АКАДЕМИЯ ГРАЖДАНСКОЙ ЗАЩИТЫ МЧС РОССИИ

Кафедра № 36
Информатики и вычислительной техники

Дисциплина
ПРОГРАММИРОВАНИЕ

Тема занятия:

Виртуальные функции – члены класса. Полиморфизм

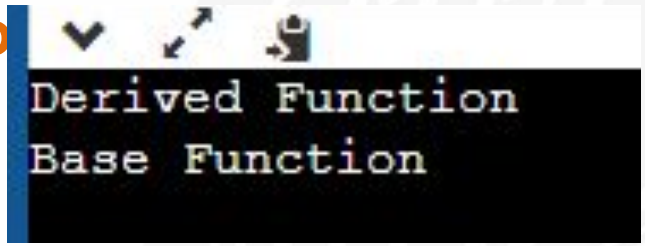
Учебные вопросы:

1. Доступ к виртуальным функциям через указатель базового класса.
2. Полиморфизм.

Работы с функциями через указатель
Пример 3:

```
#include <iostream>
using namespace std;
class Base {
        public:
             void print() { cout << "Base Function" << endl; }
};
 class Derived : public Base {
       public:
              void print() { cout << "Derived Function" << endl; }
 };
int main() {
       Derived derived1, *p1;
       Base d1,*p2;
       p1=&derived1;
       p1->print();
       p2=&d1;
       p2->print();
return 0; }
```

Результат работы про



Вызов функций-членов базового и производного классов с помощью указателя на базовый класс.

Вызов функции задается выражением:

указатель_на_объект_класса -> обращение_к_функции-члену

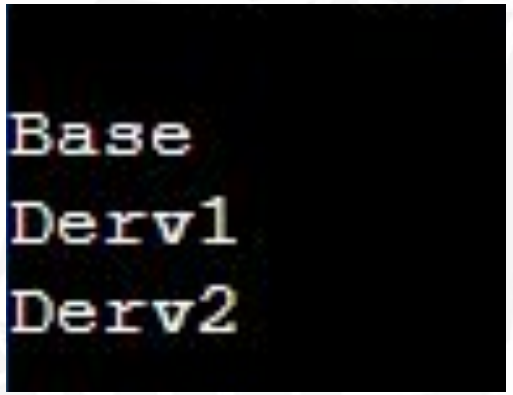
Рассмотрим программу, в которой вызовы функций-членов выполняются через указатели на объекты.

Кроме этого, особенностью программы является наследование классов и переопределение функции базового класса в двух производных от него классах.

Пример 1. Доступ к функциям через указатели

```
class Base            //Базовый класс
   {
public:
      void show()   { cout <<"\n Base"; }
  };
        class Derv1:public Base   // Производный класс 1
           {
        public:
          void show()  {cout<<"\n  Derv1";  }
          };
        class Derv2:public Base   // Производный класс 2
           {
        public:
          void show() {cout<< "\n Derv2";  }
          };
int main()
  {
      Base b, *p;
      Derv1 dv1, *p1;
      Derv2 dv2, *p2;
      p=&b;
      p->show();
      p1=&dv1;
      p1->show();
      p2=&dv2;
      p2->show();
return 0;
}
```

Результат работы программы:



Сформулируем условие к тексту программы представленной на предыдущем слайде:

В main() объявлены объекты базового и производных классов, а также указатель на объекты базового класса и указатели на объекты каждого из производных классов. Функция show () определена в базовом классе Base и переопределяется в каждом из двух производных классов Derv1 и Derv2. Вызов функции show() во всех трех случаях выполняется с использованием указателей. В первом случае show() вызывается для объекта базового класса через указатель, указывающий на данный объект. В двух других случаях вызовы функции show() выполняются для объектов производных классов через соответствующие указатели на эти классы.

Теперь впервые обратим внимание на важную связь производного и базового классов (имеющую место не только в контексте рассматриваемой программы, а вообще при наследовании классов).

Уточняем:

- **указатель базового класса может указывать на объект производного класса.**

Например, совершенно правильным является оператор

```
p = &dv1;
```

присваивающий указателю на базовый класс Base адрес объекта dv1 производного класса Derv1.

- **Указатель производного класса нельзя использовать для доступа к объектам базового класса.**

Используем указанное свойство указателя базового класса в следующем варианте рассмотренной выше программы:

Пример 2. Доступ к функциям через указатель базового класса

```
class Base  //Базовый класс
 {
 public:
    void show() { cout<<"\n Base"; }
};
   class Derv1:public Base       // Производный класс 1
      {
      public:
           void show()  {cout<<"\n Derv1";  }
      };
   class Derv2:public Base       // Производный класс 2
       {
        public:
     void show()  {cout<< "\n Derv2";  }
      };
int main()
   {
   Base b, *p;
   Derv1 dv1;
   Derv2 dv2;
   p=&b;
   p->show();
   p=&dv1;
   p->show();
   p=&dv2;
   p->show();
return 0;    }
```

Результат работы программы:

Резутьтат отличается от результата предыдущей программы.

Во всех трех случаях вызова функции show () вызывается одна и та же функция - функция show () базового класса.

Именно такой вызов предусмотрен синтаксисом языка C++, т.е. выбор функции (*не виртуальной*) зависит только от типа указателя, но не от его значения.

"Настроив" указатель базового класса на объект производного класса, не удастся с помощью этого указателя вызвать функцию из производного класса.

Полиморфизм

Виртуальная функция в языке C++ — это особый тип функции, которая, при её вызове, выполняет «наиболее» дочерний метод, который существует между родительским и дочерними классами. Это свойство еще известно, как **полиморфизм**.

Полиморфизм – свойство, которое позволяет использовать одно и тоже имя функции для решения двух и более схожих, но технически разных задач.

Полиморфизм – возможность замещения методов объекта родителя методами объекта-потомка, имеющих то же имя.

Полиморфизм по-гречески означает «много форм». Объекты, имеющие общего предка, *могут принимать разные формы, оставаясь при этом схожими*.

Чтобы использовать полиморфизм, необходимо чтобы:

1) все классы-потомки являлись наследниками одного и того же базового класса;

2) функция, реализующая метод, должна быть объявлена виртуальной в базовом классе.

Виртуальным называется метод, ссылка на который вычисляется на этапе выполнения программы.

Рассмотрим **пример 3**, когда базовый и производные классы содержат функции с одни и тем же именем, и к ним обращаются с помощью указателей, но без использования виртуальных функці

```
class Base
{
public:
void show() { cout << "Base\n";      }
};

class A: public Base
{
public:
        void show() { cout << "Class A\n"; }
};

class B: public Base
{
public:
        void show() { cout << "Class B\n"; }
};

void  main()
{
        A a;                    //объект производного класса A
        B b;                    //объект производного класса B
        Base *ptr;              // указатель на базовый класс

        ptr=&a;                         //адрес занести в указатель
        ptr->show();                    // выполнить show()

        ptr=&b;                         //адрес занести в указатель
        ptr->show();                    // выполнить show()
        }
```

A, B, Base – это типы.

Указатели на объекты производных классов совместимы по типу с указателями на объекты базового класса.

Base *ptr; ptr=&a; ptr=&b;

НО указатели производных классов между собой не совместимы!

Пример:

A *ptr; ptr=&a;

ptr=&b; // указатель класса A не совместим с указателем класса B!!!

Теперь необходимо понять, какая собственно функция выполняется в этой строчке Ptr->Show();

Это функция Base::show() или A::show() или B:show()?

Результат выполнения дает простой ответ:

Base

Base

Всегда выполняется метод базового класса. Компилятор не смотрит на содержимое указателя, а выбирает метод, определяемый

Доступ к виртуальным методам через указатели

Сделаем одну корректировку в нашей программе: поставим ключевое слово **virtual** перед объявлением функции **show()** в базовом классе.

На выходе имеем:

Class A

Class B

Теперь выполняются методы производных классов. Один и тот же вызов ставит на выполнение разные функции в зависимости от содержимого указателя **ptr**.

Если метод в базовом классе объявлен как виртуальный, то компилятор выбирает метод **по содержимому указателя**, а не по типу указателя, как было в первом примере.

Абстрактные классы и чисто виртуальные методы

Базовый класс, объекты которого никогда не будут реализованы называется ***абстрактным*** классом. Такой класс может существовать с единственной целью – быть родительским классом к производным классом, объекты которых будут реализованы.

Для того чтобы сделать базовый класс абстрактным, достаточно ввести в класс хотя бы одну чисто виртуальную функцию.

Чисто *виртуальная функция* – это функция, после объявления которой добавлено выражение **=0**.

Пример 4. Определим абстрактный класс, который представляет геометрическую фигуру

Стоит отметить, что **абстрактный класс может определять и обычные функции и переменные**, **может иметь несколько конструкторов**, **но** при этом **нельзя создавать объекты этого абстрактного класса**.

Пример 4. Определим абстрактный класс, который представляет геометрическую фигуру

```
class Figure
{
public:
    virtual double getSquare() =0;
    virtual double getPerimeter() =0;
    virtual void showFigureType()=0;
};
```

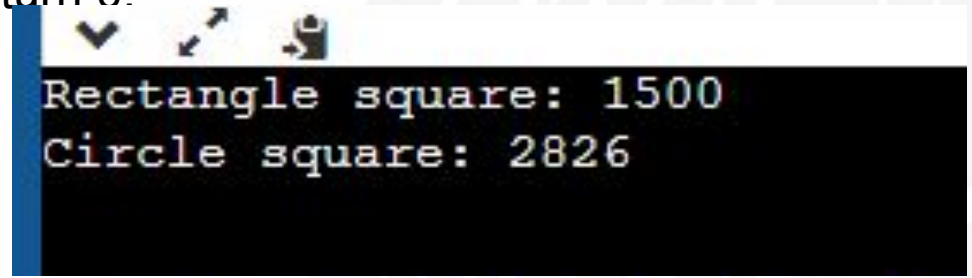
Класс Figure является абтрактным, потому он содержит как минимум одну чистую виртуальную функцию. А в данном случае даже три таких функции. И ни одна из функций не имеет никакой реализации. Реализацию должны определять классы-наследники.

При этом мы не можем создать объект абстрактного класса.

```
class Figure
{
public:
    virtual double getSquare() =0;
    virtual double getPerimeter() =0;
    virtual void showFigureType()=0;
};

class Rectangle : public Figure
{
private:
    double width;
    double height;
public:
    Rectangle(double w, double h) : width(w),
height(h)
    {     }
    double getSquare() override
    {   return width * height;     }
    double getPerimeter() override
    {  return width * 2 + height * 2;   }
    void showFigureType()
    {   cout << "Rectangle" << std::endl;    }
};
```

```
class Circle : public Figure
{
private:
    double radius;
public:
    Circle(double r) : radius(r) {  }
       double getSquare() override
    {   return radius * radius * 3.14;   }
    double getPerimeter() override
    {  return 2 * 3.14 * radius;  }
    void showFigureType()
    {   cout << "Circle" << std::endl; }
};
int main()
{
    Rectangle rect(30, 50);
    Circle circle(30);
    cout << "Rectangle square: " << rect.getSquare()
<<endl;
    cout << "Circle square: " << circle.getSquare()
<<endl;
    return 0;
}
```



Подведем итоги:

1. Полиморфизм позволяет использовать одно и то же имя функции для решения двух и более схожих, но технически разных задач.
2. Если метод в базовом классе объявлен как виртуальный, то компилятор выбирает метод *по содержимому указателя*, а не по типу указателя.
3. Чтобы сделать базовый класс абстрактным, достаточно ввести в класс хотя бы одну *чисто виртуальную функцию*.
4. Если в базовом классе объявлена чисто виртуальная функция, в производных классах объявление чисто виртуальных функций запрещено.