**例13.5 在类中，用字符数组实现字符串。**

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**class Student**

**{**

**char Num[10]; //学号，注意：用数组实现**

**char Name[10]; //姓名，注意：用数组实现**

**int Score; //成绩**

**public:**

**Student(char num[ ]=NULL, char name[ ]=NULL, int score=0)**

**{ if(num)strcpy(Num, num);**

**else Num[0]='\0';**

**if(name)strcpy(Name, name);**

**else Name[0]='\0';**

**Score=score;**

**}**

**void Show( )**

**{ cout<<"Num="<<Num<<'\t'**

**<<"Name="<<Name<<'\t'**

**<<"Score="<<Score<<endl;**

**}**

**};**

**int main( )**

**{ Student stud1("01201", "Mary", 88), stud2;**

**stud2=stud1; //A**

**stud1.Show( );**

**stud2.Show( );**

**return 0;**

**}**

**程序的运行结果是：**

**Num=01201 Name=Mary Score=88**

**Num=01201 Name=Mary Score=88**

**返回ppt讲稿**

**例13.7 += 运算符重载函数的返回值为void**

**#include <iostream>**

**using namespace std;**

**class Complex**

**{**

**double Real, Image;**

**public:**

**Complex(double r=0, double i=0)**

**{ Real=r; Image=i; }**

**void operator+=(const Complex &c)//A，返回值为void型**

**{**

**Real+=c.Real;**

**Image+=c.Image;**

**}**

**void Show( )**

**{**

**cout<<Real;**

**if(Image>0) cout<<'+'<<Image<<'i'; //如果是正数，必须输出正号**

**else if(Image<0) cout<<Image<<'i'; //如果是负数，自动输出负号**

**cout<<endl;**

**}**

**};**

**int main( )**

**{**

**Complex c1(2, 3), c2(4, -2);**

**cout<<"c1=";c1.Show( );**

**cout<<"c2=";c2.Show( );**

**c1+=c2; // B**

**cout<<"c1+=c2;c1=";c1.Show( );**

**return 0;**

**}**

**运行结果如下：**

**c1=2+3i**

**c2=4-2i**

**c1+=c2;c1=6+1i**

**程序中的B行被处理成 c1. operator+=(c2)，调用的是A行定义的函数。**

**如果有连续赋值的表达式，如c1+=c2+=c3，**

**编译器会处理成 c1. operator+=( c2. operator+=(c3) )，**

**注意划线部分是第一次赋值，表达式的结果是void类型，此时若欲做第二个赋值就不行了。**

**将上例中的operator+=( )重载函数改写成：**

**Complex operator+=(const Complex &c)**

**// 返回值为本类类型的对象**

**{**

**Real+=c.Real;**

**Image+=c.Image;**

**return \*this;**

**}**

**此时处理c1+=c2+=c3就没有问题了。**

**返回ppt讲稿**

**例13.8 对于字符串类，重载 = 运算符，返回对象自身的引用。就本例而言，可以不定义拷贝构造函数，程序能正确运行。**

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**class String**

**{ char \*Strp;**

**public:**

**String(char \*s=0) // 构造函数**

**{ if(s)**

**{ Strp=new char[strlen(s)+1];**

**strcpy(Strp, s);**

**}**

**else Strp=0;**

**}**

**String & operator = (String & s) // A赋值运算符重载函数**

**{ if(Strp) delete [ ]Strp;**

**if(s.Strp)**

**{ Strp=new char[strlen(s.Strp)+1];**

**strcpy(Strp, s.Strp);**

**}**

**else Strp=0;**

**return \* this;**

**}**

**~String( ) // 析构函数**

**{ if(Strp) delete [ ]Strp; }**

**void Show( )**

**{ cout<<Strp<<endl; }**

**};**

**int main( )**

**{ String str1("Sample String1");**

**String str2;**

**str2=str1; // 对象赋值运算**

**str2.Show( ); // 输出 Sample String1**

**return 0;**

**}**

**但是若将程序中A行的函数头部改为：**

**String operator = (String & s) ，**

**即返回本类对象，因类中有动态申请的字符串空间，则必须增加定义拷贝构造函数。因为要使用返回的对象去初始化内存临时对象，所以应在类体中增加定义拷贝构造函数如下：**

**String(String & s) // 拷贝构造函数**

**{ if(s.Strp)**

**{ Strp=new char[strlen(s.Strp)+1];**

**strcpy(Strp, s.Strp);**

**}**

**else Strp=0;**

**}**

**返回ppt讲稿**

**例13.11 成员函数和类型转换函数的比较**

**#include <iostream>**

**using namespace std;**

**class RMB //定义一个“人民币”类**

**{**

**int Yuan, Jiao, Fen; //元、角、分**

**public:**

**RMB(int y=0, int j=0, int f=0)**

**{ Yuan=y; Jiao=j; Fen=f; }**

**operator int( ) //转换成整数值(结果单位：分)**

**{ return (Yuan\*100+Jiao\*10+Fen); }**

**operator double( ) //转换成实数值(结果单位：元)**

**{ return (Yuan+double(Jiao)/10+double(Fen)/100); }**

**int GetFen( ) //用成员函数转换成“分”值**

**{ return (Yuan\*100+Jiao\*10+Fen); }**

**};**

**int main(void)**

**{ RMB r(23, 8, 6);**

**int r1, r2, r3;**

**r1=r ; //A 处理成 r1 = int(r);本质与B行一致**

**r2=int(r); //B 处理成 r2 = r.operator int( );**

**r3=r.GetFen( ); //C 调用成员函数得到分值**

**cout<<"r1="<<r1<<endl;**

**cout<<"r2="<<r2<<endl;**

**cout<<"r3="<<r3<<endl;**

**double d;**

**d = r; //D 处理成 d = r.operator double( );**

**cout<<"d="<<d<<endl;**

**return 0;**

**}**

**程序的运行结果如下：**

**r1=2386**

**r2=2386**

**r3=2386**

**d=23.86**

**返回ppt讲稿**

**假定一种货币只有元和角两种单位，一元等于10角，++和 -- 表示加减1角。定义一个类Money 用于实现++和 -- 运算。**

**例13.12 实现++和--的前置和后置运算符重载**

**#include <iostream> ++ - - 表示家减1角**

**using namespace std;**

**class Money**

**{**

**int Yuan, Jiao; //元、角**

**public:**

**Money(int y=0, int j=0)**

**{ Yuan=y; Jiao=j; }**

**Money(double d)**

**{ Yuan=int(d); Jiao=int((d-Yuan)\*10); }**

**Money operator++( ); //①用成员函数实现前置++**

**Money operator++(int); //②用成员函数实现后置++**

**friend Money operator--(Money &m);**

**//③用友元函数实现前置- -**

**friend Money operator--(Money &m, int);**

**//④用友元函数实现后置- -**

**void Show(char \*s)**

**{**

**cout<<s<<':'<<Yuan<<","<<Jiao<<endl;**

**}**

**};**

**Money Money::operator++( )**

**//①用成员函数实现前置++**

**{**

**Jiao++;**

**调用方式：**

**m1 = ++m;**

**解释为m.operator++( )**

**if(Jiao>=10)**

**{**

**Yuan+=1;**

**Jiao-=10;**

**}**

**return \*this;**

**}**

**Money Money::operator++(int)**

**//②用成员函数实现后置++**

**{**

**Money temp = \*this; //保存对象原始的值**

**Jiao++;**

**调用方式：**

**m2 = m++;**

**解释为m.operator++(int )**

**if(Jiao>=10)**

**{**

**Yuan+=1;**

**Jiao-=10;**

**}**

**return temp; //返回对象原始的值**

**}**

**Money operator--(Money &m)**

**//③用友元函数实现前置--**

**{**

**if(m.Jiao==0)**

**{**

**m.Yuan -= 1;**

**m.Jiao=10;**

**调用方式：**

**m3 = --m;**

**解释operator--(m)**

**}**

**m.Jiao--;**

**return m;**

**}**

**Money operator--(Money &m, int)**

**//④用友元函数实现后置--**

**{**

**Money temp=m; //保存对象原始的值**

**if(m.Jiao==0)**

**{**

**调用方式：**

**m4 = m--;**

**解释为operator-- (m, int )**

**m.Yuan-=1;**

**m.Jiao=10;**

**}**

**m.Jiao--;**

**return temp; //返回对象原始的值**

**}**

**int main(void)**

**{**

**Money m(15, 3), m1, m2, m3, m4;**

**cout<<"m: "; m.Show("m");**

**m1 = ++m;**

**cout<<"m1=++m;\n";m1.Show("m1"); m.Show("m");**

**m2=m++;**

**cout<<"m2=m++;\n";m2.Show("m2"); m.Show("m");**

**m3=--m;**

**cout<<"m3=--m;\n"; m3.Show("m3"); m.Show("m");**

**m4=m--;**

**cout<<"m4=m--;\n"; m4.Show("m4"); m.Show("m");**

**Money m5(8.5);**

**m5.Show("m5");**

**return 0;**

**}**

**返回ppt讲稿**

**例13.18 定义字符串类String，并测试重载的运算符以及成员函数**

**//Li1318.h**

**class String**

**{**

**protected:**

**int Length; //字符串的长度**

**char \*Strp; //指向字符串的指针**

**public:**

**String( ){ Strp=NULL; Length=0; } //缺省构造函数**

**String(const char \*s); //构造函数,以字符指针作为参数**

**String(const String &);**

**//拷贝构造函数,以字符串对象的引用作为参数**

**~String( )**

**?**

**{ if(Strp) delete [ ] Strp; }**

**const char \*IsIn( const char ) const;**

**//返回参数字符在字符串中第一次出现的地址**

**bool IsSubStr(const char \*) const;**

**//判断参数字符串是否为子串**

**void Show( ) //输出字符串**

**{ cout << Strp << '\n'; }**

**int GetLen( ) { return Length; } //返回字符串的长度**

**const char \* GetString( ) { return Strp; }**

**//取字符串首指针**

**operator const char \* ( ) const**

**//A，类型转换函数，返回指向常量的字符串指针**

**{ return (const char \*) Strp; }**

**String & operator = (String &); //重载赋值运算符**

**friend String operator+(const String &, const String & );**

**//友元实现+重载**

**friend String operator-(const String &, const char \* );**

**//友元实现－重载**

**int operator < ( const String &) const;**

**int operator > ( const String &) const;**

**int operator == ( const String &) const;**

**};**

**String::String(const char \*s)**

**//构造函数，以字符指针作为参数**

**{ if(s)**

**{ Length = strlen(s);**

**Strp = new char[Length+1];**

**strcpy(Strp, s);**

**}**

**else**

**{ Strp = NULL; Length = 0; }**

**}**

**String::String(const String &s) //拷贝构造函数，以字符串对象的引用作为参数**

**{ Length = s.Length;**

**需要删除原空间吗？**

**与赋值运算符对照？**

**if(s.Strp)**

**{ Strp = new char[Length+1];**

**strcpy(Strp, s.Strp);**

**}**

**else Strp = NULL;**

**}**

**const char \*String::IsIn( const char c) const // 判断字符c是否在字符串中, 如果在,返回其第一次出现时的地址, 否则返回空指针NULL**

**{ char \*p=Strp;**

**while(\*p)**

**if(\*p++ == c) return --p;**

**return NULL;**

**}**

**bool String::IsSubStr(const char \*s) const //判断s所指向的字符串是否为类中字符串的子串。若是，返回true；若不是，返回false。**

**{**

**if(strstr(Strp, s)) return true;**

**// strstr( ) 为字符串处理库函数，返回子串首地址**

**else return false;**

**}**

**String & String::operator = (String &s) //实现赋值运算**

**{**

**if( this == & s ) return \*this; //处理字符串自身赋值**

**if(Strp) delete [ ] Strp; //释放对象自身的空间**

**Length = s.Length;**

**if(s.Strp)**

**{ Strp = new char[Length+1];**

**strcpy(Strp, s.Strp);**

**}**

**else Strp = NULL;**

**return \*this;**

**}**

**String operator+(const String &s1, const String &s2 )**

**//拼接两个字符串**

**{ String t;**

**t.Length = s1.Length + s2.Length;**

**t.Strp = new char [ t.Length + 1 ];**

**strcpy(t.Strp, s1.Strp);**

**strcat(t.Strp, s2.Strp);**

**return t;**

**}**

**String operator-(const String &s1, const char \*s2 ) // D 删除s1中第一次出现的s2**

**{ String t;**

**char \*p1=s1.Strp, \*p2;**

**int i=0, len=strlen(s2);**

**if(p2=strstr(s1.Strp, s2)) // 如果是子串**

**{ t.Length=s1.Length - len;**

**t.Strp = new char[t.Length+1];**

**while(p1<p2)**

**t.Strp[i++]=\*p1++;**

**p1+=len;**

**while(t.Strp[i++]=\*p1++);**

**return t;**

**}**

**else**

**return s1 ;**

**}**

**int String::operator < ( const String &s) const // 实现小于比较，成员函数实现**

**{ return(strcmp(Strp, s.Strp)<0 ); }**

**int String::operator > ( const String &s) const // 实现大于比较，成员函数实现**

**{ return(strcmp(Strp, s.Strp)>0 ); }**

**int String::operator == ( const String &s) const // 实现恒等于比较，成员函数实现**

**{ return(strcmp(Strp, s.Strp)==0 ); }**

**//Li1318.cpp**

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**#include "Li1318.h"**

**void Test\_IsIn(char c, String &s)**

**{**

**cout<<"\'"<<c<<"\' in "; s.Show( );**

**if(s.IsIn(c)) //测试类的成员函数IsIn( )**

**{ // s.IsIn(c)返回的是c第一次在s中出现的地址**

**cout <<"Yes:"<<s.IsIn(c)<<endl;**

**}**

**else cout <<"No!\n";**

**}**

**void Test\_IsSubStr(char \*sp, String &s)**

**{ cout<<"\""<<sp<<"\" in "; s.Show( );**

**if(s.IsSubStr(sp)) // 测试类的成员函数IsSubStr( )**

**cout <<"Yes!\n";**

**else cout <<"No!\n";**

**}**

**int main(void)**

**{ String s1("C++程序设计 "), s2, s3("学生学习 ");**

**String s, s5;**

**s1.Show( );**

**s2=s1; // 测试赋值运算符“=”**

**s2.Show( );**

**s=s3+s2; // 测试字符串拼接运算符“+”以及赋值运算符“=”**

**s.Show( );**

**s5=s-s1; // 测试字符串相减运算符“－”**

**s5.Show( );**

**String s6="C++ programming! ";**

**Test\_IsIn('g', s6); //判断一个字符是否在“串”中**

**Test\_IsIn('k', s6);**

**Test\_IsSubStr("prog", s6); //判断字符串是否是“串”的子串**

**Test\_IsSubStr("red", s6);**

**cout<<"s1=";s1.Show( );**

**cout<<"s3=";s3.Show( );**

**if(s1>s3) //字符串比较**

**cout<<"s1>s3\n";**

**else if(s1<s3) //字符串比较**

**cout<<"s1<s3\n";**

**else if(s1==s3) //字符串比较**

**cout<<"s1==s3\n";**

**const char \*sp1, \*sp3;**

**sp1=s1; sp3=s3; //调用A行的类型转换函数，由String类转换成const char \*类**

**cout<<sp3<<sp1<<endl;**

**cout<<s3.GetString( )<<endl;**

**return 0;**

**}**

**程序的运行结果如下：**

**C++程序设计**

**C++程序设计**

**学生学习 C++程序设计**

**学生学习**

**'g' in C++ programming!**

**Yes:gramming!**

**'k' in C++ programming!**

**No!**

**"prog" in C++ programming!**

**Yes!**

**"red" in C++ programming!**

**No!**

**s1=C++程序设计**

**s3=学生学习**

**s1<s3**

**学生学习 C++程序设计**

**学生学习**

**返回ppt讲稿**

**例13.21 类的成员函数的静态联编**

**#include <iostream>**

**using namespace std;**

**class Point**

**{**

**protected:**

**double x, y; //点的坐标值**

**public:**

**Point(double a=0, double b=0)**

**{ x=a; y=b; }**

**double Area( ) //函数1**

**{ return 0.0; }**

**};**

**class Rectangle : public Point**

**{**

**protected:**

**double x1, y1; //长方形右下角点的坐标值，**

**基类中的x, y为左上角坐标点**

**public:**

**Rectangle(double a=0, double b=0, double c=0, double d=0):Point(a, b)**

**{ x1=c; y1=d; }**

**double Area( ) //函数2**

**{ return (x-x1)\*(y-y1); }**

**};**

**class Circle: public Point**

**{**

**protected:**

**double r; //半径，基类中x, y为圆心坐标点**

**public:**

**Circle(double a=0, double b=0, double c=0):Point(a, b)**

**{ r=c; }**

**double Area( ) //函数3**

**{ return 3.14\*r\*r; }**

**};**

**double CalcArea(Point &p) //普通函数**

**{**

**return(p.Area( )); //A 编译连接时确定调用函数1**

**}**

**int main( )**

**{**

**Rectangle r(0, 0, 1, 1);**

**Circle c(0, 0, 1);**

**cout<<CalcArea(r)<<'\t'; 参数传递：Point &p＝r**

**cout<<CalcArea(c)<<'\n'; 参数传递：Point &p＝c**

**return 0;**

**}**

**程序运行输出：0 0**

**返回ppt讲稿**

**例13.22 将成员函数定义成虚函数，以实现动态联编**

**#include <iostream>**

**using namespace std;**

**class Point**

**{**

**protected:**

**double x, y; //点的坐标值**

**public:**

**Point(double a=0, double b=0)**

**{ x=a; y=b; }**

**virtual double Area( ) //虚函数1**

**{ return 0.0; }**

**};**

**class Rectangle:public Point**

**{**

**protected:**

**double x1, y1;**

**//长方形右下角点的坐标值，基类中x, y为左上角坐标点**

**public:**

**Rectangle(double a=0, double b=0, double c=0, double d=0):Point(a,b)**

**{ x1=c; y1=d; }**

**virtual double Area( ) //虚函数2**

**{ return (x-x1)\*(y-y1); }**

**};**

**class Circle:public Point**

**{ protected:**

**double r; //半径，基类中x, y为圆心坐标点**

**public:**

**Circle(double a=0, double b=0, double c=0):Point(a, b)**

**{ r=c; }**

**virtual double Area( ) //虚函数3**

**{ return 3.14\*r\*r; }**

**};**

**double CalcArea(Point &p)//基类对象引用派生类对象**

**{ return(p.Area( )); } //基类的对象调用派生类虚函数**

**int main()**

**{**

**Point p(1, 2);**

**Rectangle r(0, 0, 1, 1);**

**Circle c(0, 0, 1); 基类对象引用派生类对象**

**cout<<CalcArea(p)<<'\t' 参数传递：Point &p＝p**

**<<CalcArea(r)<<'\t' 参数传递：Point &p＝r**

**<<CalcArea(c)<<'\n'; 参数传递：Point &p＝c**

**return 0;**

**}**

**或者将最后的两个函数改为：**

**double CalcArea(Point \*p) //基类的指针指向派生类对象**

**{ return(p->Area( )); } //基类的指针调用派生类虚函数**

**int main( )**

**{**

**Point p(1, 2);**

**Rectangle r(0, 0, 1, 1);**

**Circle c(0, 0, 1); 基类指针指向派生类对象**

**cout<<CalcArea(&p)<<'\t' 参数传递：Point \*p＝&p**

**<<CalcArea(&r)<<'\t' 参数传递：Point \*p＝&r**

**<<CalcArea(&c)<<'\n'; 参数传递：Point \*p＝&c**

**return 0;**

**}**

**注意：只能通过基类的对象引用派生类对象，**

**或通过基类的指针指向派生类对象实现动态多态。**

**返回ppt讲稿**

**例13.23 成员函数调用虚函数**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**public:**

**virtual void fun1( ) // D 虚函数**

**{ cout << "A::fun1" << '\t';**

**fun2( );**

**}**

**void fun2( )**

**{ cout << "A::fun2" << '\t';**

**fun3( ); // E**

**}**

**virtual void fun3( ) // F 虚函数**

**{ cout << "A::fun3" << '\t';**

**fun4( );**

**}**

**void fun4( )**

**{ cout << "A::fun4" << '\n'; }**

**};**

**class B: public A**

**{**

**public:**

**void fun3( )**

**{ cout << "B::fun3" << '\t';**

**fun4( );**

**}**

**void fun4( )**

**{ cout << "B::fun4" << '\n'; }**

**};**

**int main( )**

**{ A a;**

**a.fun1( );**

**B b;**

**b.fun1( ); // G**

**return 0;**

**}**

**程序的运行结果是：**

**A::fun1 A::fun2 A::fun3 A::fun4**

**A::fun1 A::fun2 B::fun3 B::fun4**

**返回ppt讲稿**

**例13.26 虚析构函数**

**Base**

**A**

**B**

**#include <iostream>**

**#include <cstring>**

**using namespace std;**

**class base**

**{**

**char \*baseptr;**

**public:**

**base( )**

**{ baseptr=new char[100];**

**strcpy(baseptr, "In class base");**

**fc( );**

**}**

**virtual void fc( ) // D**

**{ cout<<baseptr<<endl; }**

**virtual ~base( ) // E**

**{ delete [ ]baseptr;**

**cout<<"Delete [ ]baseptr"<<endl;**

**}**

**};**

**class A : public base**

**{**

**char \*Aptr;**

**public:**

**A( )**

**{ Aptr=new char[100];**

**strcpy(Aptr, "In class A");**

**fc( ); // F**

**}**

**void f( ) { fc( ); } // G**

**~A( )**

**{ delete [ ]Aptr;**

**cout<<"Delete [ ]Aptr"<<endl;**

**}**

**};**

**class B : public A**

**{**

**char \*Bptr;**

**public:**

**B( )**

**{ Bptr=new char[100];**

**strcpy(Bptr, "In class B");**

**}**

**void fc( ) { cout<<Bptr<< endl; }**

**~B( )**

**{ delete [ ]Bptr;**

**cout<<"Delete [ ]Bptr"<<endl;**

**}**

**};**

**int main(void)**

**{ B b; // H**

**b.f( ); // K**

**base \*p=new B; // M**

**delete p; // N**

**return 0;**

**}**

**程序的输出结果是：**

**In class base**

**In class base**

**In class B**

**In class base**

**In class base**

**Delete [ ]Bptr**

**Delete [ ]Aptr**

**Delete [ ]baseptr**

**Delete [ ]Bptr**

**Delete [ ]Aptr**

**Delete [ ]baseptr**

**返回ppt讲稿**

**例13.27 定义抽象类，派生出若干类，在派生类中实现纯虚函数**

**#include <iostream>**

**using namespace std;**

**class Shape**

**{**

**public:**

**virtual double Area( )=0; //抽象的求面积的纯虚函数**

**virtual void Draw( )=0; //抽象的绘制图形的纯虚函数**

**};**

**class Point:public Shape**

**{**

**protected:**

**double x, y; //点的坐标值**

**public:**

**Point(double a=0, double b=0)**

**{ x=a; y=b; }**

**double Area( ) //基类纯虚函数Area( )的具体实现**

**{ return 0.0; }**

**void Draw( ) //基类纯虚函数Draw( )的具体实现**

**{ cout <<"Draw Point!\n"; }**

**};**

**class Rectangle:public Point**

**{**

**protected:**

**double x1, y1;//长方形右下角点的坐标值，基类中x, y为左上角坐标点**

**public:**

**Rectangle(double a=0, double b=0, double c=0, double d=0):Point(a, b)**

**{ x1=c; y1=d; }**

**double Area( ) //基类纯虚函数Area( )的具体实现**

**{ return (x-x1)\*(y-y1); }**

**void Draw( ) //基类纯虚函数Draw( )的具体实现**

**{ cout <<"Draw Rectangle!\n"; }**

**};**

**class Circle:public Point**

**{**

**protected:**

**double r; //半径，基类中x, y为圆心坐标点**

**public:**

**Circle(double a=0, double b=0, double c=0):Point(a, b)**

**{ r=c; }**

**double Area( ) //基类纯虚函数Area( )的具体实现**

**{ return 3.14\*r\*r; }**

**void Draw( ) //基类纯虚函数Draw( )的具体实现**

**{ cout <<"Draw Circle!\n"; }**

**};**

**double CalcArea(Shape &s)**

**{ return(s.Area( )); } //A通过基类对象的引用实现动态多态**

**void DrawShape(Shape \*sp)**

**{ sp->Draw( ); } //B通过基类对象的指针实现动态多态**

**int main( )**

**{ Point p(1, 2);**

**Rectangle r(0, 0, 1, 1);**

**Circle c(0, 0, 1);**

**cout<<CalcArea(p)<<'\t'<<CalcArea(r)<<'\t'**

**<<CalcArea(c)<<'\n';**

**DrawShape(&p);**

**DrawShape(&r);**

**DrawShape(&c);**

**return 0;**

**}**

**程序的运行结果是：**

**0 1 3.14**

**Draw Point!**

**Draw Rectangle!**

**Draw Circle!**

**返回ppt讲稿**

**例13.28 给出纯虚函数的函数体。**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**public:**

**virtual void fun( )=0 { cout<<"virtual fun=0"<<endl; }**

**};**

**class B: public A**

**{**

**public:**

**virtual void fun( ) { cout<<"virtual fun"<<endl; }**

**};**

**int main( )**

**{**

**B b;**

**b.fun( );**

**b.A::fun( ); // 调用纯虚函数的函数体**

**return 0;**

**}**

**程序的运行结果是：**

**virtual fun**

**virtual fun=0**

**注意，即使给出了纯虚函数的函数体，该函数依然是纯虚函数，含有该纯虚函数的类依然是抽象类，即在本例中不能定义A类的对象。**

**返回ppt讲稿**