EX1

(1)

…BaseClass1 Object is created! //对象a的BaseClass1类构造函数

…BaseClass1 Object is created! //对象b的BaseClass1基类类构造函数

…First layer derived Object is created! //对象b的Myderivrd1派生类构造函数

…BaseClass1 Object is created! //对象c的BaseClass1基类的基类构造函数

…First layer derived Object is created! //对象c的Myderivrd1基类构造函数

…Second layer derived Object is created! //对象c的Myderivrd11派生类构造函数

…Second layer derived Object is destroyed! //对象c的Myderivrd11派生类析构函数

…First layer derived Object is Destroyed! //对象c的Myderivrd1基类析构函数

…BaseClass1 Object is destroyed! //对象c的BaseClass1基类的基类析构函数

…First layer derived Object is Destroyed! //对象b的Myderivrd1派生类析构函数

…BaseClass1 Object is destroyed! //对象b的BaseClass1基类析构函数

…BaseClass1 Object is destroyed! //对象a的BaseClass1类析构函数

(2)

…BaseClass1 Object is created! //对象a的BaseClass2类private成员a1的构造函数

…BaseClass2 Object is created! //对象a的BaseClass2类构造函数

…BaseClass1 Object is created! //对象b的BaseClass2基类private成员的构造函数

…BaseClass2 Object is created! //对象b的BaseClass2基类构造函数

…BaseClass1 Object is created! //对象b的Myderived2派生类中private成员a1的构造函数

…First layer derived Object is created! //对象b的Myderived2派生类的构造函数

…BaseClass1 Object is created! //对象c的BaseClass2基类的基类private成员的构造函数

…BaseClass2 Object is created! //对象c的BaseClass2基类的基类构造函数

…BaseClass1 Object is created! //对象c的Myderived2基类类中private成员a1的构造函数

…First layer derived Object is created! //对象c的Myderived2基类的构造函数

…Second layer derived Object is created! //对象c的Myderived21派生类的构造函数

…Second layer derived Object is destroyed! //对象c的Myderived21派生类的析构函数

…First layer derived Object is Destroyed! //对象c的Myderived2基类的析构函数

…BaseClass1 Object is destroyed! //对象c的Myderived2基类类中private成员a1的析构函数

…BaseClass2 Object is destroyed! //对象c的BaseClass2基类的基类析构函数

…BaseClass1 Object is destroyed! //对象c的BaseClass2基类的基类private成员的析构函数

…First layer derived Object is Destroyed! //对象b的Myderived2派生类的析构函数

…BaseClass1 Object is destroyed! //对象b的Myderived2派生类中private成员a1的析构函数

…BaseClass2 Object is destroyed! //对象b的BaseClass2基类析构函数

…BaseClass1 Object is destroyed! //对象b的BaseClass2基类private成员的析构函数

…BaseClass2 Object is destroyed! //对象a的BaseClass2类析构函数

…BaseClass1 Object is destroyed! //对象a的BaseClass2类private成员a1的析构函数

在创建派生类对象时会依次自上而下调用从基类到派生类的构造函数，销毁时逆序调用析构函数。若类中存在其他类的对象成员，则先调用该成员对应类的构造函数

EX2

(1)

…BaseClass31 Object is created! //对象a的类BaseClass31的构造函数

1 2 3 //对象a中成员a,b,c赋值后的输出

…BaseClass32 Object is created! //对象b的类BaseClass32的构造函数

-858993460 -858993460 -858993460 //对象b的成员x,y,z的初始值，但没有人为设置初始值

4 5 6 //对象b中成员x,y,z赋值之后的输出

…BaseClass32 Object is destroyed! //对象b的BaseClass32类析构函数

…BaseClass31 Object is destroyed! //对象a的BaseClass31类析构函数

(2)

Correct:

#include <iostream>

using namespace std;

class MyBase31 {

int a, b, c;

public:

MyBase31(int x, int y, int z) :a(x), b(y), c(z)

{

cout << "…BaseClass31 Object is created!" << endl;

cout << a << " " << b << " " << c << endl;

}

~MyBase31() { cout << "…BaseClass31 Object is destroyed!" << endl; }

};

class MyBase32 {

int x, y, z;

public:

MyBase32(int a, int b, int c)

{

cout << "…BaseClass32 Object is created!" << endl;

cout << x << " " << y << " " << z << endl;

x = a, y = b, z = c;

cout << a << " " << b << " " << c << endl;

}

int getx() { return x; }//用于访问对象xyz的public函数成员

int gety() { return y; }

int getz() { return z; }

~MyBase32() { cout << "…BaseClass32 Object is destroyed!" << endl; }

};

class MyDerived1 : public MyBase31 {

MyBase32 a; //创建属于Mubase32类对象的私有成员

int c; //int私有成员

public:

MyDerived1(int x) : c(x), a(x, 8, 9), MyBase31(x, 8, 9) //构造函数，使用初始化器，使c=x,构造a(x,8,9)和基类构造函数(x,8,9)

{

cout << "…Base Object has been created!" << endl;

cout << "…Member Object has been created! "

<< a.getx() << " " << a.gety() << " " << a.getz() << endl;//使用函数成员返回xyz的值

cout << "…Derived Object is created! " << c << endl;

}

};

int main()

{

MyBase31 a(1, 2, 3);

MyBase32 b(4, 5, 6);

MyDerived1 c(88);

return 0;

}

…BaseClass31 Object is created! //对象a的Mybase31类构造函数

1 2 3 //输出对象a的成员a,b,c赋值后的值

…BaseClass32 Object is created! //对象b的Mybase32类构造函数

-858993460 -858993460 -858993460 //输出对象b的成员x,y,z初始值，但未人为设置初始值

4 5 6 //输出对象b的成员x,y,z赋值后的值

…BaseClass31 Object is created! //对象c基类Mybase31的构造函数

88 8 9 //该Mybase31对象的输出值

…BaseClass32 Object is created! //对象c成员a的Mybase32类构造函数

-858993460 -858993460 -858993460 //该对象a的成员x,y,z的初始值，未人为设置初始值

88 8 9 //该对象a的成员x,y,z赋值后的值输出

…Base Object has been created! //对象c的派生类Myderived1类构造函数

…Member Object has been created! 88 8 9 //输出对象a的x,y,z值

…Derived Object is created! 88 //输出对象c其中成员c的值

…BaseClass32 Object is destroyed! //对象c的成员a的Mybase32类析构函数

…BaseClass31 Object is destroyed! //对象c基类Mybase31的析构函数

…BaseClass32 Object is destroyed!//对象b的Mybase32类析构函数

…BaseClass31 Object is destroyed!//对象a的Mybase31类析构函数

派生类不能直接访问基类的私有成员（题目原代码中的错误），应当使用对应类的成员函数来间接访问其对象。

在初始化时不仅对自身类进行初始化，对基类也进行初始化

EX4

//1)To create a base class as following :

class MyBase {

int x;

public:

MyBase(int a) :x(a) {};

int getX() { cout << " " << endl; return x; }

};

//2)To create a derived class as following :

class MyDerived : public MyBase {

int y;

public:

MyDerived(int a) :y(a), MyBase(a + 4){};

int getY() { cout << " " << endl; return y; }

};

//3)To create a test program as following and analyze the result.

int main()

{

MyBase a(2), \*p = &a; //类指针p应指向a所在的地址

MyDerived b(4), \*q = &b; //类指针q应指向b所在的地址

MyBase &c = a;

MyBase &d = b;

cout << a.getX() << " " << p->getX() << endl;

cout << b.getY() << " " << q->getY() << b.getX() << " " << q->getX() << endl;

a = b;

cout << a.getX() << " " << a.getY() << endl;

//此处，a是基类对象，不能够访问派生类的getY成员函数

p = q;

cout << p->getX() << " " << p->getY() << endl;

//此处，p是指向基类对象的指针，同样不能够访问派生类的getY成员函数

cout << c.getX() << " " << d.getX() << " " << d.getY() << endl;

//d本身是基类对象，虽然引用了派生类b，但并不能访问派生类的成员函数getY

b = a;

//对象不能由基类转化至派生类

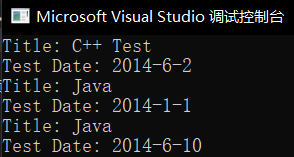
cout << b.getX() << " " << b.getY() << endl;

}

类之间的传递只存在同类→同类，派生类→基类，而不存在基类→派生类，

类指针访问类成员时应使用->，引用类时，以引用的对象类型为准而不是被引用的对象

EX5



#include<iostream>

using namespace std;

class Date

{

public:

Date(int a = 2014, int b = 1, int c = 1)//默认构造函数，默认日期2014-1-1

{

year = a;

month = b;

day = c;

}

void date\_print()

{

cout << year << "-" << month << "-" << day;//打印日期 year-month-day

}

void setDate(Date date)//设置日期的函数

{

year = date.year;

month = date.month;

day = date.day;

}

private:

int year, month, day;

};

class FinalTest:public Date

{

public:

FinalTest(const char\* n, Date date)//派生类FinalTest的构造函数

{

name = n;

setDate(date);//调用基类的函数

}

FinalTest(const char\* n=NULL)//重载构造函数

{

name = n;

}

void print()

{

cout << "Title: " << name << endl;

cout << "Test Date: ";

date\_print();//调用基类函数输出日期

cout << endl;

}//打印考试信息

private:

const char\* name;

};

int main()

{

FinalTest item1("C++ Test", Date(2014, 6, 2));

item1.print();

FinalTest item2("Java");

item2.print();

item2.setDate(Date(2014, 6, 10));

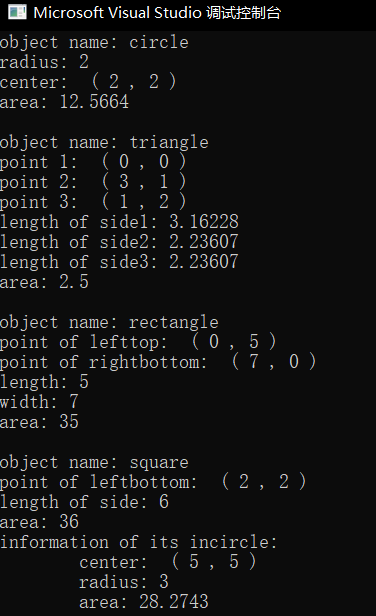
item2.print();

return 0;

}

实现派生类和基类之间交流可以活用set,get这类函数，可以避免派生类无法访问基类成员的情况，同时，可以嵌套调用函数来减少代码量，实现代码的高效利用

Ex6



#include<iostream>

#include<cmath>

#include<string>

using namespace std;

//坐标类

class Coordinate

{

friend ostream &operator<<(ostream &output, const Coordinate &a)

{

cout << " ( " << a.x << " , " << a.y << " ) ";

return output;

}

public:

Coordinate(double a = 0, double b = 0)

{

x = a;

y = b;

}

~Coordinate() {}

void setxy(double a, double b)//设置坐标

{

x = a;

y = b;

}

void print() { cout << " ( " << x << " , " << y << " ) "; }//打印坐标格式( x , y )

double getx() { return x; }

double gety() { return y; }

double distance(Coordinate a)

{

return sqrt((x - a.x)\*(x - a.x) + (y - a.y)\*(y - a.y));

}

Coordinate operator\*=(const Coordinate&a)

{

x = a.x;

y = a.y;

}

private:

double x, y;

};

//定义Shape类

class Shape

{

public:

Shape(string name = "") { id = name; }

void setid(string name) { id = name; }

~Shape() {}

void printname() { cout << "object name: " << id << endl; }

private:

string id;

};

//定义派生类Cricle

class Circle :public Shape

{

friend class Square;//声明友元

public:

Circle(Coordinate c = Coordinate(0, 0), string name = "", double rad = 0) :Shape(name), center(c), radius(rad)

{}

~Circle() {}

double area() { return 3.14159 \* radius\*radius; }//pi取近似值3.14159

void print()

{

printname();

cout << "radius: " << radius << endl;

cout << "center: " << center << endl;

cout << "area: " << area() << endl << endl;

}

private:

double radius;

Coordinate center;

};

//定义派生类Triangle

class Triangle :public Shape

{

public:

Triangle(Coordinate a = Coordinate(0, 0), Coordinate b = Coordinate(0, 0), Coordinate c = Coordinate(0, 0), string name = "NoName")

:Shape(name), point1(a), point2(b), point3(c)

{

side1 = point1.distance(point2);

side2 = point1.distance(point3);

side3 = point2.distance(point3);

//计算三边长

}//设置三角形三顶点位置

~Triangle() {}

double area()

{

double p = (side1 + side2 + side3) / 2;

return sqrt(p\*(p - side1)\*(p - side2)\*(p - side3));//Heron公式计算三角形面积

}

void print()

{

printname();

cout << "point 1: " << point1 << endl;

cout << "point 2: " << point2 << endl;

cout << "point 3: " << point3 << endl;

cout << "length of side1: " << side1 << endl;

cout << "length of side2: " << side2 << endl;

cout << "length of side3: " << side3 << endl;

cout << "area: " << area() << endl << endl;

}

private:

double side1, side2, side3;

Coordinate point1, point2, point3;

};

//定义派生类Rectangle

class Rectangle :public Shape

{

public:

Rectangle(Coordinate a = Coordinate(0, 0), Coordinate b = Coordinate(0, 0), string name = "")

:Shape(name), lefttop(a), rightbottom(b)

{

length = lefttop.gety() - rightbottom.gety();

width = rightbottom.getx() - lefttop.getx();//计算长和宽

}

~Rectangle() {}

double area() { return length \* width; }

void print()

{

printname();

cout << "point of lefttop: " << lefttop << endl;

cout << "point of rightbottom: " << rightbottom << endl;

cout << "length: " << length << endl;

cout << "width: " << width << endl;

cout << "area: " << area() << endl << endl;

}

private:

Coordinate lefttop, rightbottom;

double length, width;

};

//定义派生类Square

class Square :public Rectangle

{

public:

Square(Coordinate a, string name = "", double side = 0)

:Rectangle(Coordinate(a.getx(), side + a.gety()), Coordinate(a.gety(), a.getx() + side), name), point(a), side(side)

{}

~Square() {}

double area() { return side \* side; }

Circle incircle()

{

Circle circle(Coordinate(point.getx() + side / 2, point.gety() + side / 2), "", side / 2);

return circle;

}

void print()

{

printname();

cout << "point of leftbottom: " << point << endl;

cout << "length of side: " << side << endl;

cout << "area: " << area() << endl;

cout << "information of its incircle:" << endl;

cout << "\t" << "center: " << incircle().center << endl;

cout << "\t" << "radius: " << incircle().radius << endl;

cout << "\t" << "area: " << incircle().area() << endl << endl;

}

private:

double side;

Coordinate point;

};

int main()

{

Circle obj1(Coordinate(2, 2), "circle", 2);

Triangle obj2(Coordinate(0, 0), Coordinate(3, 1), Coordinate(1, 2), "triangle");

Rectangle obj3(Coordinate(0, 5), Coordinate(7, 0), "rectangle");

Square obj4(Coordinate(2, 2), "square", 6);

obj1.print();

obj2.print();

obj3.print();

obj4.print();

return 0;

}

要注意派生类不能访问基类的私有成员（而公有成员可以），而基类不能访问派生类。类的继承和友元类都可以实现类与类之间的交流