтесь вызывать и использовать соответствующие методы родительских классов там, где это облегчает работу и логически уместно.

Третий унаследованный класс («внучатый») проще создавать, делая из плоской фигуры родительского класса объемную (от окружности можно унаследовать шар, конус, цилиндр (добавив double поле Высоту), от квадрата – куб, прямоугольника – параллелепипед, треугольника – призму, трапеции – трапециевидную призму и т.д.).

Задание 2.1) В функции main() создать по 2 экземпляра (статический и динамический) каждого класса и вызвать у них все их методы.

Задание 2.2) Создать по массиву с элементами типа вашего класса и в цикле вызвать у них все их методы.

Задание 2.3) Создать динамический массив из указателей типа родительского класса и заполнить его экземплярами как его самого, так и его дочерних и «внучатых» классов, причем именно пользователь с клавиатуры определяет размер массива и, посредством меню, реализованного через switch-case, какие именно объекты (какого типа и с какими значениями полей), создаются в массиве. Вызвать в цикле все методы у каждого элемента такого массива.

Задание 3

Написать программу, реализующую затребованный ниже функционал. В программе предусмотреть нужные классы и их члены (поля, конструкторы с параметрами и т.д.), реализовать наследование классов и переопределяемые виртуальные методы. В функции main() продемонстрировать работоспособность объектов типа ваших классов и их функционал (методы). Создать указатель типа родительского класса и через него вызвать виртуальные методы каждого объекта класса вашей иерархии наследования.

№	Задание (по варианту)
1	Создать базовый класс «Стереометрическая фигура». Создать производные классы: «параллелепипед», «тетраэдр», «шар» со своими функциями площади поверхности. Площадь поверхности параллелепипеда: S=6*xy*. Площадь поверхности шара: $S=4r^2$. Площадь поверхности тетраэдра: $S=a^2\sqrt{3}$
2	Создать базовый класс «воин» и производные классы «пехотинец» (с винтовкой), «матрос» (с кортиком). Выведите на экран его возраст и вид оружия.

3	Создать базовый класс «медработник» и производные классы «медсестра», «хирург». Выведите на экран возраст и название должности.
4	Определить класс «Шахматная фигура» с ее координатами на шахматной доске, ее цветом (черный или белый), виртуальным методом «битья» другой фигуры, и унаследовать от него классы, соответствующие шахматным фигурам «Ферзь», «Пешка», «Конь». Написать виртуальные методы «битья» другой фигуры, которые принимают координаты другой фигуры и определяют, может ли данная (this) фигура «бить» фигуру с теми (принятыми) координатами.
5	Создать базовый класс «Вещество» и производные классы «углерод», «железо». Выведите на экран его количество и свойства (форма кристаллической решетки для углерода и чистота выработки руды для железа).
6	Создать базовый класс «сотрудник» и производные классы «стажер», «работник», «начальник отдела», «директор». Выведите на экран целое число – уровень допуска, вещественное число – зарплату и название должности.
7	Создать базовый класс «юноша» и производные классы «студент», «солдат», «военный курсант». Выведите на экран сведения о военнообязанности.
8	Создать базовый класс «фигура» с методом вычисления ее объема. Создать производные классы: «параллелепипед», «пирамида», «тетраэдр», «шар» со своими функциями объема. Объем параллелепипеда: $V=xyz$ (x, y, z – длины трех граней параллелепипеда, пирамиды: $V=xyh$ (x,y – длины двух сторон пирамиды, h – ее высота), тетраэдра: $V=\frac{a}{12}\sqrt{2}$, шара: $V=4r^3/3$.
9	Создать базовый класс «Грузоперевозчик» и производные классы «Самолет», «Поезд», «Автомобиль». Определить время и стоимость перевозки для указанных городов и расстояний.
10	Создать базовый класс «Домашнее животное» и производные классы «Собака», «Кошка», «Хомяк» и др. С помощью конструктора установить имя каждого животного и его характеристики. Распечатать характеристики каждого животного посредством вызова виртуального метода.
11	Создать базовый класс «Учащийся» и производные классы «Школьник» и «Студент». Создать массив объектов базового класса и заполнить этот массив объектами всех классов. Показать (распечатать) отдельно студентов и отдельно школьников.
12	Создать базовый класс «Музыкальный инструмент» и производные классы «Ударный», «Струнный», «Духовой». Создать массив объектов «Оркестр». Выдать состав оркестра, переопределив метод.
13	Создать базовый класс «Работник фирмы» и производные классы «Менеджер», «Администратор», «Программист». Вывести название должности, стаж работы.
14	Создать базовый класс «Прогрессия». Создать производные классы: «арифметическая прогрессия» и «геометрическая прогрессия». Каждый класс имеет два поля типа double. Первое – первый член прогрессии, второе (double) – постоянная разность (для арифметической) и постоянное отношение (для геометрической). Определить функцию вычисления суммы, где параметром является количество элементов прогрессии. Арифметическая прогрессия $a_j=a_0+jd$, $j=0,1,2,\ldots$ Сумма арифметической прогрессии: $S_n=(n+1)(a_0+a_n)/2$ Геометрическая прогрессия: $a_j=a_0r^j$, $j=0,1,2,\ldots$ Сумма геометрической прогрессии: $S_n=(a_0-a_nr)/(1-r)$
15	Создать базовый класс «фигура» с методом расчета площади. Создать производные классы: «прямоугольник», «круг», «прямоугольный треугольник», «трапеция» со своими методами расчета площади. Площадь трапеции: S = (a+b)*h/2
16	Создать базовый класс «железнодорожный вагон» и производные классы «вагон для перевозки автомобилей», «цистерна». Выведите на экран его вес и количество единиц товара в вагоне.
17	Создать базовый класс «норма». Создать производные классы: «вектор» из 10 элементов, «матрица» (2x2). Определить функцию (метод) расчета нормы для вектора – корень квадратный из суммы элементов по модулю, для матрицы – максимальное значение по модулю.
18	Создать базовый класс «Транспортное средство» и производные классы «Автомобиль», «Велосипед», «Повозка». Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.
19	Создать базовый класс «фигура», и производные класс – «круг», «прямоугольник», «трапеция». Определить их периметры.
20	Создать базовый класс «работник» и производные классы – «служащий с почасовой оплатой», «служащий в штате

	с премией» и «служащий с процентной ставкой». Определить функцию начисления зарплаты.
21	Создать базовый класс – «Родитель», у которого есть имя. Определить виртуальную функцию печати данных об объекте. Создать производный класс «Ребенок», у которого метод печати дополнительно выводит отчество. Создать производный класс от последнего класса – «Внук», у которого есть фамилия. Написать для них виртуальные перегруженные методы печати данных ФИО (какие есть) на консоль.
22	Напишите базовый класс «Сотрудник» и производные от него «Директор», «Замдиректора», «НачальникЦеха», «Мастер», «КонвейерныйРабочий». Определите для каждого сотрудника виртуальный метод Работать(), который печатает на консоль краткое описание обязанностей сотрудника и виртуальный метод ПолучитьЗарплату(), рассчитывающий заработную плату сотрудника.
23	Создать класс «Живое». Определить наследуемые классы – «лиса», «кролик» и «растение». Лиса ест кролика. Кролик ест растения. Растение поглощает солнечный свет. Представитель каждого класса может умереть, если достигнет определенного возраста или для него не будет еды. Напишите виртуальные методы поедания и определения состояния живого существа (живой или нет, в зависимости от достижения предельного возраста и наличия еды (входной параметр)).
24	Создать базовый класс «Уравнение». Создать производные классы: «Линейные уравнения» и «Квадратные уравнения». Определить метод вычисления корней уравнений.
25	Создать базовый класс «точка» и производные классы «квадрат», «пирамида». Выведите на экран площадь фигур и их координаты.
26	Создать базовый класс «кривая» с методом вычисления координаты по оси *y* для некоторой точки с известной координатой по оси *x*. Создать производные классы: «прямая», «эллипс», «гипербола» со своими методами вычисления y в зависимости от входного параметра *x*. Уравнение прямой: $y=ax+b$, эллипса: $x^2/a^2+y^2/b^2=1$, гиперболы: $x^2/a^2 - y^2/b^2=1$
27	Создать базовый класс «Пассажироперевозчик» и производные классы «Самолет», «Поезд», «Автомобиль». Определить время и стоимость передвижения.
28	Напишите базовый класс «ТранспортноеСредство» и производные от него классы «Автомобиль», «Самолет», «Катер», «ПодводнаяЛодка» с виртуальными методами, рассчитывающими объем потребленного топлива и пройденное расстояние.