Experiment #3

Questions:

1. To create classes by inheriting from exiting classes.

//Completed.

1. The notations of base classes and derived classes and the relationships between them.

//The declaration of base classes are the same as normal classes.

//The declaration of derived classes need to add ": (public/private/protected) (name of base class)".

//Relationship: derived classes are base classes with more additional functions/variables attached to them.

1. The order in which objects were constructed and destructed in inheritance hierarchies.

//Base class ⬅ Derived class1 ⬅ Derived class2

//Construction: From left to right.

//Destruction: From right to left.(totally opposite to the order of construction)

1. The initial in heritance.

//MyBase1/2.

1. The difference between public, protected and private member access specifier.

//public: can be accessed by any non-static member functions, non-member functions and friend functions.

//protected: can be accessed by any non-static member functions and friend functions.

//private: can be accessed by non-static functions and public/protected friend functions of its own class only.

1. The difference between public, protected and private inheritance.

|  |  |  |  |
| --- | --- | --- | --- |
| Base class member access specifier | Type of inheritance | | |
| Public | Protected | Private |
| Public | **Public** | **Protected** | **Private** |
| Protected | **Protected** | **Protected** | **Private** |
| Private | **Hidden** | **Hidden** | **Hidden** |

1. The inheritance, add and hide of class member functions.

//The same as 6)

1. The translation between base class and derived class.

//An object of a derived class can be translated to an object of its base class

//But if opposite to this, it is a wild operation.

1. 1)

//main.cpp

#include<iostream>

using namespace std;

class MyBase1 {

public:

MyBase1() {

cout << "...BaseClass1 Object is created!" << endl;

}

~MyBase1() {

cout << "...BaseClass1 Object is destroyed!" << endl;

}

};

class MyDerived1 : public MyBase1 {

public:

MyDerived1() {

cout << "...First layer derived Object is created!" << endl;

}

~MyDerived1() {

cout << "...First layer derived Object is Destroyed!" << endl;

}

};

class MyDerived11 : public MyDerived1 {

public:

MyDerived11() {

cout << "...Second layer derived Object is created!" << endl;

}

~MyDerived11() {

cout << "...Second layer derived Object is destroyed!" << endl;

}

};

int main() {

MyBase1 a;

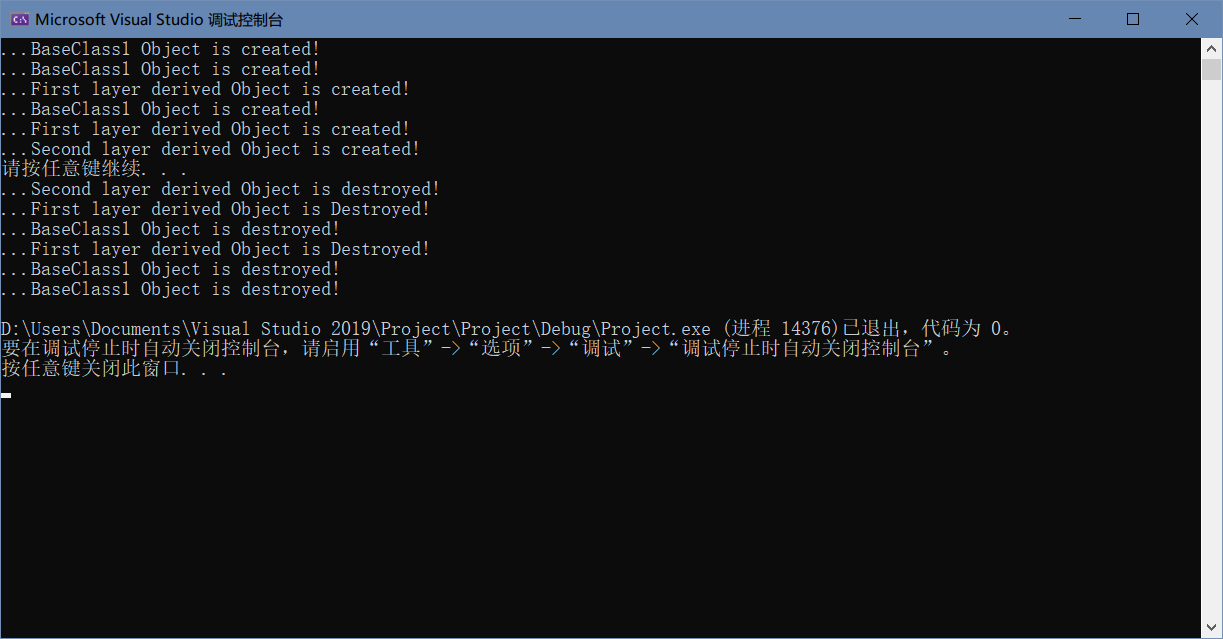
MyDerived1 b;

MyDerived11 c;

system("pause");

return 0;

}



首先，main函数中，先创建MyBase1 a，调用默认构造函数，输出"...BaseClass1 Object is created!"。

继续，创建MyDerived1 b，先调用基类默认构造函数，输出"...BaseClass1 Object is created!"，再输出自身构造函数内容："...First layer derived Object is created!"。

继续，创建MyDerived11 c，调用MyDerived1的构造函数，内容同上，再加上一条自身的内容"...Second layer derived Object is created!"。

结束，按照以上顺序的倒序执行析构函数。

2)

//main.cpp

#include<iostream>

using namespace std;

class MyBase1 {

public:

MyBase1() {

cout << "...BaseClass1 Object is created!" << endl;

}

~MyBase1() {

cout << "...BaseClass1 Object is destroyed!" << endl;

}

};

class MyBase2 {

MyBase1 a1;

public:

MyBase2() {

cout << "...BaseClass2 Object is created!" << endl;

}

~MyBase2() {

cout << "...BaseClass2 Object is destroyed!" << endl;

}

};

class MyDerived1 : public MyBase2 {

MyBase1 a1;

public:

MyDerived1() {

cout << "...First layer derived Object is created!" << endl;

}

~MyDerived1() {

cout << "...First layer derived Object is Destroyed!" << endl;

}

};

class MyDerived11 : public MyDerived1 {

public:

MyDerived11() {

cout << "...Second layer derived Object is created!" << endl;

}

~MyDerived11() {

cout << "...Second layer derived Object is destroyed!" << endl;

}

};

int main() {

MyBase2 a;

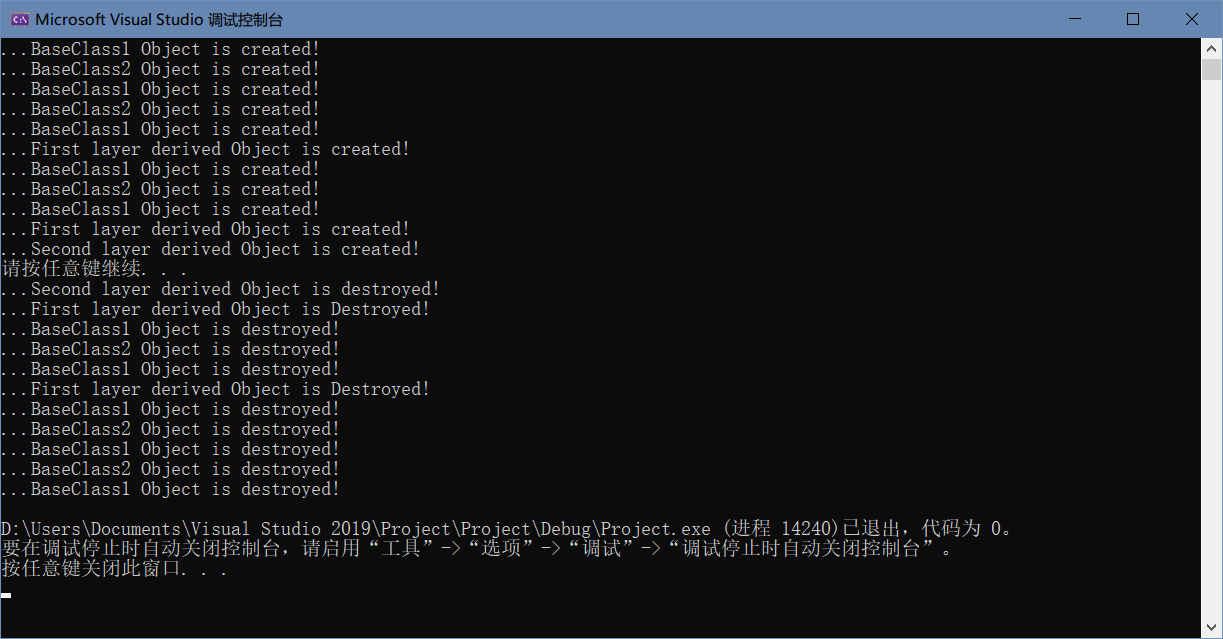
MyDerived1 b;

MyDerived11 c;

system("pause");

return 0;

}



首先，main函数中创建MyBase2 a，MyBase2中又创建了一个成员MyBase1 a1，于是先执行MyBase1的构造函数，输出："...BaseClass1 Object is created!"，再执行MyBase2的构造函数，输出："...BaseClass2 Object is created!"。

继续，创建MyDerived1 b，MyDerived1中又创建了一个成员MyBase1 a1，执行MyBase1的构造函数，输出："...BaseClass1 Object is created!"，再执行其基类内容，顺序同上，然后再执行其自身构造函数，输出"...First layer derived Object is created!"。

继续，创建MyDerived11 c，以MyDerived1为基类，故在以上内容的输出之后，再输出一条其自身构造函数中的内容："...Second layer derived Object is created!"。

结束，析构顺序与上述相反。

2.1)

//main.cpp

#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 a, b, c;

public:

MyBase32(int x, int y, int z) {

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

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

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

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

}

~MyBase32() {

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

}

};

int main() {

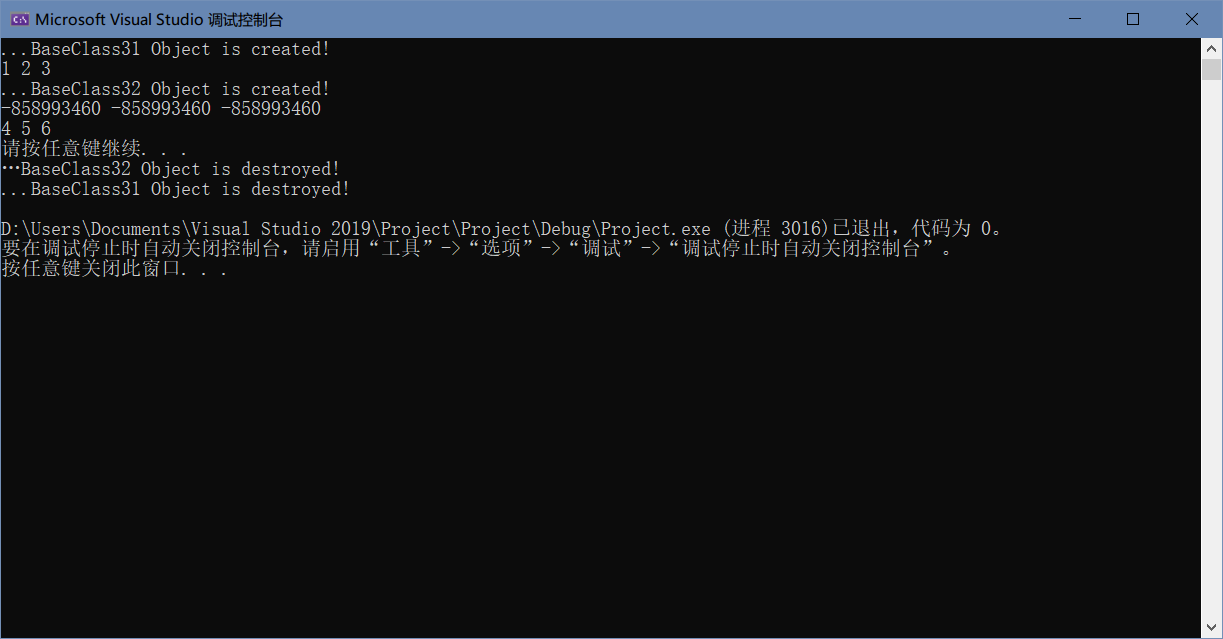
MyBase31 a(1, 2, 3);

MyBase32 b(4, 5, 6);

system("pause");

return 0;

}



首先，在main函数中，先创建MyBase31类型的对象a，传入参数1，2，3。在MyBase31的构造函数中，对私有成员a，b，c分别初始化为1，2，3，并输出"...BaseClass31 Object is created!"以及私有成员a，b，c三者的值。

继续，创建MyBase32类型的对象b，传入参数4，5，6。先输出"...BaseClass32 Object is created!"，然后再输出私有成员a，b，c三者的值，由于未经初始化，所以其值不可控。输出之后，对私有成员a，b，c进行赋值4，5，6，再对其输出，结果为4，5，6。

最后，程序结束，调用析构函数，顺序与创建顺序相反，即先析构MyBase32 b(4, 5, 6)，再是MyBase31 a(1, 2, 3)。

2)

//main.cpp

#include<iostream>

using namespace std;

class MyBase31 {

public:

int a, b, c;

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 MyDerived1 : public MyBase31 {

int c;

public:

MyDerived1(int x)

: c(x), MyBase31(x, 8, 9) {

MyBase31 a(5, 6, 7);

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

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

<< a.a << " " << a.b << " " << a.c << endl;

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

}

};

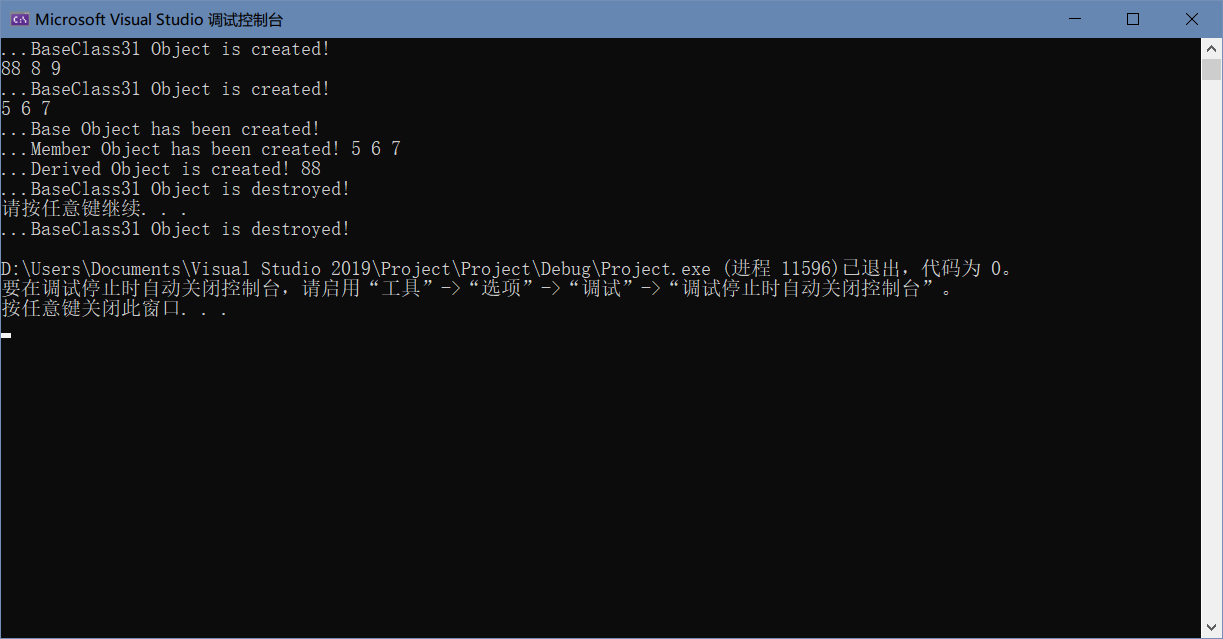
int main() {

MyDerived1 b(88);

system("pause");

return 0;

}



首先，main函数中创建MyDerived1类型的对象b，参数为88。在MyDerived1的构造函数中，先对其私有成员c初始化为88，再对其基类MyBase31初始化，传入的值为88，8，9，此时调用了一次MB31的构造函数，输出"...BaseClass31 Object is created!”和成员a，b，c的值，为88，8，9。

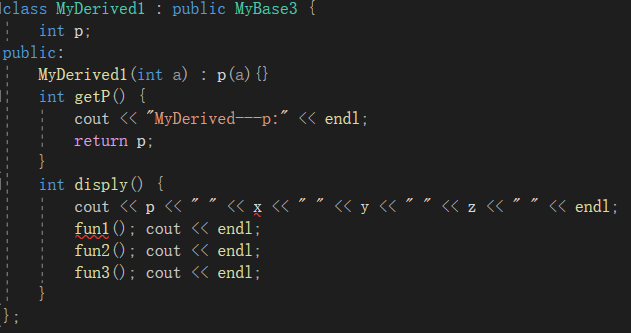
继续，在MyDerived1的构造函数中，先创建一个MyBase31类型的对象，参数5，6，7，同上述步骤，调用MyBase31的构造函数，输出两行内容。

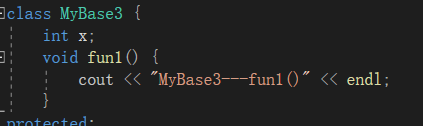
继续，输出"...Base Object has been created!"和"...Member Object has been created! "和MyBase31中成员a，b，c的值，为5，6，7.再输出"...Derived Object is created! "，结束。

调用析构函数，对MyDerived1的构造函数中的MyBase31 a(5, 6, 7)进行析构，输出第一句"...BaseClass31 Object is destroyed!"。

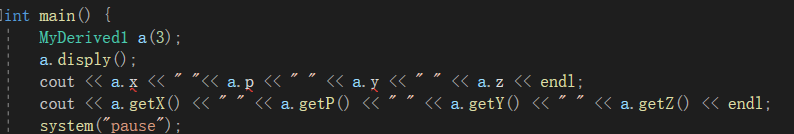
结束，对主函数中的MyDerived1 b(88)进行析构，输出第二句"...BaseClass31 Object is destroyed!"。

3.1)



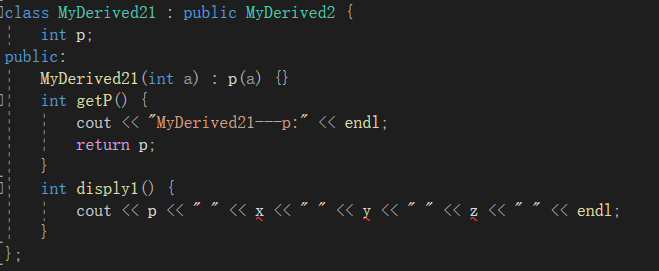


x与fun1，在MyBase3中均在private中声明，所以在派生类中不可访问。



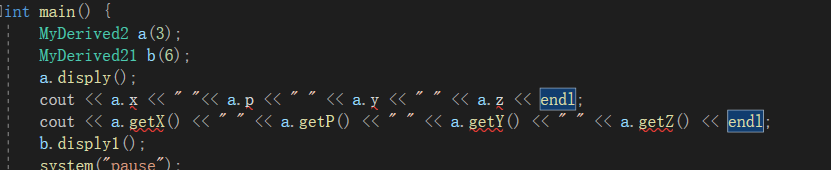
而在主函数中，对于派生类中private的p以及基类中protected的y、private的x，均不可访问，而z以及下面的get函数是public，可以在主函数中访问。

2)



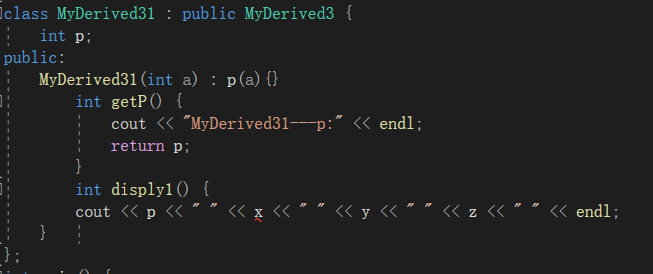


因为在MD2对MB3的继承是private，所以在以MD2为基类的派生类MD21中，x、y、z均不可访问。



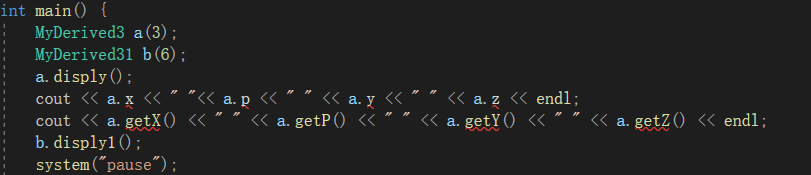
主函数中，对MB3中public的一系列get函数也变为无法访问。

3)





因为MD3对MB3的继承为protected，所以在MD31中，只有private的x不可访问。



而在主函数中，则均不可访问。

4)

//main.cpp

#include<iostream>

using namespace std;

class MyBase {

public:

void f1() { cout << "…MyBase f1-----!" << endl; }

void f2() { cout << "…MyBase f2-----!" << endl; }

};

class MyDerived : public MyBase {

public:

void f2() { cout << "…MyDerived f2-----!" << endl; }

void f22() { MyBase::f2(); cout << "…MyDerived f2-----!" << endl; }

void f3() { cout << "…MyDerived f3-----!" << endl; }

};

int main() {

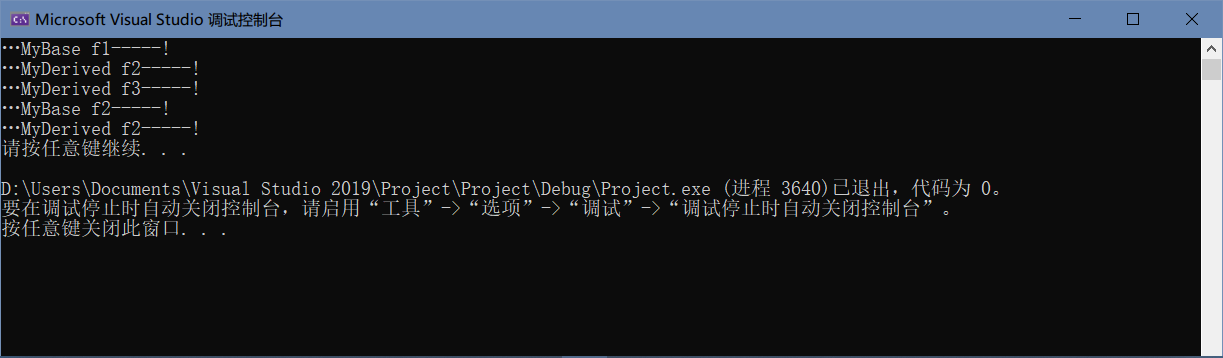
MyDerived a;

a.f1(); a.f2(); a.f3(); a.f22();

system("pause");

return 0;

}



第一个是派生类对象在主函数中调用基类函数（在派生类中无同名函数），直接调用。

第二个是调用派生类与基类中有同名的函数，不加作用域运算符默认调用派生类中的函数。

第三个是调用派生类中的函数（无同名），直接调用。

第四个是调用派生类中的函数，此函数又调用f2，加了MyBase::f2()，所以调用的是基类中的f2函数，输出一行，再加上自身的一行输出，结束。

4.

//main.cpp

#include<iostream>

using namespace std;

class MyBase {

int x;

public:

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

int getX() {

cout << " " << endl;

return x;

}

};

class MyDerived : public MyBase {

int y;

public:

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

int getY() {

cout << " " << endl;

return y;

}

};

int main() {

MyBase a(2), \* p = a;

MyDerived b(4), \* q = b;

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;

p = q;

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

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

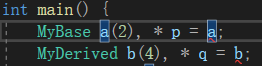
b = a;

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

system("pause");

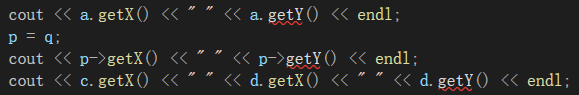
return 0;

}



p是MyBase类型的指针，不可以直接将MyBase类型的a赋值给p；对q同理。

可改为



a、p、d分别是基类MyBase下的对象、指针、引用，而getY()是派生类MyDerived中的函数，基类对象不可以调用派生类对象，因此会报错。



B是派生类对象，a是基类对象，并不允许这种形式的转换。

5.

//Date.h

#ifndef Date\_h

#define Date\_h

class Date {

public:

Date(int = 2014, int = 1, int = 1);

int getY()const;

int getM()const;

int getD()const;

void setY(int);

void setM(int);

void setD(int);

private:

int y, m, d;

};

#endif // !Date\_h

//////////////////////////////////////////////////

//Date.cpp

#include<iostream>

using namespace std;

#include"Date.h"

Date::Date(int year, int month, int day) : y(year), m(month), d(day){}

int Date::getY()const { return y; }

int Date::getM()const { return m; }

int Date::getD()const { return d; }

void Date::setY(int year) { y = year; }

void Date::setM(int month) { m = month; }

void Date::setD(int day) { d = day; }

//////////////////////////////////////////////////

//FinalTast.h

#ifndef FinalTest\_h

#define FinalTest\_h

class FinalTest : public Date {

public:

FinalTest(const string = " ", const Date& = 2014);

void setDue(const Date& = 0);

void print()const;

private:

string T;

};

#endif // !FinalTest\_h

//////////////////////////////////////////////////

//FinalTest.cpp

#include<iostream>

#include<string>

using namespace std;

#include"Date.h"

#include"FinalTest.h"

FinalTest::FinalTest(const string title, const Date& date)

: T(title), Date(date.getY(), date.getM(), date.getD()) {}

void FinalTest::setDue(const Date& date) {

setY(date.getY()); setM(date.getM()); setD(date.getD());

}

void FinalTest::print()const {

cout << "Title: " << T << "\nTest Date: " << getY() << '-' << getM() << '-' << getD() << endl;

}

//////////////////////////////////////////////////

//main.cpp

#include<iostream>

#include<string>

using namespace std;

#include"Date.h"

#include"FinalTest.h"

int main() {

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

item1.print();

FinalTest item2("Java");

item2.print();

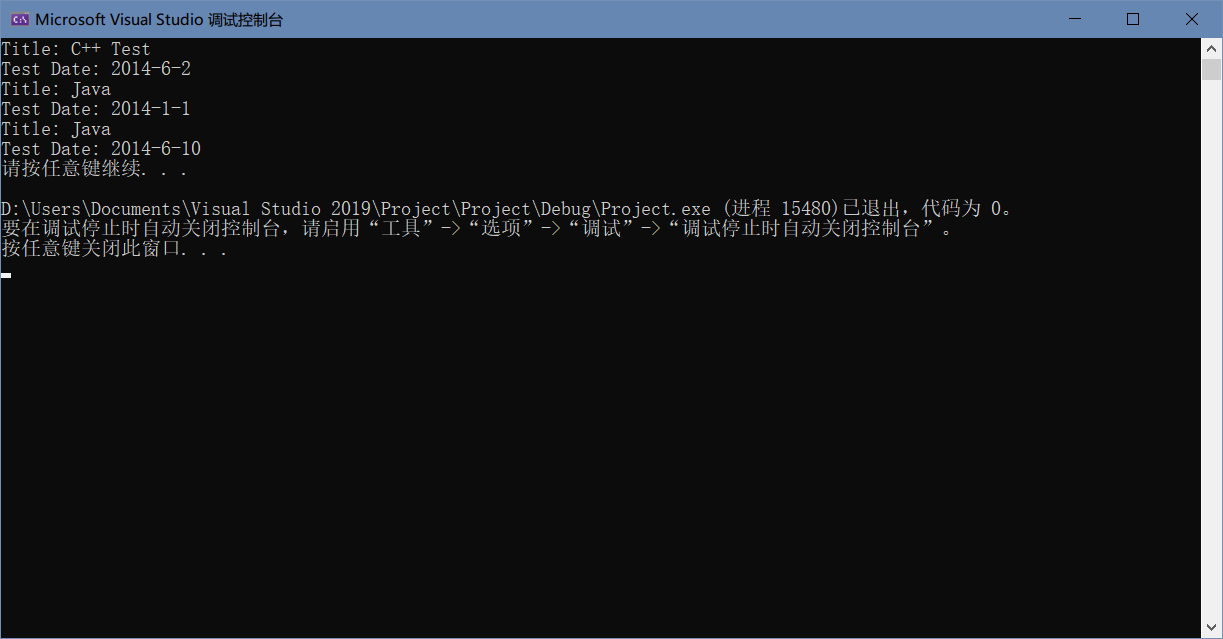
item2.setDue(Date(2014, 6, 10));

item2.print();

system("pause");

return 0;

}



6.

//Shape.h

#ifndef Shape\_h

#define Shape\_h

class Shape {

public:

Shape(const string ID = "defult shape") { this->id = ID; }

~Shape() { cout << "(Shape is destructed.)\n"; }

void Print() { cout << "Shape: " << id << endl << endl; }

protected:

string id;

};

#endif // !Shape\_h

//////////////////////////////////////////////////

//C\_T\_R.h

#ifndef Circle\_Triangle\_Rectangle\_h

#define Circle\_Triangle\_Rectangle\_h

#define PI 3.1415

class Circle : public Shape {

public:

Circle(const string ID, const double C[2], const double R)

:Shape(ID), radius(R) {

center[0] = C[0];

center[1] = C[1];

}

Circle()

:Shape("defult circle"), radius(0) {

center[0] = 0;

center[1] = 0;

}

~Circle() { cout << "(Circle is destructed.)\n"; }

double Area() { return (PI \* radius \* radius / 2); }

void Print(){

cout << "Circle: " << id << endl;

cout << "Center: (" << center[0] << " , " << center[1] << ")\n";

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

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

}

protected:

double center[2], radius;

};

class Triangle : public Shape {

public:

Triangle(const string ID, const double s[3], const double a[3], const double A[2], const double B[2], const double C[2])

:Shape(ID) {

for (int i = 0; i < 3; i++) {

side[i] = s[i];

angle[i] = a[i];

if (i < 2){

vertex\_A[i] = A[i];

vertex\_B[i] = B[i];

vertex\_C[i] = C[i];

}

}

}

Triangle()

:Shape("default triangle"){

for (int i = 0; i < 3; i++) {

side[i] = 0;

angle[i] = 0;

if (i < 2) {

vertex\_A[i] = 0;

vertex\_B[i] = 0;

vertex\_C[i] = 0;

}

}

}

~Triangle() { cout << "(Triangle is destructed.)\n"; }

double Area() { return (side[0] \* side[1] \* sin(angle[2] / 180 \* PI) / 2); }

void Print() {

cout << "Triangle: " << id << endl;

cout << "Side\_a: " << side[0] << ", side\_b: " << side[1] << ", side\_c: " << side[2] << endl;

cout << "Angle\_A: " << angle[0] << "°, angle\_B: " << angle[1] << "°, angle\_C: " << angle[2] << "°\n";

cout << "A: (" << vertex\_A[0] << " , " << vertex\_A[1] << ")\n";

cout << "B: (" << vertex\_B[0] << " , " << vertex\_B[1] << ")\n";

cout << "C: (" << vertex\_C[0] << " , " << vertex\_C[1] << ")\n";

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

}

protected:

double side[3], angle[3], vertex\_A[2], vertex\_B[2], vertex\_C[2];

};

class Rectangle : public Shape {

public:

Rectangle(const string ID, const double s[2], const double A[2], const double B[2], const double C[2], const double D[2])

:Shape(ID) {

for (int i = 0; i < 2; i++) {

side[i] = s[i];

vertex\_A[i] = A[i];

vertex\_B[i] = B[i];

vertex\_C[i] = C[i];

vertex\_D[i] = D[i];

}

}

Rectangle() :Shape("default rectangle") {

for (int i = 0; i < 2; i++) {

side[i] = 0;

vertex\_A[i] = 0;

vertex\_B[i] = 0;

vertex\_C[i] = 0;

vertex\_D[i] = 0;

}

}

~Rectangle() { cout << "(Rectangle is destructed.)\n"; }

double Area() { return (side[0] \* side[1]); }

void Print() {

cout << "Rectangle: " << id << endl;

cout << "Side\_a: " << side[0] << ", side\_b: " << side[1] << endl;

cout << "A: (" << vertex\_A[0] << " , " << vertex\_A[1] << ")\n";

cout << "B: (" << vertex\_B[0] << " , " << vertex\_B[1] << ")\n";

cout << "C: (" << vertex\_C[0] << " , " << vertex\_C[1] << ")\n";

cout << "D: (" << vertex\_D[0] << " , " << vertex\_D[1] << ")\n";

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

}

protected:

double side[2], vertex\_A[2], vertex\_B[2], vertex\_C[2], vertex\_D[2];

};

#endif // !Circle\_Triangle\_Rectangle\_h

//////////////////////////////////////////////////

//Square.h

#ifndef Square\_h

#define Square\_h

class Square :public Rectangle {

public:

Square(string ID,double s,double A[2]){

id = ID;

side[0] = s; side[1] = s;

vertex\_A[0] = A[0], vertex\_A[1] = A[1];

vertex\_B[0] = A[0] + s; vertex\_B[1] = A[1];

vertex\_C[0] = A[0] + s; vertex\_B[1] = A[1] + s;

vertex\_D[0] = A[0]; vertex\_D[1] = A[1] + s;

}

Square() :Rectangle("default square", 0, 0, 0, 0, 0){}

~Square() { cout << "(Suqare is destructed.)\n"; }

Circle inCircle() {

double C[2]{ (vertex\_A[0] + vertex\_B[0]) / 2,(vertex\_A[1] + vertex\_D[1]) / 2 };

double R = side[0] / 2;

Circle IC("Incircle", C, R);

return IC;

}

void Print() {

cout << "Square: " << id << endl;

cout << "Side\_a: " << side[0] << ", side\_b: " << side[1] << endl;

cout << "A: (" << vertex\_A[0] << " , " << vertex\_A[1] << ")\n";

cout << "B: (" << vertex\_B[0] << " , " << vertex\_B[1] << ")\n";

cout << "C: (" << vertex\_C[0] << " , " << vertex\_C[1] << ")\n";

cout << "D: (" << vertex\_D[0] << " , " << vertex\_D[1] << ")\n";

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

cout << "Incircle:\n";

inCircle().Print();

}

protected:

};

#endif // !Square\_h

//////////////////////////////////////////////////

//main.cpp

#include<iostream>

#include<string>

#include<cmath>

using namespace std;

#include"Shape.h"

#include"C\_T\_R.h"

#include"Square.h"

int main() {

//test\_shape:

cout << "//////////test\_shape:\n";

Shape shape\_1;

shape\_1.Print();

Shape shape\_2("Unknown Shape");

shape\_2.Print();

//test\_circle:

cout << "//////////test\_circle:\n";

Circle circle\_1;

circle\_1.Print();

double center\_c[2] = { 6,4 }, radius\_c = 5.8;

Circle circle\_2("Circle\_(6,4)\_5.8", center\_c, radius\_c);

circle\_2.Shape::Print();

circle\_2.Print();

//test\_triangle:

cout << "//////////test\_triangle:\n";

Triangle triangle\_1;

triangle\_1.Print();

double side\_t[3] = { 3,3,3 }, angle\_t[3] = { 60,60,60 }, A\_t[2] = { -1.5,-0.87 }, B\_t[2] = { 1.5,-0.87 }, C\_t[2] = { 0,1.73 };

Triangle triangle\_2("Equilateral Triangle", side\_t, angle\_t, A\_t, B\_t, C\_t);

triangle\_2.Shape::Print();

triangle\_2.Print();

//test\_rectangle:

cout << "//////////test\_rectangle:\n";

Rectangle rectangle\_1;

rectangle\_1.Print();

double side\_r[2] = { 7,5 }, A\_r[2] = { -2.5,-3.5 }, B\_r[2] = { 2.5,-3.5 }, C\_r[2] = { 2.5,3.5 }, D\_r[2] = { -2.5,3.5 };

Rectangle rectangle\_2("Custom Rectangle", side\_r, A\_r, B\_r, C\_r, D\_r);

rectangle\_2.Shape::Print();

rectangle\_2.Print();

//test\_square:

cout << "//////////test\_square:\n";

double side\_s = 6, A\_s[2] = { 3.7,9.4 };

Square square("Custom Square", side\_s, A\_s);

square.Shape::Print();

square.Rectangle::Print();

square.Print();

system("pause");

return 0;

