Севастопольский государственный университет
Кафедра информационных систем

Курс лекций по дисциплине

# «ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ"

(ООП)

Лектора: Пелипас Всеволод Олегович
Сметанина Татьяна Ивановна

# Лекция 5

# ВИРТУАЛЬНЫЕ ФУНКЦИИ. АБСТРАКТНЫЕ КЛАССЫ. ПОЛИМОРФИЗМ.

# Ранее и позднее связывание

В C++ полиморфизм поддерживается двумя способами.

Во-первых, при компиляции он поддерживается посредством перегрузки функций и операторов. Такой вид полиморфизма называется ***статическим полиморфизмом***, поскольку он реализуется еще до выполнения программы, путем ***раннего связывания*** идентификаторов функций с физическими адресами на стадии компиляции и компоновки.

Во-вторых, во время выполнения программы он поддерживается посредством виртуальных функций. Встретив в коде программы вызов виртуальной функции, компилятор (а, точнее, компоновщик) только обозначает этот вызов, оставляя связывание идентификатора функции с ее адресом до стадии выполнения. Такой процесс называется ***поздним связыванием***.

# Виртуальные функции

***Виртуальная функция*** – это функция, вызов которой (и выполняемые при этом действия) зависят от типа объекта, для которого она вызвана. Объект определяет, какую функцию нужно вызвать уже во время выполнения программы. Этот вид полиморфизма называется ***динамическим полиморфизмом***.

Работа с объектами чаще всего производится через указатели. Указателю на базовый класс можно присвоить значение адреса объекта ***любого*** производного класса. Такой указатель связывается со своим объектом только во время выполнения программы, то есть динамически. Класс, содержащий хоть одну виртуальную функцию называется ***полиморфным***.

Для каждого полиморфного типа данных компилятор создает **таблицу виртуальных функций** и встраивает в каждый объект такого класса скрытый указатель на эту таблицу.

Таблица содержит адреса виртуальных функций соответствующего типа. Имя указателя на таблицу виртуальных функций и название таблицы зависят от реализации в конкретном компиляторе.

# Виртуальные функции

Компилятор автоматически встраивает в начало конструктора полиморфного класса фрагмент кода, который инициализирует указатель на таблицу виртуальных функций.

Поскольку объекты с виртуальными функциями должны поддерживать таблицу виртуальных методов, то их использование всегда ведет к некоторому повышению затрат памяти и снижению быстродействия программы.

Функции, у которых известен интерфейс вызова (то есть прототип), но реализация не может быть задана в общем случае, а может быть определена только для конкретных случаев, называются ***виртуальными***.

***Виртуальные*** функции – это функции, которые гарантируют, что будет вызвана правильная функция для объекта безотносительно к тому, какое выражение используется для осуществления вызова.

# Правила описания и использования виртуальных функций

- Если в базовом классе метод определен как виртуальный, метод, определенный в производном классе с тем же именем и набором параметров, автоматически становится виртуальным, а с отличающимся набором параметров — обычным.
- Виртуальные методы наследуются, то есть переопределять их в производном классе требуется только при необходимости задать отличающиеся действия. Права доступа при переопределении изменить нельзя.
- Если виртуальный метод переопределен в производном классе, объекты этого класса могут получить доступ к методу базового класса с помощью операции доступа к области видимости.
- Виртуальный метод не может объявляться с модификатором **static**, но может быть объявлен как дружественный.
- Если в классе вводится описание виртуального метода, он должен быть определен хотя бы как чисто виртуальный.

# Правила описания и использования виртуальных функций

Чисто виртуальный метод содержит признак =0 вместо тела, например:

**virtual void f(int) = 0;**

Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный).

**Виртуальным** называется метод, ссылка на который разрешается (уточняется) на этапе выполнения программы (перевод красивого английского слова **virtual** — в данном значении всего-навсего «фактический», то есть ссылка разрешается по факту вызова).

# Виртуальные деструкторы

Конструкторы не могут быть виртуальными, а деструкторы могут и часто ими являются.

```
class Base{                          // базовый класс
 public: Base(){}
   ~Base(){cout<<"destr_base";}
};
class Derived: public Base{          // производный класс
 public:
  ~Derived(){cout<<"destr_derived";}
};
 main(){
  Base *ob= new Derived;
   delete ob;
 }
```


При удалении объекта производного класса вызывается только деструктор базового класса, **деструктор производного класса не вызывается** (может происходить утечка памяти). Исправим:

# Виртуальные деструкторы

```
class Base{
 public: Base(){}
  virtual ~Base(){cout<<"destr_base";} // виртуальный деструктор
};
class Derived: public Base{
 public:
  ~Derived(){cout<<"destr_deriver"<<endl;}
};
main(){
 Base *ob= new Derived;
 delete ob; }
```

```
destr_deriver
destr_base
```

При удалении объекта производного класса, если деструктор объявлен как виртуальный, то будет вызван деструктор соответствующего производного класса. Затем деструктор производного класса вызовет деструктор базового класса и объект будет правильно удален.

# Виртуальные деструкторы

Рекомендации:

- используйте виртуальный деструктор, если базовый класс содержит виртуальный функции;
- используйте виртуальные функции только в том случае, если программа содержит и базовый, и производный классы;
- не пытайтесь создать виртуальный конструктор;
- методы, которые во всей иерархии останутся неизменными или те, которыми производные классы пользоваться не будут, делать виртуальными нет смысла.

Объект, определенный через указатель или ссылку и содержащий виртуальные методы, называется ***полиморфным***.

***Полиморфизм*** состоит в том, что с помощью одного и того же обращения к методу выполняются различные действия в зависимости от типа объекта, на который ссылается указатель в каждый момент времени.

# Абстрактные классы

Класс, содержащий хотя бы один чисто виртуальный метод, называется ***абстрактным***.

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах.

Абстрактный класс может использоваться только в качестве базового для других классов — объекты абстрактного класса создавать нельзя, поскольку прямой или косвенный вызов чисто виртуального метода приводит к ошибке при выполнении.

При определении абстрактного класса необходимо иметь в виду следующее:

- абстрактный класс нельзя использовать при явном приведении типов, для описания типа параметра и типа возвращаемого функцией значения;
- допускается объявлять указатели и ссылки на абстрактный класс, если при инициализации не требуется создавать временный объект;

# Пример использования виртуальных функций

- если класс, производный от абстрактного, не определяет все чисто виртуальные функции, он также является абстрактным.

Таким образом, можно создать функцию, параметром которой является указатель на абстрактный класс. На место этого параметра при выполнении программы может передаваться указатель на объект любого производного класса. Это позволяет создавать полиморфные функции работающие с объектом любого типа в пределах одной иерархии.

***Пример:***

```
class people{                    // абстрактный базовый класс people
   public:
   people() {cout<<"const_people"<<endl;};
   virtual ~people() {cout<<"destr_people"<<endl;};
   virtual void show()=0;    // чисто виртуальная функция
};
```

# Пример использования виртуальных функций

```
class men: public people                // производный класс мужчина
{  char name[15];
   int age;
public:
   men(char _name[15], int _age)  // конструктор
   {   strcpy(name, _name);
       age=_age;
       cout<<"constr_men"<<endl;
   }

   ~men() {cout<<"destr_men"<<endl;};  //деструктор

void show()              // переопределяемая функция
   {    cout<<"\n name "<<name <<"\n age ="<< age<<endl;
   }
};
```

# Пример использования виртуальных функций

```
class children: public men          // производный класс ребенок
{  char name[15];
   int  year;
public:                             // конструктор
   children(char _name[15], int _year): men( _name, _year)
  {   strcpy(name,_name);
      year=_year;
      cout<<"constr_children"<<endl;
   }
   ~children() {cout<<"destr_children"<<endl;};    //деструктор

   void show()          // переопределяемая функция
   {    cout<<"\n name "<<name <<"\n year ="<< year <<endl;
   }
};
```

# Пример использования виртуальных функций

```
int main(){
    char name[15];   int age, year;
    cout<<"Введите для мужчины имя, возраст";
    cin>>name>>age;
// создание указателя m на объект
    people *m=new men(name, age);
// вызов переопределенной функции
    m->show();
    delete m;

cout<<"\n Введите для ребенка имя, год";
    cin>> name >> year;
// создание указателя ch на объект
    people *ch= new children(name, year);
// вызов переопределенной функции
ch->show();
   delete ch;
}
```

```
Введите для мужчины имя, возраст
POMA 25
const_people
constr_men

 name POMA
 age =25
destr_men
destr_people

 Введите для ребенка имя, год
BOBA 1995
const_people
constr_men
constr_children

 name BOBA
 year =1995
destr_children
destr_men
destr_people
```