C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

考试题型

单选题　　1分×20题 = 20分

填空题　　1分×15题 = 15分

程序填空题　4分×5题 = 20分

程序分析题　6分×5题 = 30分

程序设计题　　2题 = 15分

（第一题5分，第二题10分）

第六章　多态与虚函数

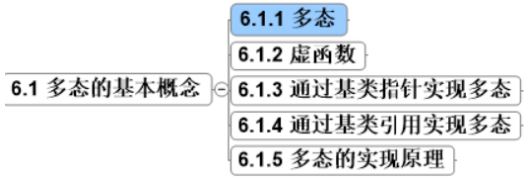
本章主要内容

- 多态的基本概念
- 多态实例
- 多态的使用
- 虚析构函数
- 纯虚函数和抽象类

6.1 多态的基本概念

- 多态分为编译时多态和运行时多态。
- 编译时多态主要是指函数的重载（包括运算符的重载）。对重载函数的调用，在编译时就可以根据实参确定应该调用哪个函数，因此称为编译时多态。**编译阶段的多态**称为静态多态；
- 运行时多态则和继承、虚函数等概念有关。本章中提及的多态主要是指运行时多态。**运行阶段的多态**称为动态多态。

6.1.1 多态



- 程序编译阶段都早于程序运行阶段，所以静态绑定称为早绑定，动态绑定称为晚绑定。静态多态和动态多态的区别，只在于在什么时候将函数实现和函数调用关联起来，是在**编译阶段**还是在**运行阶段**，即函数地址是早绑定的还是晚绑定的。
- 在类之间满足**赋值兼容的前提**下，实现动态绑定必须满足以下两个条件：

  1)必须声明虚函数。

  2)通过**基类类型的引用或者指针**调用虚函数。

课堂练习

程序产生运行时的多态性的前提不包括（ ）。

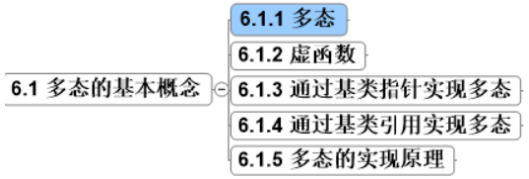
A:类之间的继承关系满足赋值兼容性规则

B:在调用中对虚函数使用成员名限定

C:必须声明虚函数

D:根据赋值兼容性规则使用指针（或引用）

课堂练习



程序产生运行时的多态性的前提不包括（ ）。

A:类之间的继承关系满足赋值兼容性规则

B:在调用中对虚函数使用成员名限定

C:必须声明虚函数

D:根据赋值兼容性规则使用指针（或引用）

答案：B

在类之间满足赋值兼容的前提下，实现动态绑定必须满足以下两个条件。

1)必须声明虚函数。

2)通过基类类型的引用或者指针调用虚函数。

课堂练习

有关多态性说法不正确的是（　）。

A:C++语言的多态性分为编译时的多态性和运行时的多态性

B:编译时的多态性可通过函数重载实现

C:运行时的多态性可通过模板和虚函数实现

D:实现运行时多态性的机制称为动态多态性

课堂练习

有关多态性说法不正确的是（　）。

A:C++语言的多态性分为编译时的多态性和运行时的多态性

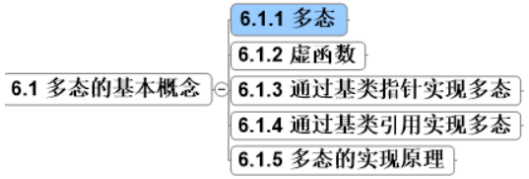
B:编译时的多态性可通过函数重载实现

C:运行时的多态性可通过模板和虚函数实现

D:实现运行时多态性的机制称为动态多态性

答案：C

课堂练习



所谓多态性是指【 】

A.不同的对象调用不同名称的函数

B.不同的对象调用相同名称的函数

C.一个对象调用不同名称的函数

D.一个对象调用不同名称的对象

课堂练习

所谓多态性是指【 】

A.不同的对象调用不同名称的函数

B.不同的对象调用相同名称的函数

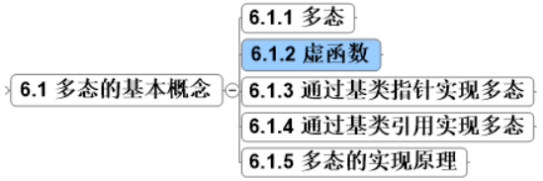
C.一个对象调用不同名称的函数

D.一个对象调用不同名称的对象

答案：B

【分析】多态性有静态多态性和动态多态性两种。静态多态性是指调用同名函数，根据参数的不同而调用不同的同名函数。动态多态性是指不同对象调用同名函数时，由于对象不同而调用不同的同名函数。所以不论是静态还是动态，多态性肯定是调用同名的函数。

6.1.2 虚函数



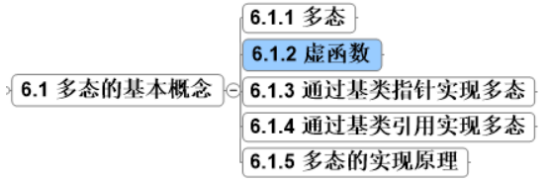
- 所谓“虚函数”，就是在函数声明时前面加了**virtual**关键字的成员函数。virtual关键字只在类定义中的成员函数声明处使用，不能在类外部写成员函数体时使用。静态成员函数不能是虚函数。包含虚函数的类称为“多态类”。

  声明虚函数成员的一般格式如下：

  virtual 函数返回值类型 函数名(形参表);

- 在类的定义中使用virtual关键字来限定的成员函数即成为虚函数。再次强调一下，虚函数的声明只能出现在类定义中的函数原型声明时，不能在类外成员函数实现的时候。
- 派生类可以继承基类的同名函数，并且可以在派生类中重写这个函数。如果不使用虚函数，当使用派生类对象调用这个函数，且派生类中重写了这个函数时，则调用派生类中的同名函数，即“隐藏”了基类中的函数。
- 当然，如果还想调用基类的函数，只需在调用函数时，在前面加上基类名及作用域限定符即可。

6.1.2 虚函数



关于虚函数，有以下几点需要注意。

1)虽然将虚函数声明为内联函数不会引起错误，但因为内联函数是在编译阶段进行静态处理的，而对虚函数的调用是动态绑定的，所以**虚函数一般不声明为内联函数**。

2)派生类重写基类的虚函数实现多态，要求函数名、参数列表及返回值类型要完全相同。

3)基类中定义了虚函数，在派生类中该函数始终保持虚函数的特性。

4)只有类的非静态成员函数才能定义为虚函数，**静态成员函数和友元函数不能定义为虚函数**。

5)如果虚函数的定义是在类体外，则只需在声明函数时添加virtual关键字，定义时不加virtual关键字。

6)**构造函数不能定义为虚函数**。最好也不要将operator=定义为虚函数，因为使用时容易混淆。

7)**不要在构造函数和析构函数中调用虚函数**。在构造函数和析构函数中，对象是不完整的，可能会出现未定义的行为。

8)最好将**基类的析构函数声明为虚函数**。

课堂练习

当一个类的某个函数被声明为virtual时，该函数在该类的所有派生类中【 】

A.都是虚函数

B.只有被重新声明时才是虚函数

C.只有被重新声明为virtual时才是虚函数

D.都不是虚函数

课堂练习

当一个类的某个函数被声明为virtual时，该函数在该类的所有派生类中【 】

A.都是虚函数

B.只有被重新声明时才是虚函数

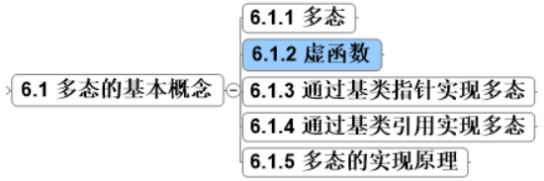
C.只有被重新声明为virtual时才是虚函数

D.都不是虚函数

答案：A。

【分析】在基类声明为virtual的成员函数是虚函数，在派生类中只要有相同的成员函数（函数名相同、返回值类型相同、形参类型和个数相同），即使不使用virtual说明，也都是虚函数。

课堂练习



类B是类A的公有派生类，类A和类B中都定义了虚函数func( )，p是一个指向类A对象的指针，则p->A::func( )【 】

A.调用类A中的函数func( )

B.调用类B中的函数func( )

C.根据p所指的对象类型而确定调用类A中或类B中的函数func()

D.既调用类A中的函数，也调用类B中的函数

课堂练习

类B是类A的公有派生类，类A和类B中都定义了虚函数func( )，p是一个指向类A对象的指针，则p->A::func( )【 】

A.调用类A中的函数func( )

B.调用类B中的函数func( )

C.根据p所指的对象类型而确定调用类A中或类B中的函数func( )

D.既调用类A中的函数，也调用类B中的函数

答案：A。

【分析】p->A::func()中因为有类名限定符，所以明确指示将调用类A中的func()函数，在这种情况下，指针p指向基类或派生类对象，都不影响调用的函数。如果调用语句是p->func()，则要根据指针p的实际指向情况，决定调用类A或类B中的成员函数，这才是动态多态。

课堂练习

下列选项中，调用虚函数时可以实现动态多态的是【 】

A.基类对象指针

B.对象名

C.基类对象

D.派生类名

课堂练习

下列选项中，调用虚函数时可以实现动态多态的是【 】

A.基类对象指针

B.对象名

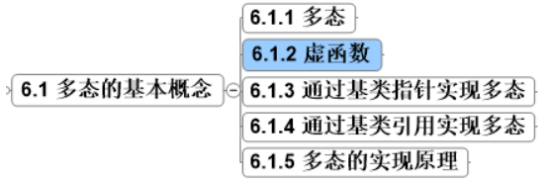
C.基类对象

D.派生类名

答案：A

【分析】实际上，通过**基类指针或基类引用**调用虚函数时，都会产生动态多态。给出的选项中，只有基类对象指针，所以选项A是正确的。

课堂练习



下面函数原型声明中，声明了fun为虚函数的是（　）。

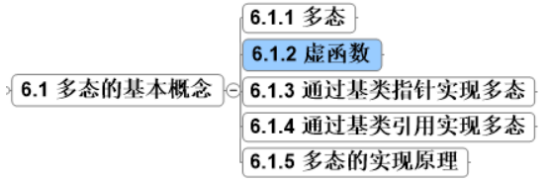
A:void fun()=0

B:virtual void fun()=0

C:virtual void fun()

D:virtual void fun(){}

课堂练习



下面函数原型声明中，**声明**了fun为虚函数的是（　）。

A:void fun()=0

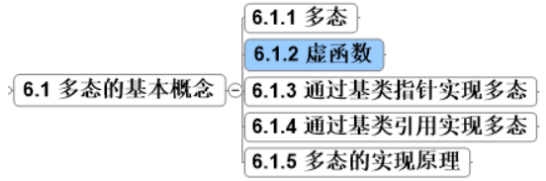
B:virtual void fun()=0

C:virtual void fun()

D:virtual void fun(){}

答案：C

课堂练习



一旦基类定义了虚函数，也自动成为虚函数的是该基类派生类中的（　）。

A:同名函数

B:构造函数

C:析构函数

D:值函数

课堂练习

一旦基类定义了虚函数，也自动成为虚函数的是该基类派生类中的（　）。

A:同名函数

B:构造函数

C:析构函数

D:值函数

答案：A

课堂练习

下列关于虚函数的描述中，正确的是（　）。

A:使用虚函数就一定产生多态性

B:虚函数只能是类中的一个成员函数，但不能是静态成员

C:一个类中仅可以声明一个纯虚函数

D:在构造函数和析构函数中调用虚函数采用动态联编

课堂练习

下列关于虚函数的描述中，正确的是（　）。

A:使用虚函数就一定产生多态性

B:虚函数只能是类中的一个成员函数，但不能是静态成员

C:一个类中仅可以声明一个纯虚函数

D:在构造函数和析构函数中调用虚函数采用动态联编

答案：B

课堂练习

下面关于构造函数和析构函数的描述中，错误的是（　）。

A:析构函数中调用虚函数采用静态联编

B:对虚析构函数的调用可以采用动态联编

C:当基类的析构函数是虚函数时，其派生类的析构函数也一定是虚函数

D:构造函数可以声明为虚函数

课堂练习

下面关于构造函数和析构函数的描述中，错误的是（　）。

A:析构函数中调用虚函数采用静态联编

B:对虚析构函数的调用可以采用动态联编

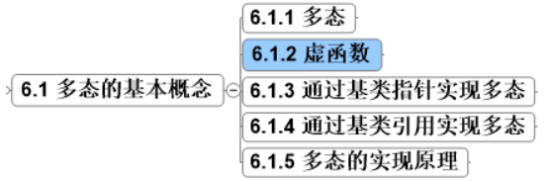
C:当基类的析构函数是虚函数时，其派生类的析构函数也一定是虚函数

D:构造函数可以声明为虚函数

答案：D

常见的不能声明为虚函数的有：全局函数（非成员函数）、静态成员函数、内联成员函数、构造函数和友元函数。

课堂练习



能声明为虚函数的是（　）。

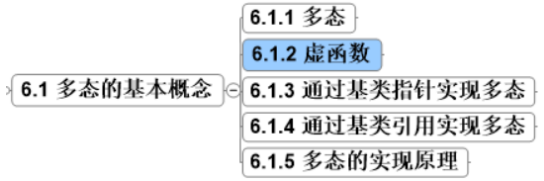
A:类的成员函数

B:静态成员函数

C:内联函数

D:构造函数

课堂练习



能声明为虚函数的是（　）。

A:类的成员函数

B:静态成员函数

C:内联函数

D:构造函数

答案：A

“虚函数”：就是在函数声明时前面加了virtual关键字的成员函数。virtual关键字只在类定义中的成员函数声明处使用，不能在类外部写成员函数体时使用。

常见的不能声明为虚函数的有：全局函数（非成员函数）、静态成员函数、内联成员函数、构造函数和友元函数。

6.1.3 通过基类指针实现多态

声明虚函数后，派生类对象的地址可以赋值给基类指针，也就是基类指针可以指向派生类对象。对于通过基类指针调用基类和派生类中都有的同名、同参数表的虚函数的语句，编译时系统并不确定要执行的是基类还是派生类的虚函数；而当程序运行到该语句时，如果基类指针指向的是一个基类对象，则调用基类的虚函数；如果基类指针指向的是一个派生类对象，则调用派生类的虚函数。

```
#include<iostream>
using namespace std;
class A
{
public:
     virtual void Print() //虚函数
     {
          cout<<"A::Print"<<endl;
     }
};
class B:public A                    //公有继承
{
public:
     virtual void Print() //虚函数
     {
          cout<<"B::Print"<<endl;
     }
};
class D:public A                    //公有继承
{
public:
     virtual void Print() //虚函数
     {
          cout<<"D::Print"<<endl;
     }
};
class E:public B                    //公有继承
{
public:
     virtual void Print()           //虚函数
     {
          cout<<"E::Print"<<endl;
     }
};
int main()
{
     A a;B b;D d;E e;
     A*pa=&a;     //基类pa指指针向基类对象a
     B*pb=&b;     //派生类指针pa指向派生类对象b
     pa->Print()  ; //多态，目前指向基类对象，调用a.Print()，输出A::Print
     pa=pb;          //派生类指针赋给基类指针，pa指向派生类对象b
     pa->Print();   //多态，目前指向派生对象，调用b.Print ()，输出B::Print
     pa=&d;         //基类指针pa指向派生类对象d
     pa->Print();   //多态，目前指向派生类对象，调用d.Print ()，输出D::Print
     pa=&e;         //基类指针pa指向派生对象e
     pa->Print();   //多态，目前指向派生类对象，调用e.Print ()，输出E::Print
     return 0;
}
```

```
A::Print
B::Print
D::Print
E::Print
```

6.1.4 通过基类引用实现多态

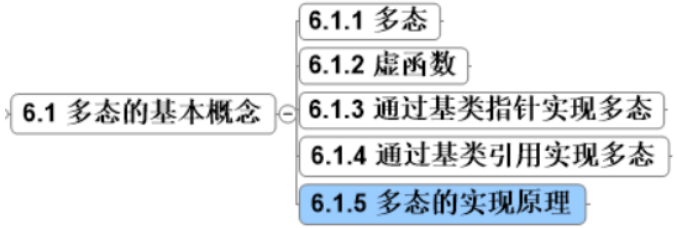
通过基类指针调用虚函数时可以实现多态，通过基类的引用调用虚函数的语句也是多态的。即通过基类的引用调用基类和派生类中同名、同参数表的虚函数时，若其引用的是一个基类的对象，则调用的是基类的虚函数；若其引用的是一个派生类的对象，则调用的是派生类的虚函数，见程序6-3。

6.1.4 通过基类引用实现多态

```
#include<iostream>
using namespace std;
class A         //基类
{public:
        virtual void Print()
        {
                cout<<"A::Print"<<endl;
        }
};
class B:public A                //公有派生
{public:
        virtual void Print()        //虚函数
        {
                cout<<"B::Print"<<endl;
        }
};
void PrintInfo(A &r)
{
        r.Print();      //多态，使用基类引用调用哪个Print（）取决于r引用了哪个类的对象
}
int main()
{       A a;
        B b;
        PrintInfo(a);   //使用基类对象，调用基类中的函数，输出A::Print
        PrintInfo(b);   //使用基类对象，调用派生类中的函数，输出B::Print
        return 0;
}
```

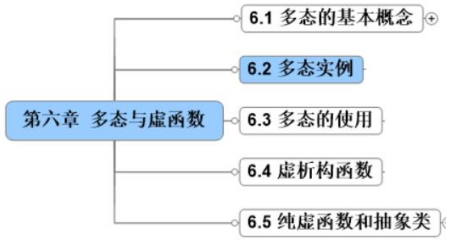
```
A::Print
B::Print
```

6.1.5 *多态的实现原理



多态的关键在于通过基类指针或引用调用一个虚函数时，编译阶段不能确定到底调用的是基类还是派生类的函数，运行时才能确定。

派生类对象占用的存储空间大小，等于基类成员变量占用的存储空间大小加上派生类对象自身成员变量占用的存储空间大小。



6.2 多态实例

定义一个基类CShape表示一般图形，然后派生3个子类分别表示矩形类、圆形类和三角形类。基类中定义了计算图形面积和输出信息的虚函数，3个派生类中均继承了这两个虚函数，可以针对具体的图形计算各自的面积并输出结果。

分别使用指针和引用的display函数

```
#include <iostream>
using namespace std;
const double PI=3.14159;
class Point
{
private:
        double x,y;
public:
        Point(double i,double j) {x=i;y=j;}
        virtual double area(){return 0;}
};
class Circle:public Point {
private:
         double radius;
public:
         Circle(double a,double b,double r):Point(a,b){radius=r;}
         double area() {return PI*radius*radius;}
};
void display(Point *p)  {cout<<p->area()<<endl;}
void display(Point&a)  {cout<<a.area()<<endl;}
```

```
void main()
{
        Point a(1.5,6.7);
        Circle c(1.5,6.7,2.5);
        Point *p=&c;  //派生类对象的地址赋给基类指针
        Point &rc=c;  //派生类对象初始化基类引用
        display(a);     //基类对象调用基类虚函数area，输出0
        display(p);     //指针调用派生类虚函数area，输出19.6349
        display(rc);    //指针调用派生类虚函数area，输出19.6349
}
```

```
0
19.6349
19.6349
```

6.3 多态的使用

在普通成员函数（**静态成员函数、构造函数和析构函数除外**）中调用其他虚成员函数也是允许的，并且是多态的。

```
#include<iostream>
using namespace std;
class CBase                          //基类
{
public:
      void func1()                   //不是虚函数
      {
         cout<<"CBase::func1()"<<endl;
         func2();                    //在成员函数中调用虚函数
         func3();
      }
      virtual void func2()           //虚函数
      {
         cout <<"CBase::func2()"<<endl;
      }
      void func3()                   //不是虚函数
      {
         cout <<"CBase::func3()"<<endl;
      }
};
class CDerived:public CBase          //派生类
{
public:
      virtual void func2( )          //虚函数
      {
            cout<<"CDerived:func2()"<<endl;
      }
      void func3()
      {
            cout<<"CDerived:func3()"<<endl;
      }
};
int main()
{
      CDerived d;
      d.func1();    //调用的的基类中的成员函数
      return 0;
}
```

```
CBase::func1()
CDerived:func2()
CBase::func3()
```

```
#include<iostream>
using namespace std;
class  A                          //基类
{public:
        virtual void hello()        //虚函数
        {
              cout <<"A::hello"<<endl;
        }
        virtual void bye()          //虚函数
        {
              cout <<"A::bye"<<endl;
        }
};
class B:public A                  //派生类
{
public:
       virtual void hello()        //虚函数
       {      cout <<"B::hello"<<endl;  }
       B()
       { hello();} //调用虚函数，但不是多态
       ~B()
       { bye();   //调用虚函数，但不是多态
       }
};
class C:public B                  //派生类
{
public:
       virtual void hello()        //虚函数
       {
             cout <<"C::hello"<<endl;
        }
};

int main()
{
        C obj;
        return 0;
}
```

不仅能在成员函数中调用虚函数，还可以在构造函数和析构函数中调用虚函数，但这样调用的虚函数不是多态的。

```
B::hello
A::bye
```

6.3 多态的使用

- 在构造函数中调用的，编译系统可以据此决定调用哪个类中的版本，所以它不是多态的；
- 在析构函数中调用的，所以也不是多态的；
- 实现多态时，必须满足的条件是：**使用基类指针或引用来调用基类中声明的虚函数。**
- 派生类中继承自基类的虚函数，可以写virtual关键字，也可以省略这个关键字，这不影响派生类中的函数也是虚函数。

```
#include<iostream>
using namespace std;
class  A
{
public:
    void func1()
    {     cout <<"A::func1"<<endl;   }
    virtual void func2() //虚函数
    {    cout <<"A::func2"<<endl;
    }
};

class B:public A
{
public:
    virtual void func1()
    {
        cout <<"B::func1"<<endl;
    }
    void func2()     // func2自动成为虚函数
    {
        cout <<"B::func2"<<endl;
    }
};
class C:public B    //类C以类A为间接基类
{
public:
        void func1()    // func1自动成为虚函数
        {
                cout <<"C::func1"<<endl;
        }
        void func2()     // func2自动成为虚函数
        {
                cout <<"C::func2"<<endl;
        }
};

int main()
{
        C obj;
        A *pa=&obj;
        B *pb=&obj;
        pa->func2();     //多态
        pa->func1();     //不是多态
        pb->func1();     //多态
        return 0;
}
```

```
C::func2
A::func1
C::func1
```

课堂练习

在以下函数中，调用虚函数能实现多态的是（　）

A:普通成员函数

B:构造函数

C:析构函数

D:友元函数

课堂练习

在以下函数中，调用虚函数能实现多态的是（　）

A:普通成员函数

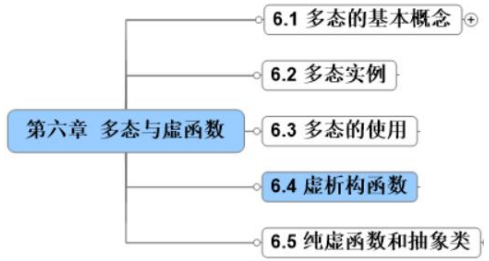
B:构造函数

C:析构函数

D:友元函数

答案：A

6.4 虚析构函数



- 如果一个基类指针指向的对象是用new运算符动态生成的派生类对象，那么释放该对象所占用的空间时，如果仅调用基类的析构函数，则只会完成该析构函数内的空间释放，不会涉及派生类析构函数内的空间释放，容易造成内存泄漏。声明虚析构函数的一般格式如下：

  virtual ~类名( );

- 虚析构函数没有返回值类型，没有参数，所以它的格式非常简单。
- 虚析构函数没有返回值类型，没有参数，所以它的格式非常简单。
- 如果一个类的析构函数是虚函数，则由它派生的所有子类的析构函数也是虚析构函数。使用虚析构函数的目的是为了在对象消亡时实现多态。

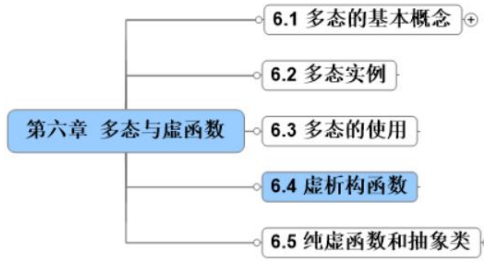
```
#include<iostream>
using namespace std;
class  ABase
{public:
        ABase()
        {    cout<<“ABase构造函数"<<endl;
        }
        ~ABase()
        {
            cout<<"ABase::析构函数"<<endl;
        }
};
class Derived:public ABase
{public:
        int w,h;          //两个成员
        Derived()
        {    cout<<"Derived构造函数"<<endl;
             w=4;  h=7;
        }
        ~Derived()
        {
             cout<<"Derived::析构函数"<<endl;
        }
};
int main()
{
      ABase *p=new Derived();
        //使用基类指针指向new创建的派生类对象
      delete p;
      return 0;
}
```

```
ABase构造函数
Derived构造函数
ABase::析构函数
```

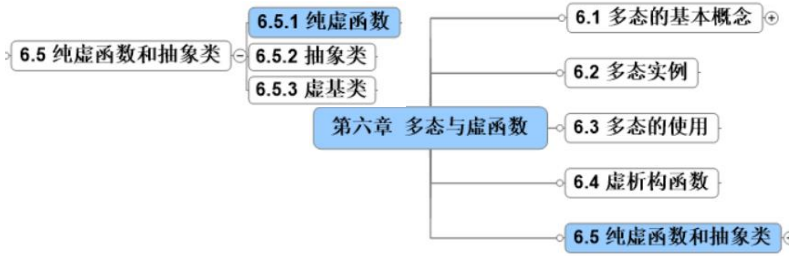
```
#include<iostream>
using namespace std;
class  ABase
{public:
      ABase()
      {    cout<<“ABase构造函数"<<endl;
      }
      virtual ~ABase()
      {
           cout<<"ABase::析构函数"<<endl;
      }
};
class Derived:public ABase
{public:
           int w,h;          //两个成员
           Derived()
           {    cout<<"Derived构造函数"<<endl;
                w=4;  h=7;
           }
           ~Derived()
           {
                cout<<"Derived::析构函数"<<endl;
           }
};
int main()
{
      ABase *p=new Derived();
        //使用基类指针指向new创建的派生类对象
      delete p;
      return 0;
}
```

```
ABase构造函数
Derived构造函数
Derived::析构函数
ABase::析构函数
```

6.4 虚析构函数



- 可以看出，这次不仅调用了基类的析构函数，也调用了派生类的析构函数。
- 只要基类的析构函数是虚函数，那么派生类的析构函数不论是否用virtual关键字声明，都自动成为虚析构函数。
- 一般来说，一个类如果定义了虚函数，则最好将析构函数也定义成虚函数。不过切记，构造函数不能是虚函数。



6.5 纯虚函数和抽象类

- 纯虚函数的作用相当于一个统一的接口形式，表明在基类的各派生类中应该有这样的一个操作，然后在各派生类中具体实现与本派生类相关的操作。
- 纯虚函数是声明在基类中的虚函数，没有具体的定义，而由各派生类根据实际需要给出各自的定义。
- 声明纯虚函数的一般格式如下：

  virtual 函数类型 函数名(参数表)=0;　　例如：virtual void fun( )=0;

- 纯虚函数没有函数体，参数表后要写“=0”。派生类中必须重写这个函数。按照纯虚函数名调用时，执行的是派生类中重写的语句，即调用的是派生类中的版本。

6.5.2 抽象类

包含纯虚函数的类称为**抽象类**。因为抽象类中有尚未完成的函数定义，所以它不能实例化一个对象。抽象类的派生类中，如果没有给出全部纯虚函数的定义，则派生类继续是抽象类。直到派生类中给出全部纯虚函数定义后，它才不再是抽象类，也才能实例化一个对象。**虽然不能创建抽象类的对象，但可以定义抽象类的指针和引用。**这样的指针和引用可以指向并访问派生类的成员，这种访问具有多态性。

6.5.2 抽象类

纯虚函数不同于函数体为空的虚函数，它们的不同之处如下：

1)纯虚函数没有函数体，而空的虚函数的函数体为空。

2)纯虚函数所在的类是抽象类，不能直接进行实例化；而空的虚函数所在的类是可以实例化的。

它们共同的特点是：

纯虚函数与函数体为空的虚函数都可以派生出新的类，然后在新类中给出虚函数的实现，而且这种新的实现具有多态特征。

课堂练习

假设类Aclass为抽象类，则下列说明语句中，正确的是（　）

A:Aclass fun(int);

B:Aclass *p;

C:int fun(Aclass);

D:Aclass Obj;

课堂练习

假设类Aclass为抽象类，则下列说明语句中，正确的是（　）

A:Aclass fun(int);

B:Aclass *p;

C:int fun(Aclass);

D:Aclass Obj;

答案：B

课堂练习

如果A是抽象类，则下面正确的是（　）

A:A中没有纯虚函数

B:A a;

C:A a[3];

D:A *pa;

课堂练习

如果A是抽象类，则下面正确的是（　）

A:A中没有纯虚函数

B:A a;

C:A a[3];

D:A *pa;

答案：D

课堂练习

下列关于纯虚函数与抽象类的描述中，错误的是（　）。

A:纯虚函数是一种特殊的函数，它允许没有具体的实现

B:抽象类是指具有纯虚函数的类

C:一个基类中有纯虚函数，该基类的派生类一定不再是抽象类

D:抽象类只能作为基类来使用，其纯虚函数的实现由派生类给出

课堂练习

下列关于纯虚函数与抽象类的描述中，错误的是（　）。

A:纯虚函数是一种特殊的函数，它允许没有具体的实现

B:抽象类是指具有纯虚函数的类

C:一个基类中有纯虚函数，该基类的派生类一定不再是抽象类

D:抽象类只能作为基类来使用，其纯虚函数的实现由派生类给出

答案：C

```
#include<iostream>
using namespace std;
class  A
{
private:
        int a;
public:
        virtual void print()=0; //纯虚函数
        void func1()
        {
                cout <<"func1"<<endl;
        }
};
class B:public A
{
public:
        void print();
        void func1()
        {
            cout <<"B_func1"<<endl;
        }
};
void B::print()
{
        cout <<"B_Print"<<endl;
}
int main()
{
        //A a;                  //错误，抽象类不能实例化
        //A *p=new A;           //错误，不能创建类A的实例
        //A  b[2];              //错误，不能声明抽象类的数组
         A*pa;                  //正确，可以声明抽象类的指针
         A*pb=new B;            //使用基类指针指向派生类对象
         pb->print();           //调用的是类B中的函数，多态，输出B_Print
         B b;
         A*pc=&b;
         pc->func1(); //因为不是虚函数,调用的是类A中的函数,输出func1
         return 0;
}
```

B_Print
func1

6.5.3 虚基类

定义虚基类的一般格式如下：

class 派生类名： **virtual** 派生方式 基类名

{

派生类体

};

例如，图6-3所示的各类的继承关系如下：

class A

class B : virtual public A

class C : virtual public A

class D : public B, public C

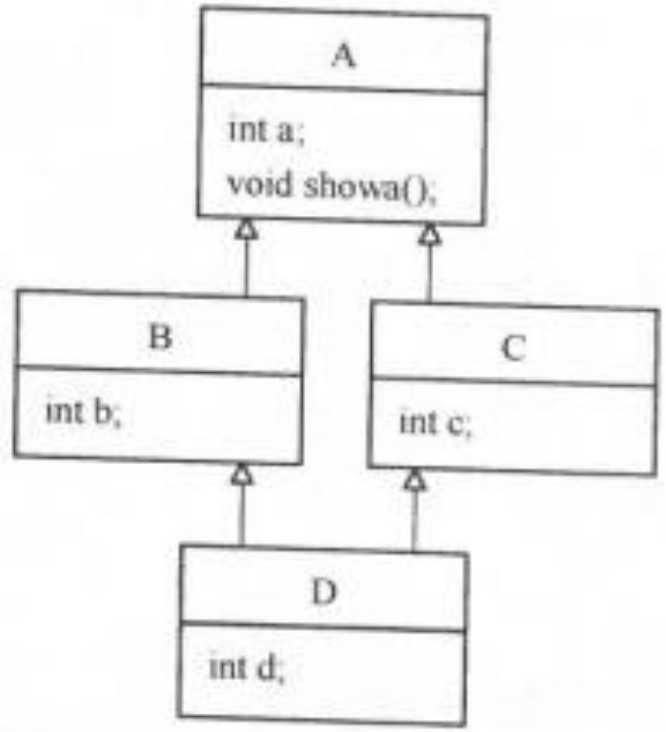


图 6-3　多重继承的模型结构图

6.5.3 虚基类

为了避免产生二义性，C++提供**虚基类机制**，使得在派生类中，继承同一个间接基类的成员仅保留一个版本。

```
#include<iostream>
using namespace std;
class  A
{
public:
    int a;
    void showa()
    {
        cout <<"a="<<a<<endl;
    }
};
class B:virtual public A  //对类A进行了虚继承
{
public:
     int b;
};
class C:virtual public A  //对类A进行了虚继承
{
public:
    int c;
};
class D:public B,public C
//派生类D的两个基类B、C具有共同的基类A,
//采用了虚继承，从而使类D的对象中只包含着类A的1个实例
{
public:
    int d;
};
int main()
{
    D Dobj;      //说明派生类D的对象
    Dobj.a=11; //若不是虚继承，此行会出错！因为"D::a"具有二义性
    Dobj.b=22;
    Dobj.showa();  //输出a=11
        //若不是虚继承，此行会出错！因为"D::showa"具有二义性
    cout<<"Dobj.b="<<Dobj.b<<endl;  //输出Dobj.b=22
}
```

```
a=11
Dobj.b=22
```

课堂练习

通过下列哪一选项调用虚函数，会采用动态联编（　　）。

A:对象指针

B:对象名

C:成员名限定

D:派生类名

课堂练习

通过下列哪一选项调用虚函数，会采用动态联编（　　）。

A:对象指针

B:对象名

C:成员名限定

D:派生类名

答案：A

通过**基类指针或基类引用**调用虚函数时，都会产生动态多态。

课堂练习

有关多态性说法不正确的是（　　）。

A:C++语言的多态性分为编译时的多态性和运行时的多态性

B:编译时的多态性可通过函数重载实现

C:运行时的多态性可通过模板和虚函数实现

D:实现运行时多态性的机制称为动态多态性

课堂练习

有关多态性说法不正确的是（　　）。

A:C++语言的多态性分为编译时的多态性和运行时的多态性

B:编译时的多态性可通过函数重载实现

C:运行时的多态性可通过模板和虚函数实现

D:实现运行时多态性的机制称为动态多态性

答案：C

本章总结


第六章 多态与虚函数
6.1 多态的基本概念
6.1.1 多态
6.1.2 虚函数
6.1.3 通过基类指针实现多态
6.1.4 通过基类引用实现多态
6.1.5 多态的实现原理
6.2 多态实例
6.3 多态的使用
6.4 虚析构函数
6.5 纯虚函数和抽象类
6.5.1 纯虚函数
6.5.2 抽象类
6.5.3 虚基类

祝大家顺利通过考试！