華中科技大學

面向对象程序设计

第七章 虚函数

许向阳

xuxy@hust.edu.cn

大纲

7.1 多态性

7.2 静态联编和动态联编

7.3 虚函数

7.4 纯虚函数和抽象类

7.5 虚析构函数

7.6 运行时多态的实现机理

7.1 多态性

菜农：

白菜

商贩：

食客：

7.1 多态性

水果：买水果，水果叫什么名字，水果什么价格。

在不同的水果摊位前，可以使用同一样的词汇。

一词多义

```
苹果老板：SoldFruit() {
            cout<<“好吃的苹果，便宜呀”；
          }
香蕉老板：SoldFruit(){
            我这有生一点的香蕉，也有熟香蕉；
            您要哪一种？
            ……
          }
```

7.1 多态性

静态多态

相同的函数名称，但参数不同

动态多态

- 相同的函数名称、相同的参数
- 分属不同的类；基类与多个派生类

多态性：

- 具有相似功能的不同函数使用同一名称
- 用相同的调用方式，调用不同功能的同名函数。

7.2 静态联编和动态联编

多态性的实现———联编

联编：绑定、装配

将一个标识符名和一个地址联系在一起。

静态联编：前期联编、早期联编

在生成可执行程序时已经完成。

编译时多态性：函数重载、运算符重载

动态联编：晚期联编、后期联编

程序运行时才动态完成。

运行时多态：使用虚函数

7.3 虚函数

virtual 函数类型　函数名(形式参数表);

基类中，在函数声明时，加”**virtual**”

```
class  fruit
{
public:
        virtual void EatFruit()
        {
                cout << "Can not eat abstract fruit !"<<endl;
        }
};
```

7.3 虚函数


```
class apple : public fruit  {
    public:
            void  EatFruit()
            {
                    cout << " I like to eat apple." <<endl;
            }
};

class banana : public fruit {
    public:
            void  EatFruit()
            {  cout << " Banana, Ha Ha." <<endl;  }
};
```

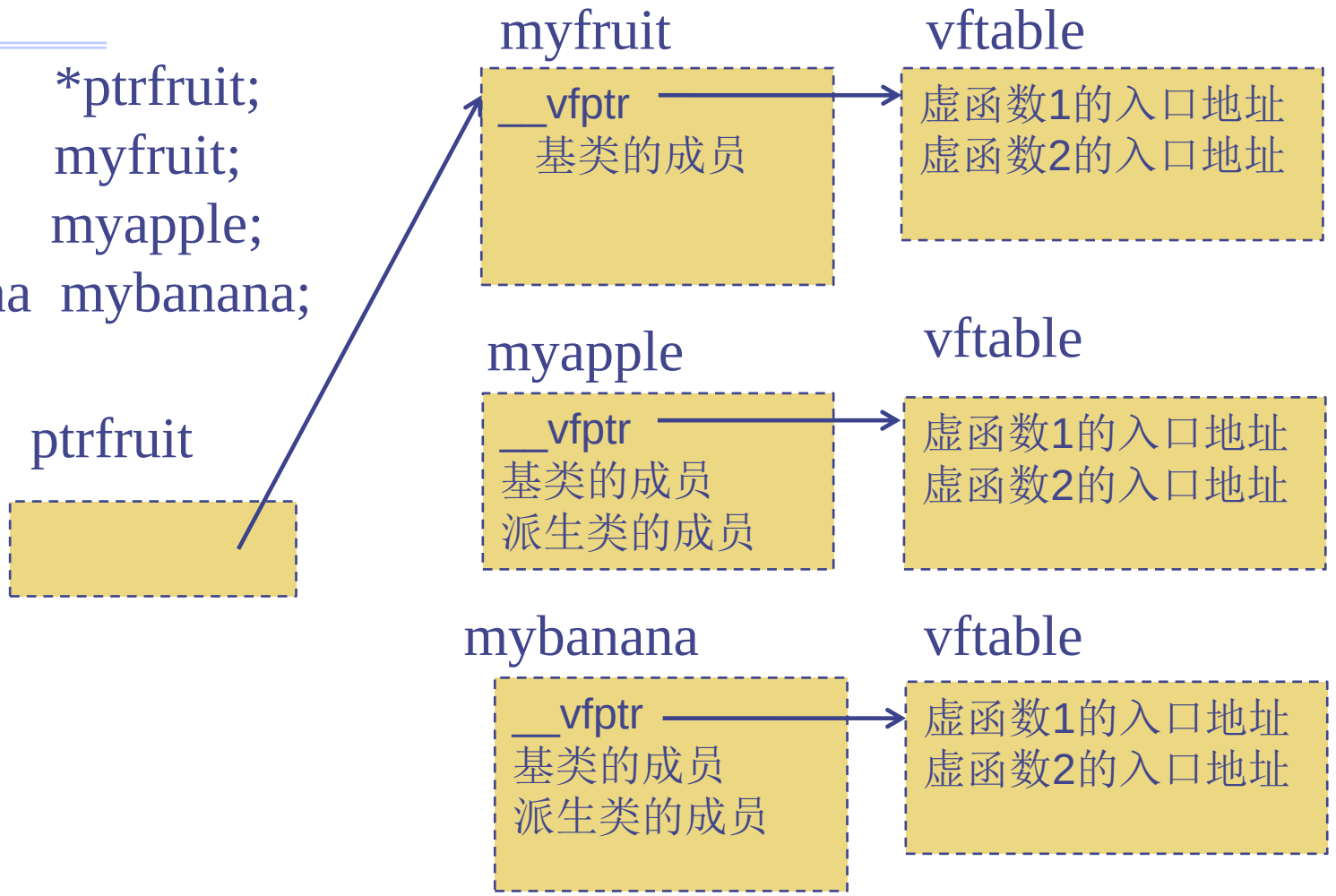
7.3 虚函数


```
fruit      *ptrfruit;
apple    *ptrapple;
banana  *ptrbanana;

int  i;
cout << "select fruit : 1 apple , other banana "<<endl;
cin >>i;
if (i==1)       ptrfruit = ptrapple;
else     ptrfruit = ptrbanana;

ptrfruit->EatFruit();
```

如果将 **virtual** 去掉，程序的运行结果是什么？

7.3 虚函数


```
fruit     *ptrfruit;
fruit     myfruit;
apple     myapple;
banana    mybanana;
```



7.3 虚函数

名称	值
ptrfruit	0x00acfce8
_vfptr	0x002a78d4 const fruit::`vftable'
[0]	0x002a1235 fruit::DispFruitName(void)
[1]	0x002a1023 fruit::EatFruit(void)

```
fruit      *ptrfruit;
fruit      myfruit;
ptrfruit = &myfruit
ptrfruit->EatFruit();
```

ptrfruit	0x00acfcbc {cAppleName=0x00acfcc0 "Red Apple" }
[apple]	{cAppleName=0x00acfcc0 "Red Apple" }
fruit	{...}
_vfptr	0x002a7924 const apple::`vftable'
[0]	0x002a1032 apple::DispFruitName(void)
[1]	0x002a1208 apple::EatFruit(void)
cAppleName	0x00acfcc0 "Red Apple"
_vfptr	0x002a7924 const apple::`vftable'
[0]	0x002a1032 apple::DispFruitName(void)
[1]	0x002a1208 apple::EatFruit(void)

```
apple  myapple
ptrfruit = &myapple;
ptrfruit->EatFruit();
```

ptrfruit =(fruit *) &myapple;

不是用public 派生控制时，强制类型转换

7.3 虚函数


```
fruit      *ptrfruit;
ptrfruit->EatFruit();

ptrfruit->EatFruit();
mov        eax,dword ptr [ebp-24h]   // ptrfruit的地址是 [ebp-24h]
mov        edx,dword ptr [eax]       // 指向的对象中的4个字节
mov        esi,esp
mov        ecx,dword ptr [ebp-24h]
mov        eax,dword ptr [edx+4]     // 函数的入口地址
call       eax
```

如果将 virtual 去掉，程序的运行结果是什么？

7.3 虚函数

将 **virtual** 去掉，程序的运行结果是什么？

```
fruit      *ptrfruit;
ptrfruit->EatFruit();

mov        ecx,dword ptr [ebp-24h]
call       fruit::EatFruit (0EC1028h)
```

7.3 虚函数

- 虚函数一般在基类的public或protected部分。
- 在派生类定义取代型虚函数时，函数原型应和基类的虚函数相同。
- 在派生类，无论是否使用 **virtual** 保留字都将成为虚函数。
- 虚函数只有在具有继承关系的类层次中才需要表现多态。

```
fruit  :            *fruitptr;
    virtual  void  EatFruit();

Apple :             fruitptr=&o_apple;
     void  EatFruit( );       // 虚函数          fruitptr->EatFruit();
     void  EatFruit(int x);  // 新增的成员函数
```

7.3 虚函数

- 构造函数不能定义为虚函数

  构造对象时类型是确定的，不需根据类型不同表现多态行为。

- 析构函数可定义为虚函数

  派生类的析构函数可通过父类指针、引用或**delete**调用。

讨论：在派生类中对基类的虚函数未重新定义，结果如何？

7.3 虚函数

函数重载

- 函数名称相同，参数不同；
- 可以是成员函数和非成员函数；
- 以传递参数的差别，确定调用哪一个函数。

虚函数

- 函数名称相同、参数、返回值完全相同；
- 只能是成员函数；
- 根据对象的不同，去调用不同类的虚函数。

7.4 纯虚函数和抽象类

virtual 函数类型　函数名(形式参数表) = 0;

纯虚函数没有函数体。

函数体由派生类给出。

含有纯虚函数的类：抽象类。

设计抽象类的目的：

建立一个公有的接口，动态的使用它的成员函数。

7.4 纯虚函数和抽象类

- 抽象类常用作派生类的基类，不能有具体的对象（不能有抽象类定义的变量、常量或new产生）。

- 如果派生类没有重新定义该基类的虚函数，或者定义了基类所没有新的纯虚函数，则派生类也会成为抽象类。

- 在多级派生的过程中，如果到某个派生类为止，所有纯虚函数都已在派生类中全部重新定义了函数体，则该派生类就会成为具体类。

7.4 纯虚函数和抽象类


```
class A {
public:
    virtual void f1()=0;
    virtual void f2()=0;
    void f3() {cout<<“A3"<<endl;}
};
void A::f1() {cout<<"A1 "<<endl;}
void A::f2() {cout<<"A2 "<<endl;}
```

A 为抽象类

- 尽管在A的体外定义了f1, f2
- 可以有非虚函数 f3
- // A a1; 抽象类不能定义任何对象

7.4 纯虚函数和抽象类

華中科技大學

```
class B : public A {
private:
   void f2()    {
        this->A::f2();
        cout<<"B2 "<<endl;
    }
};
```

B 为抽象类

- B 中未重新定义 f1
- B 中重新定义了 f2

7.4 纯虚函数和抽象类


```
class C:public B {
private:
   void f1() {cout<<"C1 "<<endl;}
public:
   void f4() {cout<<“f4 "<<endl;}
};
void main(void)
{    C  c;
     A  *p=(A *)&c;
     p->f1();     //   显示 C1
     p->f2();     //   显示 A2、B2
     p->f3();     //   显示 A3
     p->f4();    // f4 不是A类的成员
}
```

A：
public: f1、f2、f3

7.4 纯虚函数和抽象类


```
class C:public B {
private:    void f1() {cout<<"C1 "<<endl;}
public:     void f4() {cout<<“f4 "<<endl;}
};
void main(void)
{     C  c;
      A  a1;      // A 无法实例化抽象类
      B  b1;      //  B 无法实例化抽象类
      A  *a2;
      a2= new A;  //  A 无法实例化抽象类
      c.f1();           //  无法访问私有成员（在C类中申明）
      c.f2();           //  无法访问私有成员（在B类中申明）
}
```

7.4 纯虚函数和抽象类


```
class C:public B {
private:    void f1() {cout<<"C1 "<<endl;}
public:     void f4() {cout<<“f4 "<<endl;}
};
void main(void)
{     C  c;
      c.f1();          //  无法访问私有成员（在C类中申明）
      c.A::f1();     //   f1 在 A是public,  A ->public 派生 B
                       //                B  ->public 派生 C
      c.f4();
      c.f2();   // 等同于 c.B::f2(); 无法访问私有成员
      c.A::f2();    //显示 A2
}
```

7.5 虚析构函数

- 如果基类的析构函数定义为虚析构函数，
  则派生类的析构函数就会自动成为虚析构函数。

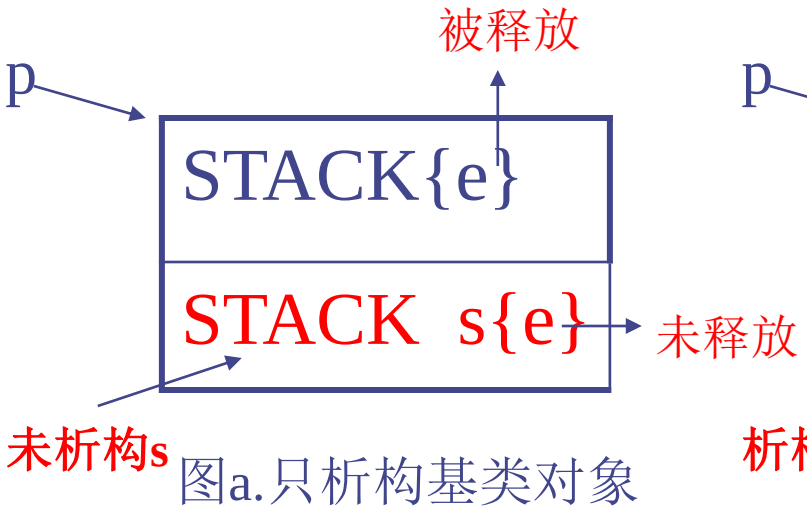
- 在使用delete运算符删除一个对象时，
  为了保证执行的析构函数就是该对象自己的析构函数，
  应将析构函数定义为虚析构函数。

7.5 虚析构函数

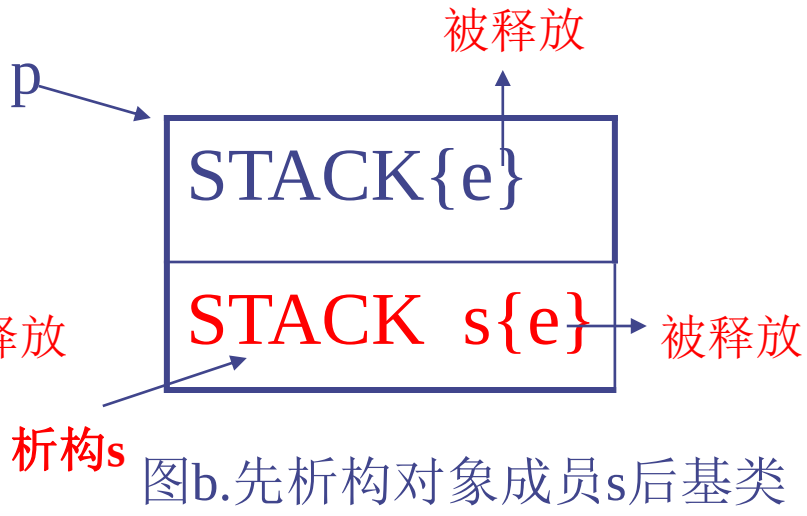

```
class STACK{
   int *e, p, c;
public:
   virtual  int  getp( ){ return p; }
   virtual  int  push(int f){ return  p<c?(e[p++]=f,1): 0; }
   virtual  int  pop (int&f){ return  p>0?(f=e[--p],1): 0; }
   STACK(int m): e(new int[m]), c(e?m:0), p(0){ }
   virtual ~STACK( ){ if(e){ delete e; e=0; c=0; p=0;}}
};
class QUEUE: public STACK{//公有派生，基类和派生类构成父子关系
   STACK s;
public:
   virtual int enter(int f) { return s.getp( ) ? push(f): s.push(f); }
   virtual int leave(int&f) { if (!s.getp( )) while(pop(f)) s.push(f);  return
    s.pop(f); }
   QUEUE(int m): STACK(m), s(m){  }
   ~QUEUE( ){ }
};
void main(void){  STACK *p=new QUEUE(9); delete  p; }
```

7.5 虚析构函数

- 如果~STACK没定义为虚函数，则delete p调用析构函数~STACK，只释放基类对象成员e占用的内存，未析构对象成员s
- 如果~STACK定义为虚函数，则delete p调用析构函数~QUEUE，把QUEUE(9)当作QUEUE对象析构(如图b)。



图a.只析构基类对象



图b.先析构对象成员s后基类

7.6 运行时多态的实现机理

- 编译程序为有虚函数的类创建一个虚函数入口地址表VFT，
- 表首地址存放在对象的起始单元中。
- 当对象调用虚函数时，通过其起始单元得到VFT首址，动态绑定到相应的函数成员。

7.6 运行时多态的实现机理

设基类A和派生类B都有虚函数，对应的虚函数入口地址表分别为VFT_A和VFT_B。

VFTA

在 A类中申明的虚函数

VFTB

在 B类中申明的虚函数

A类中未被取代的虚函数

派生类对象b构造阶段

- 先将VFT_A的首地址存放到b的起始单元
- 在A类构造函数的函数体执行时，A类对象调用的虚函数与VFT_A绑定，执行的虚函数将是A类的函数；
- 在B类构造函数的函数体执行前，将VFT_B的首地址存放到b的起始单元，绑定和执行的将是B类的函数。
- 如果B类没有定义这样的函数，根据面向对象的作用域，将调用基类A的相同原型的函数。

7.6 运行时多态的实现机理

生存阶段：

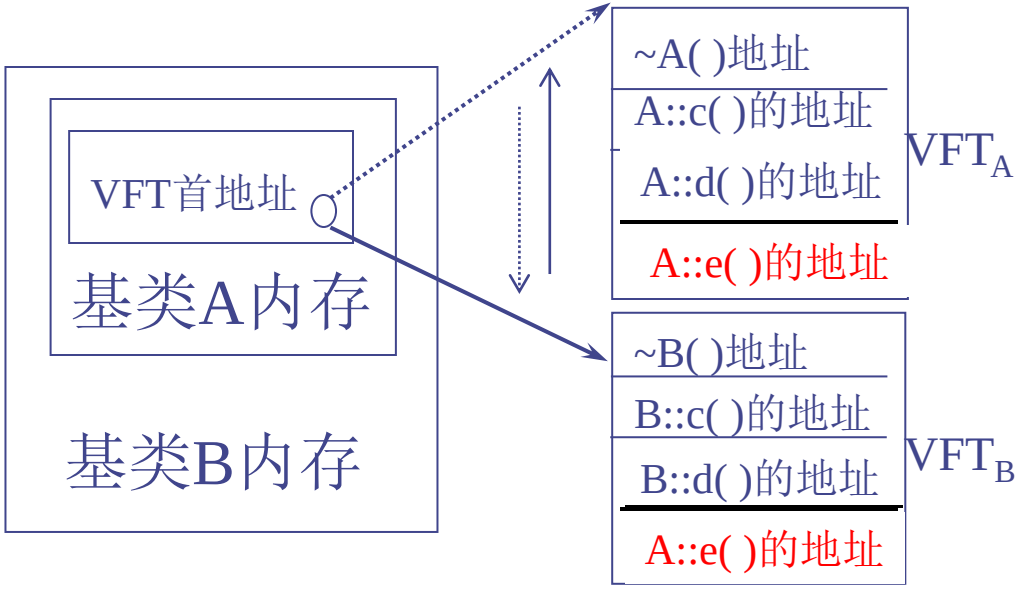
b的起始单元指向VFTB，绑定和执行的将是B类的函数。如果B类没有定义这样的函数，根据面向对象的作用域，将调用基类A的相同原型的函数。

析构阶段：

b的起始单元指向VFTB，绑定和执行的方式同上；在b的析构函数执行完后、基类的析构函数执行前，将VFTA首地址存放到b的起始单元，此后绑定和执行的将是A的函数。如果A类没有定义这样的函数，根据面向对象的作用域，将调用基类A的相同原型的函数。

7.6 运行时多态的实现机理

```
#include <iostream.h>
class A{
    virtual void c( )
    {cout<<"Construct A\n";}
    virtual void d( )
    {cout<<"Deconstruct A\n";}
    virtual void e( ){ };
public:
    A( ){c( );}
    virtual ~A( ){d( );}
};
class B:A{
    virtual void c( )
    {cout<<"Construct B\n";}
    virtual void d( )
    {cout<<"Deconstruct B\n";}
public:
    B( ){c( );}//等价于B( ):A( ){c( );}
    virtual ~B( ){d( );}//virtual可省
};
```




```
void main(void){  B b;  }
```

输出结果（先构造的后析构：像栈）

```
Construct A
Construct B
Deconstruct B
Deconstruct A
```

总结

- 多态性
- 虚函数
- 纯虚函数和抽象类
- 运行时多态的实现机理