**例12.1 先定义“点”类Point，**

**再由“点”类派生出“圆”类Circle。**

**#include <iostream>**

**using namespace std;**

**#define PI 3.14159**

**class Point // 定义“点”类**

**{**

**int x, y;**

**public:**

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

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

**void ShowPoint( )**

**{ cout<<"Point:("<<x<<','<<y<<")\n"; }**

**int Getx( )**

**{ return x; }**

**int Gety( )**

**{ return y; }**

**void Setxy(int a, int b)**

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

**};**

**class Circle: public Point // 定义“圆”类，公有继承**

**{**

**int r; // “圆”的半径**

**public:**

**Circle(int x, int y, int ra) : Point(x, y) // B**

**{ r = ra; }**

**void Setr(int ra)**

**{ r = ra; }**

**double Area( ) //求圆的面积**

**{ return PI\*r\*r; }**

**void Move(int x\_offset, int y\_offset) //将圆心坐标平移**

**{**

**int x1=Getx( ); //存取基类的私有成员**

**int y1=Gety( ); // D**

**x1 += x\_offset; y1 += y\_offset;**

**Setxy(x1, y1); // E**

**}**

**void ShowCircle( )**

**{**

**ShowPoint( ); // F**

**cout<<" Radius: "<<r<<'\t';**

**cout<<"Area: "<<Area( )<<endl; //G**

**}**

**};**

**int main()**

**{**

**Circle c(0, 0, 2); // 0,0表示圆心坐标的初值，**

**2表示半径的初值**

**c.ShowCircle();**

**c.Move(2, 2);**

**c.ShowCircle();**

**c.Setxy(0, 0); // 重新设置圆心坐标**

**c.Setr(1); // 重新置半径值**

**c.ShowCircle();**

**return 0;**

**}**

**返回ppt讲稿**

**例12.2 先定义“点”类Point和“半径”类Radius，再由Point类和Radius类多重派生出“圆”类Circle。**

**#include <iostream>**

**using namespace std;**

**#define PI 3.14159**

**class Point**

**{**

**protected: //A**

**int x, y;**

**public:**

**Point(int a=0, int b=0){ x=a; y=b; }**

**void ShowPoint( )**

**{ cout<<"Point:("<<x<<','<<y<<")\n";}**

**int Getx( ) { return x; }**

**int Gety( ) { return y; }**

**void Setxy(int a, int b){ x=a; y=b; }**

**};**

**class Radius**

**{**

**protected: //B**

**int r;**

**public:**

**Radius(int ra=0){ r = ra; }**

**void Setr(int ra){ r = ra; }**

**int Getr( ) { return r; }**

**};**

**class Circle : public Point, public Radius**

**{**

**public:**

**Circle(int x, int y, int ra) : Point(x, y), Radius(ra) //D**

**{ }**

**double Area( )**

**{ return PI\*r\*r; } //直接访问基类的保护成员**

**void Move(int x\_offset, int y\_offset)**

**{ x += x\_offset; y += y\_offset; }**

**void ShowCircle( )**

**{**

**ShowPoint( );**

**cout<<"Radius: "<<r<<'\t';**

**cout<<"Area: "<<Area( )<<endl;**

**}**

**};**

**int main( )**

**{**

**Circle c(0, 0, 2);**

**c.ShowCircle( );**

**c.Move(2, 2);**

**c.ShowCircle( );**

**c.Setxy(0, 0);**

**c.Setr(1);**

**c.ShowCircle( );**

**return 0;**

**}**

**程序的运行结果为：**

**Point:(0,0)**

**Radius: 2 Area: 12.5664**

**Point:(2,2)**

**Radius: 2 Area: 12.5664**

**Point:(0,0)**

**Radius: 1 Area: 3.14159**

**返回ppt讲稿**

**例12.3 多重继承中基类构造函数和析构函数的调用顺序**

**#include <iostream>**

**using namespace std;**

**class Base1**

**{**

**protected:**

**int data1;**

**public:**

**Base1(int a=0)**

**{ data1 = a;**

**cout<<"Base1 Constructor\n";**

**}**

**~Base1( )**

**{ cout<<"Base1 Destructor\n"; }**

**};**

**class Base2**

**{**

**protected:**

**int data2;**

**public:**

**Base2(int a=0)**

**{ data2 = a;**

**cout<<"Base2 Constructor\n";**

**}**

**~Base2( )**

**{ cout<<"Base2 Destructor\n"; }**

**};**

**class Derived: public Base1, public Base2 //A**

**{**

**int d;**

**public:**

**Derived(int x, int y, int z):Base1(x), Base2(y) //B**

**{ d=z; cout<<"Derived Constructor\n"; }**

**~Derived( )**

**{ cout<<"Derived Destructor\n"; }**

**void Show( )**

**{ cout<<data1<<','<<data2<<','<<d<<endl; }**

**};**

**int main( )**

**{**

**Derived c(1, 2, 3);**

**c.Show( );**

**return 0;**

**}**

**程序的运行结果是：**

**Base1 Constructor**

**Base2 Constructor**

**Derived Constructor**

**1, 2, 3**

**Derived Destructor**

**Base2 Destructor**

**Base1 Destructor**

**返回ppt讲稿**

**例12.4 对象成员构造函数和析构函数的调用顺序**

**#include <iostream>**

**using namespace std;**

**class Base1**

**{**

**protected:**

**int data1;**

**public:**

**Base1(int a=8)**

**{ data1 = a;**

**cout<<data1<<", Base1 Constructor\n";**

**}**

**~Base1( )**

**{ cout<<data1<<", Base1 Destructor\n"; }**

**};**

**class Base2**

**{**

**protected:**

**int data2;**

**public:**

**Base2(int a=9)**

**{ data2 = a;**

**cout<<data2<<", Base2 Constructor\n";**

**}**

**~Base2( )**

**{ cout<<data2<<", Base2 Destructor\n"; }**

**};**

**class Derived:public Base1, public Base2 //A**

**{ int d;**

**Base1 b1, b2; //B**

**public:**

**Derived(int x, int y, int z) : Base1(x), Base2(y), b1(x+y), b2(x+z) //C**

**{ d=z; cout<<"Derived Constructor\n"; }**

**~Derived( )**

**{ cout<<"Derived Destructor\n"; }**

**void Show( )**

**{ cout<<data1<<','<<data2<<','<<d<<endl; }**

**};**

**int main( )**

**{ Derived c(1, 2, 3);**

**c.Show( );**

**return 0;**

**}**

**程序的运行结果是：**

**1, Base1 Constructor**

**2, Base2 Constructor**

**3, Base1 Constructor //构造对象成员b1时的输出**

**4, Base1 Constructor //构造对象成员b2时的输出**

**Derived Constructor**

**1, 2, 3**

**Derived Destructor**

**4, Base1 Destructor //析构对象成员b2时的输出**

**3, Base1 Destructor //析构对象成员b1时的输出**

**2, Base2 Destructor**

**1, Base1 Destructor**

**返回ppt讲稿**

**例12.7 多层继承中的冲突，注意本例的继承关系如右下图所示：**

**#include <iostream>**

C{ y, SetAx( ),SetBx( ),

Sety( ),Gety( ) }

A{ x, Show( ) }

B{ x, Show( ) }

D{ z, Setz( ),Getz( ) }

**using namespace std;**

**class A**

**{**

**protected:**

**int x;**

**public:**

**void Show( )**

**{ cout << "x=" << x << '\n' ; }**

**};**

**class B**

**{**

**protected:**

**int x;**

**public:**

**void Show( )**

**{ cout << "x=" << x << '\n' ; }**

**};**

**class C: public A, public B //公有继承 A、B 类**

**{**

**int y;**

**public:**

**void SetAx(int a) { A::x=a; }**

**void SetBx(int a) { B::x=a; }**

**void Sety(int b) { y=b; }**

**int Gety( ) { return y; }**

**};**

**class D: public C //公有继承 C 类**

**{**

**int z;**

**public:**

**void Setz(int a) { z=a; }**

**int Getz( ) { return z; }**

**};**

**int main(void)**

**{ D d;**

**d.SetAx(10); d.SetBx(20); d.Sety(30); d.Setz(40);**

**cout<<"A";**

**d.C::A::Show( ); //E 报错**

**cout<<"B";**

**d.C::B::Show( ); //F 报错**

**cout << "y=" << d.Gety( ) << '\n';**

**cout << "z=" << d.Getz( ) << '\n';**

**return 0;**

**}**

**解决1：在C类中增加成员函数：**

**void ShowA( ) {cout << "x=" << A::x << '\n'; }**

**void ShowB( ) {cout << "x=" << B::x << '\n'; }**

**再将E行和F行改写成：d.ShowA( ); 和d.ShowB( ); 即可。**

**解决2：把E行和F行改写成：d.A::Show( ); 和d.B::Show( ); 即可。**

**返回ppt讲稿**

**例12.8 支配规则示例**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**protected:**

**int x;**

**public:**

**void Set(int a)**

**{ x=a; }**

**void Show( )**

**{ cout << "x=" << x << '\n' ; }**

**};**

**class B : public A**

**{**

**protected: 问题：类B类对象有几个数据成员。**

**int x;**

**public:**

**void SetAx(int a)**

**{ A::x = a; } //访问的是基类A的x**

**void SetBx(int a)**

**{ x = a; } //访问的是派生类B的x**

**void Show( )**

**{ cout<<"x="<< x <<endl; }**

**};**

**int main(void)**

**{**

**B b;**

**b.SetAx(1);**

**b.SetBx(2);**

**b.A::Show( ); //访问的是基类A的Show( )**

**b.Show( ); //访问的是派生类B的Show( )**

**return 0;**

**}**

**返回ppt讲稿**

**例12.11 虚基类和非虚基类构造函数的调用。**

**#include <iostream>**

**using namespace std;**

**class A**

**{**

**public:**

**A( ) {cout<<"A";}**

**~A( ) {cout<<"~A";}**

**};**

**class B**

**{**

**public:**

**B( ) {cout<<"B";}**

**~B( ) {cout<<"~B";}**

**};**

**class C**

**{**

**public:**

**C( ) {cout<<"C";}**

**~C( ) {cout<<"~C";}**

**};**

**class D**

**{**

**public:**

**D( ) {cout<<"D";}**

**~D( ) {cout<<"~D";}**

**};**

**class E: virtual public B, public A, public D, virtual public C**

**{**

**};**

**int main( )**

**{**

**E c;**

**return 0;**

**}**

**程序的运行结果是：**

**BCAD~D~A~C~B**

**返回ppt讲稿**

**例12.12 访问对象成员的成员**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**class Point**

**{**

**int x, y;**

**public:**

**Point(int a=0, int b=0) { x=a; y=b;}**

**void Setx(int a){x=a;}**

**void Sety(int a){y=a;}**

**int Getx( ){ return x;}**

**int Gety( ){ return y;}**

**void Show( )**

**{ cout<<"point("<<x<<','<<y<<")\n"; }**

**};**

**class Line**

**{**

**Point p1, p2; //对象成员**

**public:**

**Line(int x1, int y1, int x2, int y2): p1(x1, y1), p2(x2, y2)**

**//调用对象成员构造函数**

**{ }**

**double Length( )**

**{ int x1, y1, x2, y2;**

**x1=p1.Getx( ); y1=p1.Gety( );**

**//访问对象成员p1的成员**

**x2=p2.Getx( ); y2=p2.Gety( );**

**//访问对象成员p2的成员**

**return sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));**

**}**

**void Show( )**

**{ p1.Show( ); //访问对象成员p1的成员**

**p2.Show( ); //访问对象成员p2的成员**

**cout<<"Length="<<Length( )<<endl;**

**}**

**};**

**int main( )**

**{ Line line(0, 0, 1, 1);**

**line.Show( );**

**return 0;**

**}**

**本程序的运行结果是：**

**point(0, 0)**

**point(1, 1)**

**Length=1.41421**

**注意在本程序中，Point类和Line类不是继承关系，只是Point类的两个对象，是Line类的对象成员，访问对象成员的成员，与访问一般对象的成员遵循同样的规则。**

**返回ppt讲稿**

**例12.13 访问基类成员**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**class Point**

**{**

**protected:**

**int x, y; //定义x、y为保护成员，以使在公有派生类中可直接访问它们**

**public:**

**Point(int a=0, int b=0) { x=a; y=b;}**

**void Setx(int a){x=a;}**

**void Sety(int a){y=a;}**

**int Getx( ){ return x; }**

**int Gety( ){ return y; }**

**void Show( )**

**{ cout<<"point("<<x<<','<<y<<")\n"; }**

**};**

**class Line : public Point //公有继承**

**{**

**protected :**

**int x1, y1;**

**public:**

**Line(int a, int b, int c, int d) : Point(a, b)**

**//调用基类构造函数**

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

**double Length( )**

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

**// A 直接访问基类保护成员x、y**

**void Show( )**

**{**

**Point::Show( ); //访问基类成员函数Show( )**

**cout<<"point("<<x1<<','<<y1 <<")\n";**

**cout<<"Length="<<Length( )<<endl;**

**}**

**};**

**int main( )**

**{**

**Line line(0, 0, 1, 1);**

**line.Show( );**

**return 0;**

**}**

**返回ppt讲稿**

**例12.14 赋值兼容规则**

**#include <iostream>**

**using namespace std;**

**class Point**

**{**

**protected:**

**int x, y; // 保护成员**

**public:**

**Point(int a=0, int b=0) { x=a; y=b;}**

**void Show( )**

**{ cout<<"point("<<x<<','<<y<<")\n"; }**

**};**

**class Line : public Point // 公有继承**

**{**

**protected:**

**int x1, y1;**

**public:**

**Line(int a, int b, int c, int d): Point(a, b) //调用基类构造函数**

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

**};**

**int main( )**

**{ Line line(2, 2, 6, 6);**

**Point p;**

**p = line; // A**

**p.Show( );**

**return 0;**

**}**

**程序运行结果是： Point(2, 2)**

**返回ppt讲稿**