第8讲 C++的多态性-虚函数

目录 CONTENTS

1 多态性的概述

2 虚函数

3 纯虚函数和抽象类

1 多态性的概念

- 函数联编：将源代码中的函数调用解释为执行特定函数代码块。
- 在C语言中，因为每个函数名都对应一个特定的函数，联编过程非常简单。
- 在C++由于函数重载、虚函数的存在，变得非常复杂。
- 对于函数重载，在编译过程中就可以完成联编，称为静态联编（早期联编）。
- 对于虚函数，编译阶段无法知晓对象的类型，，只能在程序运行时才能完成联编，称为动态联编（晚期联编）。

1 多态性的概念

- 多态性是面向对象程序的重要特征之一。简单来说，多态性就是指“一个接口，多种方法”。从实现的角度出发可以分为编译时多态和运行时多态。

- 编译时多态是指在多个函数的函数名（包括运算符）相同的情况下，编译器在编译阶段就能够根据函数参数的个数或类型不同确定要调用的函数。这种静态多态是通过函数重载或运算符重载实现的。

- 运行时多态是指在程序运行时才确定要调用的函数。它是通过虚函数机制实现的。

2 虚函数

- 虚函数允许函数调用与函数体之间的联系在程序运行时才建立，也就是在运行时才决定如何动作。

```
#include<iostream>
using namespace std;
class Base{                                          //声明基类Base
      private:
            int  m_i;
       public:
            Base(int i)
            {  m_i = i; }
            void Show()                    //定义函数Show()
            {
               cout<<"Base----------\n";
               cout<<“m_i: "<< m_i <<endl;
            }
};
```

```
class Derived:public Base{                    //声明派生类Derived
        private:
                int m_j;
        public:
                Derived(int i,int j):Base(i)
                {  m_j =j; }
                void Show()                   //重新定义函数Show()
                {  cout<< "Derived---------\n";
                   cout<< “m_j "  <<m_j<<endl;
                }
};
int main()
{          Base b(100),*p;                    //定义基类对象b和对象指针pc
           Derived d(200,300);                //定义派生类对象d
           p=&b;                              //对象指针p指向基类对象b
           p->Show();                         //调用基类Base的虚函数Show()
           p=&d;                              //对象指针p指向派生类对象d
           p->Show();                         //调用派生类Derived的虚函数Show()
           return 0;
}
```

程序运行结果如下：

```
Base---------
m_i:100
Base---------
m_i:200
```

指向公有派生对象的基类指针

- 在C++中规定：基类的对象指针可以指向它的公有派生的对象，但是当其指向公有派生类对象时，
  - 它只能访问派生类中从基类继承来的成员，
  - 而不能访问公有派生类中定义的成员。

```
#include<iostream>
using namespace std;
class Base{                                          //声明基类Base
      private:
            int  m_i;
       public:
            Base(int i)
            {  m_i = i; }
            virtual void Show()                      //定义函数Show()
            {
               cout<<"Base----------\n";
               cout<<“m_i: "<< m_i <<endl;
            }
};
```

```
class Derived:public Base{                    //声明派生类Derived
        private:
                int m_j;
        public:
                Derived(int i,int j):Base(i)
                {  m_j =j; }
                void Show()                   //重新定义虚函数Show()
                {  cout<< "Derived---------\n";
                   cout<< “m_j "  <<m_j<<endl;
                }
};
int main()
{          Base b(100),*p;                     //定义基类对象b和对象指针pc
           Derived d(200,300);                 //定义派生类对象d
           p=&b;                               //对象指针p指向基类对象b
           p->Show();                          //调用基类Base的虚函数Show()
           p=&d;                               //对象指针p指向派生类对象d
           p->Show();                          //调用派生类Derived的虚函数Show()
           return 0;
}
```

程序运行结果如下：

```
Base---------
m_i:100
Derived--------
m_j:300
```

虚函数的作用和定义

- 使用对象指针的目的就是为了表达一种动态的性质，即当指针指向不同对象时执行不同的操作。
- 关键字virtual指示C++编译器，函数调用“p->Show();”要在运行时确定所要调用的函数，即要对该调用进行动态联编。因此，程序在运行时根据指针p所指向的实际对象，调用该对象的成员函数。
- 把使用同一种调用形式“p->Show()”，调用同一类族中不同类的虚函数称为动态的多态性，即运行时的多态性。

虚函数的定义

- 虚函数就是在基类中被关键字virtual说明，并在派生类中重新定义的函数。
- 定义虚函数的格式如下：

```
virtual  函数类型 函数名(形参表)
{
    函数体
 }
```

虚函数的定义

- 虚函数的定义是在基类中进行的，它是在基类中将需要定义为虚函数的成员函数的声明中冠以关键字virtual，从而提供一种接口界面。
- 虚函数在派生类中重新定义时，其函数原型，包括函数类型、函数名、参数个数与参数类型的顺序，都必须与基类中的函数原型完全相同。

说明

(1) 若在基类中,只声明虚函数原型(需加上virtual)，而在类外定义虚函数时，则不必再加virtual。

例如：

```
class B {
 public:
  virtual void Print(char* p);
};

 void B::Print(char* p )
 { cout<<p<<“Print()"<<endl; }
```

声明虚函数原型,需加上virtual

在类外,定义虚函数时,不要加virtual

说明

(2) 如果在派生类中没有对基类的虚函数重新定义，则公有派生类继承其直接基类的虚函数。一个虚函数无论被公有继承多少次，它仍然保持其虚函数的特性。

例如:

```
class B{
  . . .
 public:
  virtual void Print(){. . .}
};
class D:public B{
   ...
};
```

在基类B中,定义Print为虚函数

若在公有派生类D中没有重新定义虚函数Print,
则函数Print在派生类中被继承,仍是虚函数。

说明

(3) C++规定,当一个成员函数被定义为虚函数后，其派生类中符合重新定义虚函数要求的同名函数都自动成为虚函数。

因此，在派生类中重新定义该虚函数时，关键字virtual可以写也可以不写。

但是，为了使程序更加清晰，最好在每一层派生类中定义该函数时都加上关键字virtual。

说明

```
class B{
   . . .
  public:
   virtual void Print(){. . .}
 };

class D:public B{
    ...
   virtual void Print(){. . .}
 };
```

说明

(4) 虚函数必须是其所在类的成员函数，而不能是友元函数，也不能是静态成员函数，因为虚函数调用要靠特定的对象来决定该激活哪个函数。

说明

(5) 虽然使用对象名和点运算符的方式也可以调用虚函数，但是这种调用是在编译时进行的，是静态联编，它没有利用虚函数的特性。

只有通过基类指针/基类引用访问虚函数时才能获得运行时的多态性。

说明

(6) 构造函数不能是虚函数。因为创建派生类对象时，将调用派生类的构造函数，而不是基类的构造函数，然后派生类构造函数再调用基类的构造函数，这个顺序不同于继承机制，因此派生类不继承构造函数，虚构造函数也就没有意义。

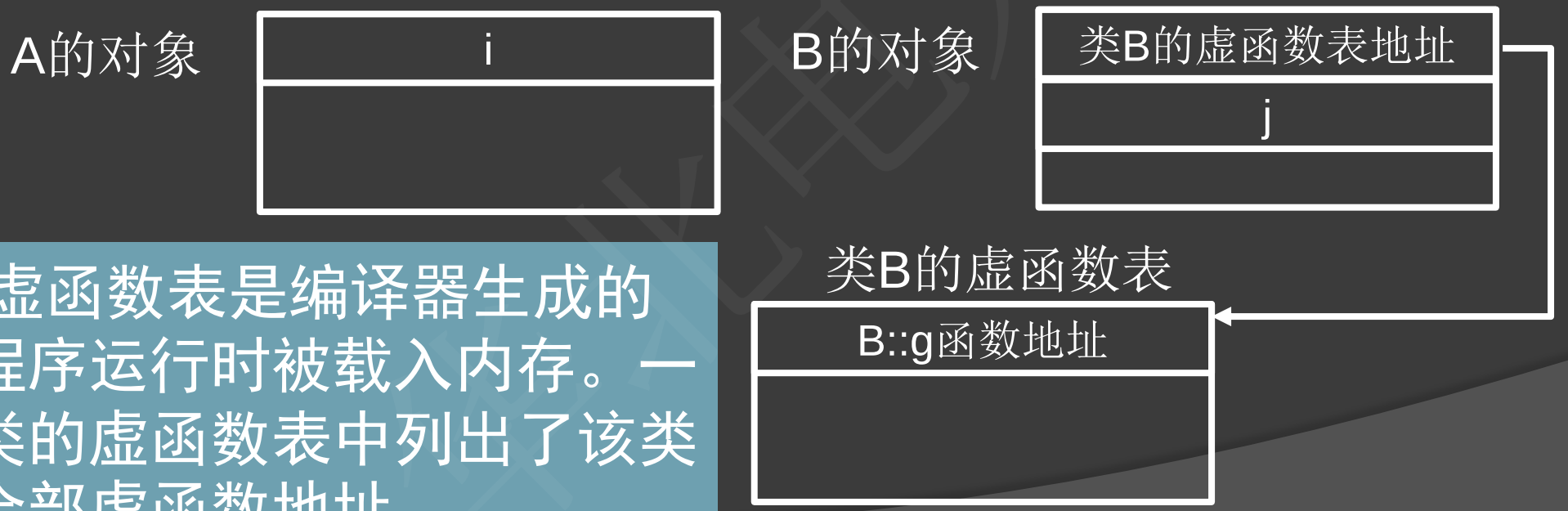
使用虚函数实现多态的原理

```
class A
{
        int i;
public:
        void f(){}
};
class B
{
        int j;
public:
        virtual void g(){}
};
int main()
{
cout<<sizeof(A)<<","<<sizeof(B);
  return 0;  }
```

程序运行的结果为:
4，8

使用虚函数实现多态的原理

任何含有虚函数的类及其派生类的对象都比没有虚函数时多了 4 个字节，这 4 个字节就是实现多态的关键——它位于对象存储空间的最前端，其中存放的是虚函数表的地址。



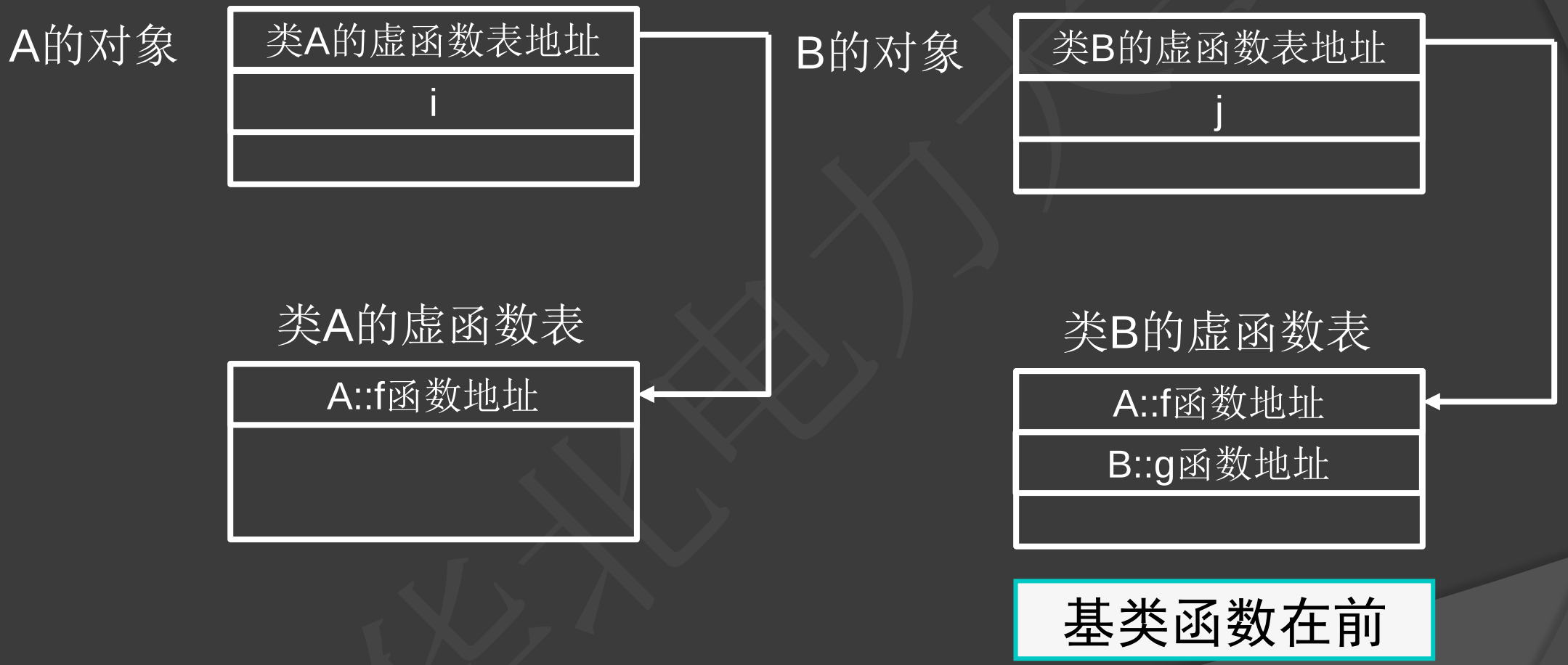
虚函数表是编译器生成的，程序运行时被载入内存。一个类的虚函数表中列出了该类的全部虚函数地址。

使用虚函数实现多态的原理

```
class A
{
        int i;
public:
         virtual void f(){}
};
class B: public A
{
        int j;
public:
        virtual void g(){}
};
```

如果子类的虚函数没有覆盖父类的虚函数，则对多态没有任何影响

使用虚函数实现多态的原理



使用虚函数实现多态的原理

```
class A
{
        int i;
public:
         virtual void f(){}
};
class B: public A
{
        int j;
public:
        virtual void f(){}
        virtual void g(){}
};
```

如果派生类的虚函数覆盖基类的虚函数，则虚函数表中的基类函数会被覆盖

使用虚函数实现多态的原理

此时，类A的指针如果指向的是类 A 的对象，就会在类 A 的虚函数表中查出 A::f 的地址；如果指向的是类 B 的对象，就会在类B的虚函数表中查出B::f的地址。

虚析构函数

```
class B{
 public:
  ~B()
 { cout<<"调用基类B的析构函数\n";}
};
class D:public B{
 public:
  ~D()
 { cout<<"调用派生类D的析构函数\n";}
};
int main()
{ D obj;
  return 0;  }
```

程序运行的结果为:
调用派生类D的析构函数
调用基类B的析构函数

```
class B{
 public:
  ~B()
 { cout<<"调用基类B的析构函数\n";  }
};
class D:public B{
 public:
   ~D()
   { cout<<"调用派生类D的析构函数\n";}
};
int main()
{ B *p;          //定义指向基类B的指针变量p
  p= new D;
  delete p;
  return 0;
}
```

程序运行的结果为:
调用基类B的析构函数

本程序只执行了基类B的析构函数,而没有执行派生类D的析构函数。原因是当撤销指针p所指的派生类的无名对象, 调用析构函数时,采用了静态联编方式,只调用了基类B的析构函数。

```
class B{
 public:
  ~B()
 { cout<<"调用基类B的析构函数\n";  }
};
class D:public B{
 public:
   ~D()
   { cout<<"调用派生类D的析构函数\n";}
};
int main()
{
  B *p;          //定义指向基类B的指针变量p
  p= new D;
  delete p;
  return 0;
}
```

程序运行的结果为:
调用基类B的析构函数

```
#include<iostream>
using namespace std;
class B{
  public:
    virtual ~B()
    { cout<<"调用基类B的析构函数\n";}
};
class D:public B{
  public:
    ~D()
    { cout<<"调用派生类D的析构函数\n";}
};
int main()
{ B *p;          //定义指向基类B的指针变量p
  p= new D;
  delete p;
  return 0;
}
```

将基类的析构函数声明为虚析构函数

使用了虚析构函数，程序执行了动态联编，实现了运行的多态性。

程序运行的结果为：
调用派生类D的析构函数
调用基类B的析构函数

声明虚析构函数

声明虚析构函数的一般格式为:

```
virtual ~类名();
```

说明:

- 虚析构函数没有类型，也没有参数。
- 如果将基类的析构函数定义为虚函数，由该基类所派生的所有派生类的析构函数也都自动成为虚函数。
- 在C++中，不能声明虚构造函数，但是可以声明虚析构函数。

虚函数与函数重载的关系

在一个派生类中重新定义虚函数是函数重载的另一种形式，但它不同于一般的函数重载。

- 当普通函数重载时，其函数的参数个数或参数类型必须有所不同，函数的返回类型也可以不同。
- 当重载一个虚函数时，要求函数名、返回类型、参数个数、参数的类型和顺序与基类中的虚函数原型完全相同。

虚函数与函数重载的关系

当重载一个虚函数时, 如果仅仅返回类型不同，其余均相同，系统会给出错误信息。

```
class B{
 public:
   virtual void f( );
};
class D:public B {
 public:
   char f( );
};
```

虚函数与函数重载的关系

- 若仅仅函数名相同，而参数的个数、类型或顺序不同，系统将它作为普通的函数重载，这时将失去虚函数的特性。

```
class B{
 public:
  virtual void f( );
};
class D:public B{
 public:
  void f(int x);
};
```

虚函数与函数重载的关系

- 如果函数返回值是基类引用或指针，可以将派生类中的函数返回值修改为指向派生类的引用或指针，成为返回类型协变。

```
class B{
 public:
  virtual B& f( );
};
class D:public B{
 public:
  D& f();
};
```

多重继承与虚函数

- 多重继承可以视为多个单继承的组合。
- 因此，多重继承情况下的虚函数的调用与单继承情况下的虚函数的调用有相似之处。

```
#include <iostream>
using namespace std;
class B1{
  public:
     virtual void fun()      //定义fun()是虚函数
     {  cout<<"--B1--\n"; }
};
class B2{
   public:
     void fun()              //定义fun()为普通的成员函数
     {  cout<<"--B2--\n"; }
};
class D:public B1,public B2{
   public:
     void fun()
     {  cout<<"--D--\n"; }
};
```

```
int main()
{
    B1 obj1,*ptr1;
    B2 obj2,*ptr2;
    D obj3;
    ptr1=&obj1;
    ptr1->fun();
    ptr2=&obj2;
    ptr2->fun();
    ptr1=&obj3;
    ptr1->fun();
    ptr2=&obj3;
    ptr2->fun();
    return 0;
}
```

A2中的函数不是虚函数

运行结果:

```
--B1--
--B2--
--D--
--B2--
```

单选题 2分

设置

此题未设置答案，请点击右侧设置按钮

关于虚函数的说法**不正确**的是

A 基类中的虚函数可为派生类继承，继承下来仍是虚函数

B 虚函数重新定义时必须保证其返回值和参数个数及类型与基类中的相一致

C 虚函数必须是类的一个成员函数，不能是静态成员函数

D 析构函数和构造函数都可是虚函数

提交

3 纯虚函数和抽象类

```
class Figure {
   protected:
      double x,y;
   public:
       Figure(double a,double b){ x=a; y=b; }
       virtual double Area( )
       { cout<<"在基类中定义的虚函数\n"; }
};
```

在基类中，函数area()无法定义，即基类本身并不需要这个虚函数，但是基类要为派生类提供一个公共接口，以便派生类根据需要重新定义虚函数。

纯虚函数的定义格式

纯虚函数的一般形式如下：

virtual 函数类型 函数名(参数表)=0;

纯虚函数的定义格式

```
class Figure {
  protected:
    double x,y;
  public:
    Figure(double a,double b){ x=a; y=b;}
    virtual double Area( )=0;
};
```

纯虚函数的特点

纯虚函数的作用是在基类中为其派生类保留一个函数的名字,以便派生类根据需要对它进行重新定义。

纯虚函数的特点：

① 纯虚函数没有函数体；

② 最后面的“=0”并不表示函数的返回值为0，它只起形式上的作用，告诉编译系统“这是纯虚函数”；

③ 纯虚函数不具备函数的功能，不能被调用。

```
class Circle: public Figure                    //圆类
{
    protected:
        double r;
    public:
        Circle(double a, double b, double c): Figure(a,b)
        {r = c;}
        double Area()
        {       return PI * r * r;}
};
```

```
class Rectangle: public Figure                    //矩形类
{   protected:
        double l, w;
    public:
    Rectangle(double a, double b, double c, double d):
    Figure(a,b)
    {   l = c;
        w = d;
    }
    double Area()
    {   return c * d;        }
};
```

```
int main()
{
    Figure *pFigure;
    Circle circle1(100, 100, 10.0);
    Rectangle rectangle1(300, 300, 110.0, 90.0);
    pFigure = &circle1;
    cout<<"圆的面积为："<<pFigure->Area()<<endl;

    pFigure = &rectangle1;
    cout<<"矩形的面积为："<<pFigure->Area()<<endl;
    return 0;
}
```

程序的运行结果如下：
圆的面积为：314.12926
矩形的面积为：9900.0

抽象类

- 什么是抽象类?
- 如果一个类至少有一个纯虚函数，那么就称该类为抽象类。

```
class Figure{
 protected:
  double x,y;
 public:
  Figure(double a,double b){ x=a; y=b;}
  virtual void Area( )=0;
};
```

抽象类

抽象类的作用

- 抽象类的作用是作为一个类族的共同基类，相关的派生类是从这个基类派生出来的。

- 使用抽象类的几点规定

(1)由于抽象类中至少包含有一个没有定义功能的纯虚函数，因此抽象类只能用作其他类的基类，不能建立抽象类对象。

使用抽象类的几点规定

(2)在派生类中,如果对基类的纯虚函数没有重新定义,则该函数在派生类中仍是纯虚函数,该派生类仍为抽象类。

```
class B {
 public:
 virtual void Show()=0;
 . . .};
class D:public B {
 . . .
};
```

派生类D仍是抽象类。

使用抽象类的几点规定

(3) 抽象类不能用作参数类型、函数返回类型或显式转换的类型。

但可以声明指向抽象类的指针变量/引用,此指针/引用指向它的派生类,进而实现多态性。

单选题 2分

设置

此题未设置答案，请点击右侧设置按钮

关于抽象类**不正确的**说法是

A 至少有一个纯虚函数

B 抽象类中的纯虚函数没有定义，它不能定义对象

C 不可声明抽象类的指针

D 抽象类不能用作参数类型、函数返回类型及显式转换类型

提交