

Лабораторная работа 8. Программирование динамических и виртуальных методов

1. Цель работы

Изучить возможности виртуальных функций при наследования классов на языке C++, абстрактные классы.

2. Краткие теоретические сведения

ВИРТУАЛЬНЫЕ ФУНКЦИИ

Виртуальные функции являются важной частью реализации механизма полиморфизма (то есть выполнения разных действий с объектами разного типа). Использование виртуальных функций позволяет выполнить нужные действия в том случае, если доступ к объекту осуществляется не по значению, а по указателю.

Пусть имеется класс «Фигура», и имеются его классы-наследники «Круг», «Квадрат» и «Треугольник». Пусть в каждом из этих классов есть функция `draw()`, которая прорисовывает фигуры на экране.

Теперь пусть имеется указатель `X` на объект класса «Фигура». По правилам языка этот указатель может хранить адрес как самого класса «Фигура», так и адрес любого из его потомков, т.е. и «Круга» и «Квадрата» и «Треугольника». Теперь необходимо реализовать функцию `draw()` так, чтобы при обращении к ней через указатель `X->draw()` вызывалась бы именно нужная функция `draw()`, то есть именно того класса, адрес которого и хранится в указателе `X`. Понятно, что у круга, квадрата и треугольника это будут разные функции. Чтобы обеспечить такую возможность для функции `draw()` её необходимо объявить виртуальной в базовом классе.

Ниже приводятся примеры, демонстрирующие возможности виртуальной функции.

Первый пример показывает, что бывает, когда базовый и производный классы содержат функции с одним и тем же именем, и к ним обращаются с помощью указателей, но без использования виртуальных функций.

```
class Base                                //Базовый класс
{
public:
    void show()                          //Обычная функция
    { cout << "Base\n"; }
};

class Derv1 : public Base //Производный класс 1
{
public:
    void show()
    { cout << "Derv1\n"; }
};

class Derv2 : public Base //Производный класс 2
{
public:
    void show()
    { cout << "Derv2\n"; }
};

int main()
{
```

```

Derv1 dv1;           //Объект производного класса 1
Derv2 dv2;           //Объект производного класса 2
Base* ptr;            //Указатель на базовый класс

ptr = &dv1;           //Адрес dv1 занести в указатель
ptr->show();           //Выполнить show()
ptr = &dv2;           //Адрес dv2 занести в указатель
ptr->show();           //Выполнить show()
return 0;
}

```

Поскольку указатели на объекты дочерних классов совместимы по типу с указателями на объекты базового класса присвоение `ptr = &dv1;` и `ptr = &dv2;` вполне корректно. В результате выполнения приведенной программы будет дважды вызвана функция `show()` именно базового класса, так как компилятор в данном случае не смотрит на содержимое указателя `ptr`, а выбирает тот метод, который удовлетворяет типу указателя (см. рисунок).

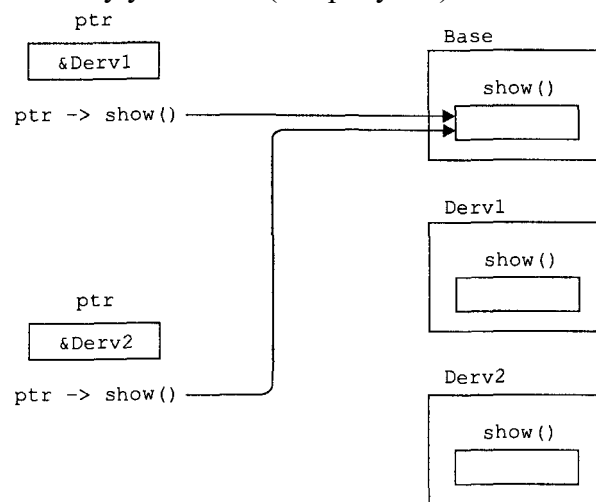


Рисунок 1. Доступ через указатель без использования виртуальных функций

Теперь изменим обсуждаемую программу таким образом, чтобы функция `show()` базового класса была бы объявлена как виртуальная функция (то есть добавим перед её объявлением в базовом классе ключевое слово `virtual`).

```

class Base           //Базовый класс
{
public:
    virtual void show()           //Виртуальная функция
    { cout << "Base\n"; }
};

class Derv1 : public Base //Производный класс 1
{
public:
    void show()
    { cout << "Derv1\n"; }
};

class Derv2 : public Base //Производный класс 2
{
public:

```

```

        void show()
        {   cout << "Derv2\n"; }
    };

int main()
{
    Derv1 dv1;           //Объект производного класса 1
    Derv2 dv2;           //Объект производного класса 2
    Base* ptr;           //Указатель на базовый класс

    ptr = &dv1;          //Адрес dv1 занести в указатель
    ptr->show();          //Выполнить show()
    ptr = &dv2;          //Адрес dv2 занести в указатель
    ptr->show();          //Выполнить show()
    return 0;
}

```

На выходе, в отличие от первого примера, будем иметь

Derv1

Derv2

то есть при вызове функции *show()* будут выполнены методы именно соответствующих производных классов, а не базового. То есть один и тот же вызов *ptr->show()*; запускает на выполнение разные функции в зависимости от содержимого указателя *ptr*. Компилятор выбирает функцию, удовлетворяющую тому, что занесено в указатель, а не типу указателя, как в первом примере (см. рисунок).

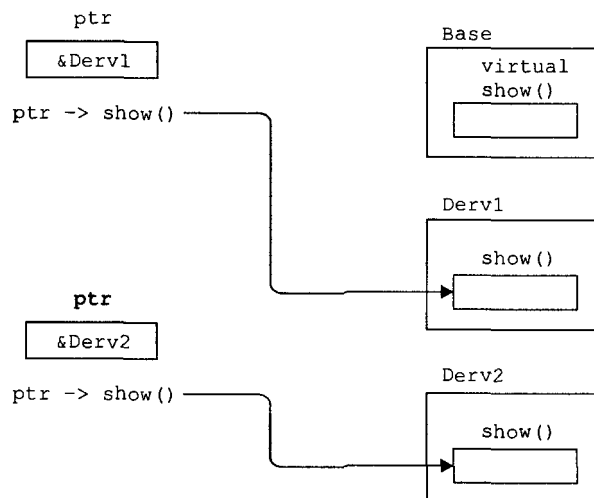


Рисунок 2. Доступ через указатель к виртуальным функциям.

АБСТРАКТНЫЕ КЛАССЫ И ЧИСТЫЕ ВИРТУАЛЬНЫЕ ФУНКЦИИ

Базовый класс, объекты которого никогда не будут созданы, называется абстрактным классом. Такой класс существует с единственной целью – быть родительским по отношению к производным классам, объекты которых уже будут реализованы. Если класс должен быть абстрактным, то от создания объектов этого класса его можно защитить программно. Для этого достаточно ввести в класс хотя бы одну чистую виртуальную функцию. Чистая виртуальная функция – это функция, после объявления которой добавлено выражение `= 0`. В следующем примере демонстрируется объявление и работа с чистыми виртуальными функциями.

```

class Base                               //базовый класс
{
    public:
    virtual void show() = 0; //чистая виртуальная функция
}

```

```

};

class Derv1 : public Base    //порожденный класс 1
{
public:
    void show()
    { cout << "Derv1\n"; }
};

class Derv2 : public Base    //порожденный класс 2
{
public:
    void show()
    { cout << "Derv2\n"; }
};

int main()
{
    // Base bad;                //невозможно создать объект
                                //из абстрактного класса

    Base* arr[2];               //массив указателей на
                                //базовый класс

    Derv1 dv1;                  //Объект производного класса 1
    Derv2 dv2;                  //Объект производного класса 2

    arr[0] = &dv1;              //Занести адрес dv1 в массив
    arr[1] = &dv2;              //Занести адрес dv2 в массив

    arr[0]->show();             //Выполнить функцию show()
    arr[1]->show();             //над обоими объектами
    return 0;
}

```

Знак равенства при объявлении чистой виртуальной функции *virtual void show() = 0*; это не операция присваивания, это просто способ сообщить компилятору, что функция будет чистой виртуальной. Если теперь попытаться создать объект этого класса, компилятор выдаст ошибку.

На практике механизм виртуальных функций часто применяется для корректного вызова деструкторов. Технология программирования прямо предписывает, что деструкторы базовых классов должны быть виртуальными. Если деструктор базового класса не является виртуальным, то, будучи обычным методом, он будет вызываться независимо от того, данные какого типа в нем хранятся. Это приведет к тому, что может быть удалена только та часть объекта, которая относится к базовому классу. Конечно, если ни один из деструкторов (ни у базового, ни у производного класса) ничего особенного не делает, тогда их виртуальность перестает быть такой уж необходимой.

3. Задание (5 баллов)

1. Определить иерархию классов (в соответствии с вариантом).
2. Определить в классе статическую компоненту - указатель на начало связанного списка объектов и статическую функцию для просмотра списка.
3. Реализовать классы.
4. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в список, после чего список просматривается.
5. Сделать соответствующие методы не виртуальными и посмотреть, что будет.

6. Реализовать вариант, когда объект добавляется в список при создании, т.е. в конструкторе.

Методические указания.

1. Для определения иерархии классов связать отношением наследования классы заданного варианта. Из перечисленных классов выбрать один, который будет стоять во главе иерархии. Это абстрактный класс.

2. Определить в классах все необходимые конструкторы и деструктор.

3. Поля класса специфицировать как **protected**.

4. Пример определения статических компонентов:

```
static person* begin; // указатель на начало списка
static void print(); // просмотр списка
```

5. Статическое поле инициализировать вне определения класса, в глобальной области.

6. Для добавления объекта в список предусмотреть метод класса, т.е. объект сам добавляет себя в список. Например, `a.Add()` – объект **a** добавляет себя в список.

Включение объекта в список можно выполнять при создании объекта, т.е. поместить операторы включения в конструктор. В случае иерархии классов, включение объекта в список должен выполнять **только** конструктор базового класса. Вы должны продемонстрировать оба этих способа.

7. Список просматривать путем вызова виртуального метода **Show** каждого объекта.

8. Статический метод просмотра списка вызывать не через объект, а через класс.

4. Варианты заданий

- 1) студент, преподаватель, персона, завкафедрой;
- 2) служащий, персона, рабочий, инженер;
- 3) рабочий, кадры, инженер, администрация;
- 4) деталь, механизм, изделие, узел;
- 5) организация, страховая компания, судостроительная компания, завод;
- 6) журнал, книга, печатное издание, учебник;
- 7) тест, экзамен, выпускной экзамен, испытание;
- 8) место, область, город, мегаполис;
- 9) игрушка, продукт, товар, молочный продукт;
- 10) квитанция, накладная, документ, чек;
- 11) автомобиль, поезд, транспортное средство, экспресс;
- 12) двигатель, двигатель внутреннего сгорания, дизель, турбореактивный двигатель;
- 13) республика, монархия, королевство, государство;
- 14) млекопитающие, парнокопытные, птицы, животное;
- 15) корабль, пароход, парусник, корвет.

5. Контрольные вопросы

1. Какие возможности перед программистом открывают виртуальные функции?
2. Истинно ли утверждение о том, что указатель на базовый класс может ссылаться на объекты порожденного класса?
3. Пусть указатель `p` ссылается на объекты базового класса и содержит адрес объекта порожденного класса. Пусть в обоих этих классах имеется неvirtуальный метод `ding()`. Тогда выражение `p->ding()` вызовет метод `ding()` из класса.
4. Напишите описание для виртуальной функции `dang()`, возвращающей результат `void` и имеющей аргумент типа `int`.
5. Пусть указатель `p` ссылается на объекты базового класса и содержит адрес объекта порожденного класса. Пусть в обоих этих классах имеется виртуальный метод `ding()`. Тогда выражение `p->ding()` вызовет метод `ding()` из класса.

6. Напишите описание для чистой виртуальной функции `aragorn()`, не возвращающей значений и не имеющей аргументов.
7. Чистая виртуальная функция, это виртуальная функция, которая:
 - а) делает свой класс абстрактным;
 - б) не возвращает результата;
 - в) используется в базовом классе;
 - г) не имеет аргументов.
8. Напишите определение массива `ragg`, содержащего 10 указателей на объекты класса `dong`.
9. Абстрактный класс используется тогда, когда
 - а) не планируется создавать порожденные классы;
 - б) есть несколько связей между двумя порожденными классами
 - в) необходимо запретить создавать объекты класса