НАСЛЕДОВАНИЕ

Виртуальные функции. Абстрактные классы.

Дружественные классы

Простой пример использования виртуальной функции

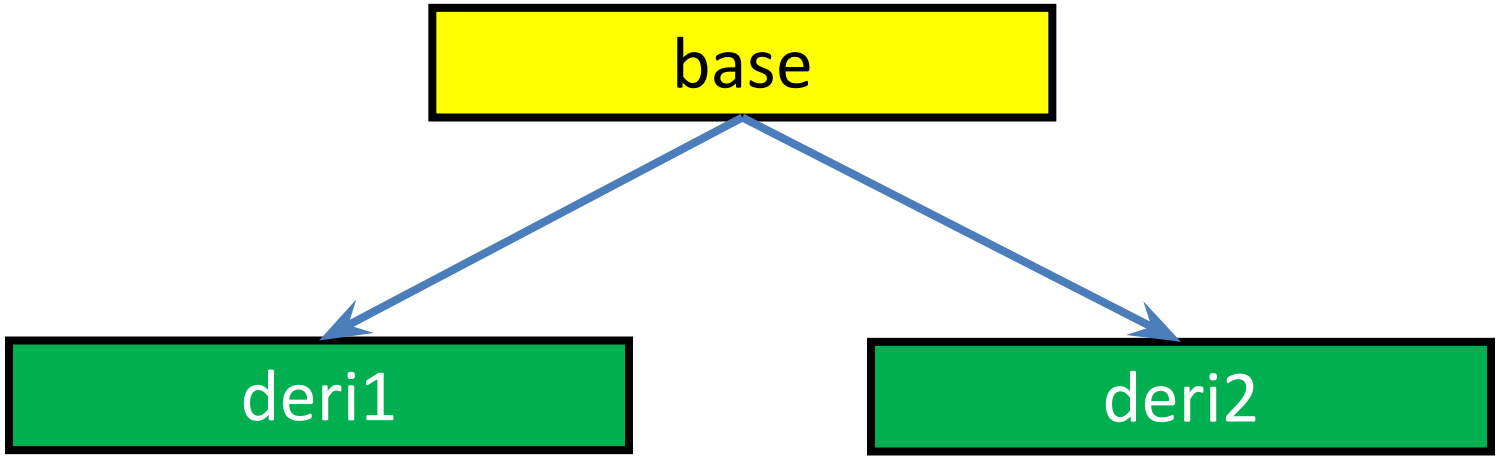
```
#include <iostream>
#include <conio>
using namespace std;
class base
{
public:
int i;
base(int x) { i = x; }
virtual  void func()
{
cout << "func() of base class: ";
cout << i << endl;
}
};
class deri1 : public base
{
public:
deri1(int x) : base(x) { }
void func()
```

```
int main()
{
base *p;
base obj_base(10);
deri1 obj_deri1 (10);
deri2 obj_deri2(10);
p = &obj_base;
p->func();
// функция func() класса base
p = &obj_deri1;
p->func();

// функция func() про
p = &obj_deri2;
```

```
func() of base class: 10
func() derived1 class: 100
func() derived2 class: 20
```

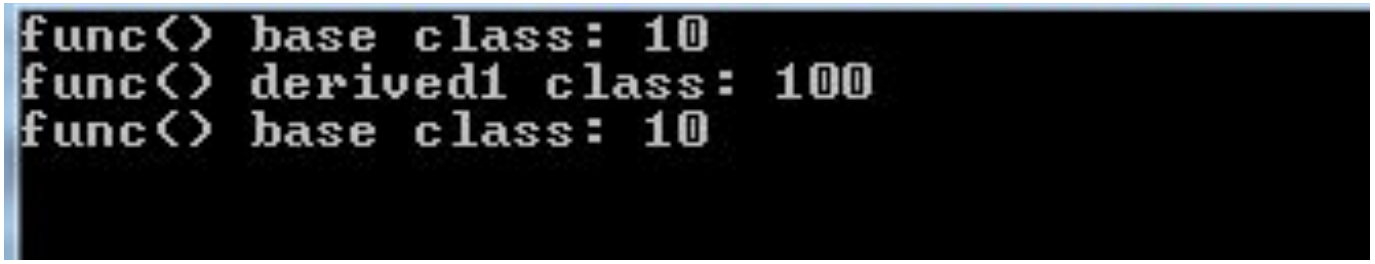
Иерархический порядок наследования виртуальных функций



//Виртуальные функции имеют иерархический порядок наследования

```
#include <iostream>
#include <conio>
using namespace std;
class base {
public:
int i;
base(int x) { i = x; }
virtual void func()
{
cout << "func() base class: ";
cout << i << '\n';
}
};
class deri1 : public base {
public:
deri1(int x) : base(x) { }
```

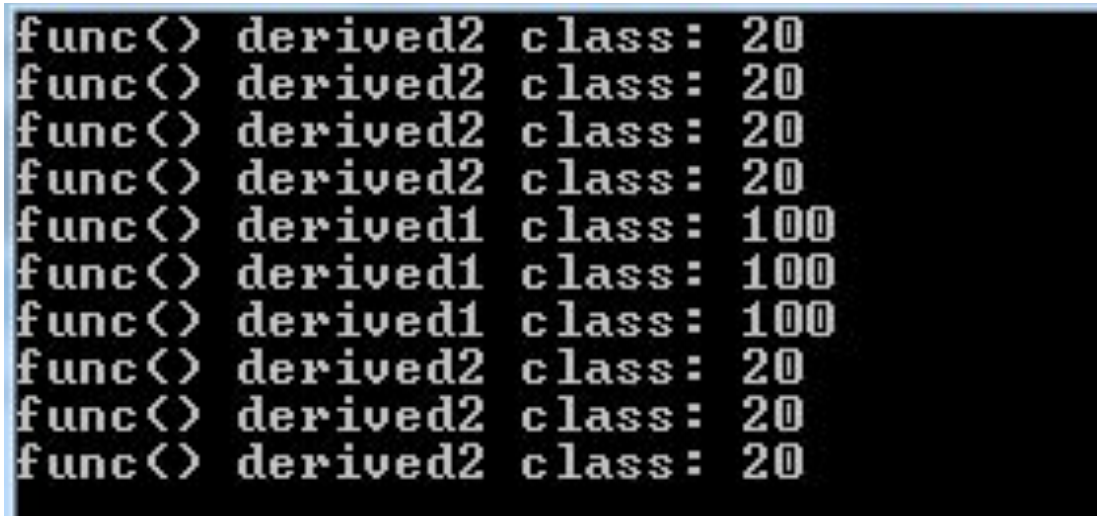
```
int main()
{
base *p;
base obj_base10);
deri1 obj_deri1(10);
deri2 obj_deri2(10);
p = &obj_base;
p->func(); // функция func() базового класса
p = &obj_deri1;
p->func();
// функция func() производного класса deri1
p = &obj_deri2;
p->func(); // функция func() базового класса
getch();
return 0;
}
```



Позднее связывание

```
//Работа виртуальной функции при наличии случайных событий во время выполнения программы
#include <iostream>
#include <cstdlib>
#include <conio>
using namespace std;
class base {
public:
int i;
base(int x) { i = x; }
virtual void func()
{
cout << "func() base class: ";
cout << i << '\n';
}
};
class deri1 : public base {
public:
deri1(int x) : base(x) { }
void func()
{
```

```
int main()
{
base *p;
deri1 obj_deri1(10);
deri2 obj_deri2(10);
int i, j;
for(i=0; i<10; i++) {
j = rand();
if((j%2))   p = &obj_deri1; // если число нечетное
// использовать объект obj_deri1
else p = &obj_deri2; // если число четное
// использовать объект obj_deri2
p->func(); // вызов подходящей версии функции
}
getch();
return 0;
}
```

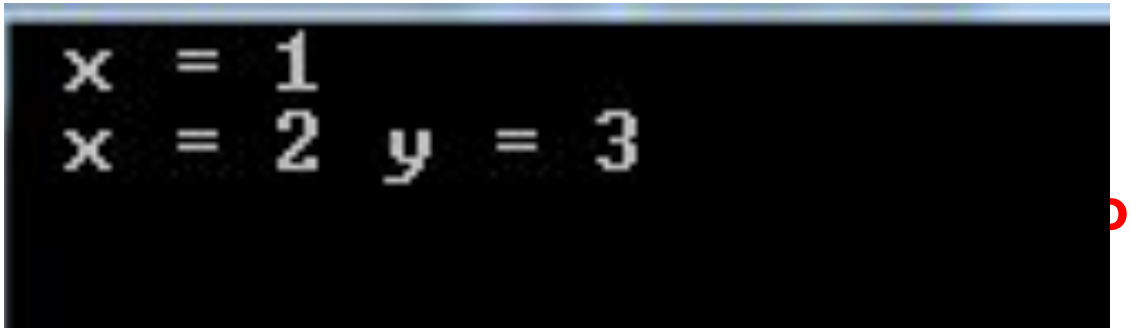


Абстрактный класс

```
//Базовый класс – абстрактный.
#include <iostream>
#include <conio>
using namespace std;
class base // объявление абстрактного класса
{
public:
base() {} // конструктор
~base () {} // деструктор
virtual void get() =0; // чистая //виртуальная функция
};
class deri1: public base//объявление //производного класса
{
protected:
double x;
public:
deri1(double bx) // конструктор
```

```
class deri2: public deri1
//объявление производного класса
{
double y;
public:
deri2(double bx,double by):deri1(bx)
// конструктор
{y = by; }
~deri2 () // деструктор
{ cout << "destructor Point2 done"<<endl;}
void get()
{ cout << " x = "<< x <<" y = "<< y <<endl; }
};
```

```
void main()
{
base *P;
// P = new Point()
// P -> get();
```



ого класса

ДОСТУП К ЭЛЕМЕНТАМ БАЗОВОГО КЛАССА В КЛАССЕ-НАСЛЕДНИКЕ

Доступность элементов базового класса из классов-наследников изменяется в зависимости от спецификаторов доступа в базовом классе и спецификатора наследования:

Спецификатор доступа в базовом классе	Спецификатор наследования	Доступ в производном классе
public	отсутствует	private
public	private	private
public	public	public
public	protected	protected
protected	отсутствует	private
protected	private	private
protected	public	protected
protected	protected	protected
private	отсутствует	недоступны
private	private	недоступны
private	public	недоступны
private	protected	недоступны

- Программы **могут обращаться к частным** (private) элементам класса **только с помощью функций-элементов** этого же класса.
- Использование частных элементов класса вместо общих во всех ситуациях, где это только возможно, уменьшает возможность программы испортить значения элементов класса, так как программа может обращаться к таким элементам только через интерфейсные функции (которые управляют доступом к частным элементам).
- Однако в зависимости от использования объектов программы можно существенно увеличить производительность, позволяя **одному классу напрямую обращаться к частным элементам другого**, что уменьшает время выполнения на вызов интерфейсных функций.

C++ позволяет определить класс в качестве друга (friend} другого класса и разрешает классу-другу доступ к частным элементам этого другого класса

- Дружественные классы могут **обращаться напрямую** к частным элементам другого класса.
- Частные элементы класса защищают данные класса следует  ограничить круг классов-друзей только теми классами, которым действительно необходим прямой доступ к частным элементам искомого класса.
- C++ позволяет ограничить дружественный доступ определенным набором функций.

Чтобы указать C++, что один класс является другом (friend) другого класса, следует указать ключевое слово friend и имя соответствующего класса-друга внутрь определения другого класса.

Форма доступа – **дружественные структуры:**

- **Дружественные функции;**
- **Дружественные классы;**
- **Дружественные функции-элементы.**

Дружественная функция по отношению к какому-либо классу получает такие же привилегии доступа, какими обладает функция-элемент этого класса.

Например, в следующем примере функция frd() объявлена другом класса cl:

```
class cl {
...
 public:
   friend void frd();
...
};
```

```
#include <iostream.h>
#include <conio>
class XXX;
/*Неполное объявление класса.
Оно необходимо для объявления типа параметра функции-члена
для следующего класса*/
class MMM
{
 private:
  int m1;
 public:
  MMM(int val);
  void TypeVal(char *ObjectName, XXX& ClassParam);
};

MMM::MMM(int val)
{ m1 = val;}
/*Определение функции-члена TypeVal располагается после
объявления класса XXX. Только тогда транслятор узнает о структуре
```

```
class XXX
{
 friend class YYY;
 friend void MMM::TypeVal(char *ObjectName, XXX& ClassParam);
 friend void TypeVal(XXX& ClassParamX, YYY& ClassParamY);
```

/*В классе объявляются три друга данного класса: класс **YYY**, функция-член класса **MMM**, простая функция TypeVal.

В класс XXX включаются лишь объявления дружественных функций и классов. Все определения

располагаются в других местах - там, где им и положено быть - в своих

собственных областях видимости.*/

```
 private:
  int x1;
 public:
  XXX(int val);
};
XXX::XXX(int val)
{ x1 = val;}
void MMM::TypeVal(char *ObjectName, XXX& ClassParam)
```

```
class YYY
{
 friend void TypeVal(XXX& ClassParamX, YYY& ClassParamY);
 private:
  int y1;
 public:
  YYY(int val);
  void TypeVal(char *ObjectName, XXX& ClassParam);
};
YYY::YYY(int val)
{
 y1 = val;
}
void YYY::TypeVal(char *ObjectName, XXX& ClassParam)
{
 cout << "Значение " << ObjectName << ": " << ClassParam.x1 << endl;
}
```

```
void main()
{
 XXX mem1(1);
 XXX mem2(2);
 XXX mem3(3);
 YYY disp1(1);
 YYY disp2(2);
 MMM special(0);
 disp1.TypeVal("mem1", mem1);
 disp2.TypeVal("mem2", mem2);
 disp2.TypeVal("mem3", mem3);
 special.TypeVal("\n mem2 from special spy:", mem2);
 TypeVal(mem1, disp2);
 TypeVal(mem2, disp1);
}

void TypeVal(XXX& ClassParamX, YY
{
 cout << endl;
```

```
Znachenie mem1: 1
Znachenie mem2: 2
Znachenie mem3: 3
Znachenie
 mem2 from special spy:: 2

???.x1 == 1
???.y1 == 2
```

Пример:

```
Class book
{
public:
   book (char *, char *, char *);
   void show_book(void);
   friend librarian;
private:
   char title [60] ;
   char author[60];
   char catalog[30];
};
```

Исправить следующую программу!

```
#include <iostream.h>
#include <string.h>
class book
{
public:
  book (char *, char *, char *);
  void show_book(void);
  friend librarian;
private:
  char title[60] ;
  char author[60];
  char catalog[30];
};
book::book(char *title, char *author, char *catalog)
{
  strcpy(book::title, title);
  strcpy(book::author, author) ;
```

```
class librarian
{ public:
   void chan_catal(book *, char *);
   char *get_catalog(book);
};
void librarian::chan_catal(book *this_book, char *new_catalog)
{ strcpy(this_book->catalog, new_catalog);
}
char *librarian: :get_catalog(book this_book)
{    static char catalog[30];
   strcpy(catalog, this_book.catalog);
   return(catalog) ;
}
void main(void)
{
   book programming( «Язык C++", «Пратта", «N2.01");
   librarian library;
```

Резюме

- *Обычно единственный способ, с помощью которого программы могут обращаться к частным элементам класса, заключается в использовании интерфейсных функций.*
- *В зависимости от использования объектов иногда может быть удобным (или более эффективным с точки зрения скорости вычислений) разрешить одному классу обращаться к частным элементам другого.*
- *Для этого необходимо информировать компилятор C++, что класс является другом (friend). Компилятор, в свою очередь, позволит классу-другу обращаться к частным элементам требуемого класса.*
- *Для объявления класса другом, следует поместить ключевое слово friend и имя класса-друга в секцию public определения класса.*
- *Классы-друзья C++ обычно не связаны между собой узами наследования.*

```
//  Описание класса A
class A
{
//...
void z();  // Описание функции  z  класса  A
};
// Описание класса B
class B
{
//...
friend void A::z();  // Описание функции  z  класса
A  как дружественной
// классу B, т.е. из функции  z  класса  A  можно
// получить доступ к внутренним переменным класса B
};
//  Описание класса  C
class C
{
//...
friend class A;  // Описание класса  A  как
дружественного классу  C,
 //  все функции класса  A  будут дружественны
классу C и
 //  из любой функции класса  A  можно получить
доступ к
 //  внутренним переменным класса  C
}
```

Дружественная данному классу функция

- не является членом этого класса она не может быть вызвана из объекта класса, для которого она объявлена другом, при помощи операции доступа к члену класса (.);
- может быть функцией-членом другого ранее объявленного класса при этом само определение дружественной функции приходится располагать после объявления класса, другом которого была объявлена данная функция;
- не имеет this указателя для работы с классом, содержащим ее объявление в качестве дружественной функции. Дружба - это всего лишь дополнение принципа инкапсуляции;
- не имеет доступа к членам производного класса, чьи базовые классы содержали объявление этой функции.

Лекция окончена
Спасибо за внимание