//知识点：单继承和多继承

1.请分析并写出下面程序的运行结果。

1)

```
#include <iostream>
using namespace std;
class A    {
public:
        A(int n):val(n){}
protected:
        int val;
};
class B : public A    {
public:
        B(int n) :A(n)
        {pB = ( n>0 ?    new    B(n-1) : 0) ; }
        ~B( )    {      delete    pB;    }
        void    Display(    ) {
        cout<<val<<endl;
        if (pB!=0)      pB->Display( );
        }
private:
        B * pB;
};
int main( ) {
        B b(4);
        b.Display();
        return 0;
}
```

2)

```
#include <iostream>
using namespace std;
class A    {
public:
        A(int n):num(n)        { Out( ); }
        A(const A& rhs):num(rhs.num)
        {    Out( );    }
        void Out( ) {    cout<<num<<endl;    }
private:
        int num;
};
class B:public A    {
public:
```

```
    B(A& a) :obj(a),A(1) { }
    void Out( ) {    obj.Out( ); }
private:
    A    obj;
};
int main( ) {
    A a(8);
    B b1(a);
    B b2(b1);
    b2.Out();
    return 0;
}
```

2.定义一个基类 Animal，它包含两个私有数据成员，一个是 string 类型，存储动物的名称(例如"Fido"或"Yogi")，另一个是整数类型 weight，存储动物的重量(单位是磅)。该类还包含一个公有成员函数 who()，它可以显示一个消息，给出 Animal 对象的名称和重量。把 Animal 用作基类，派生两个类 Lion 和 Aardvark。

a) 编写 main()函数，创建 Lion 和 Aardvark 的对象("Leo"，400 磅；"Algernon"，50 磅)，调用派生类对象的 who()成员，说明 who()成员在两个派生类中是继承得来的。

b) 在 Animal 类中，把 who()函数的访问控制符改为 protected，但类的其他内容不变。修改派生类，使原来的 main()函数仍然工作。

c) 在 b 的基础上，修改 main()函数，通过派生类对象调用 who()的基类版本和派生类版本，可根据需要修改 Animal 类中 who()函数的访问控制符。

3.定义一个 Person 类，它包含数据成员 age(年龄)、name(姓名)和 gender(性别)。从 Person 类派生一个类 Employee，在新类中添加一个数据成员，存储个人的 number(编号)。再从 Employee 中派生一个类 Executive，每个派生类都应定义一个函数，来显示相关的信息(名称和类型，如"Fred Smith is an Employee")。编写一个 main()函数，生成两个数组，一个数组包含 5 个 Executive 对象，另一个数组包含 5 个一般的 Employee 对象，然后显示它们的信息。另外，调用从 Employee 类继承的成员函数，显示 Executive 的信息。

4.阅读代码，并按要求练习。

```
class A
{
public:
    A(int num):data1(num) {}
    ~A()
    {
        cout<<" Destory A"<<endl;
    }
    void    f() const
    {
        cout<<" Excute A::f() ";
        cout<<" Data1="<<data1<<endl;
```

```
    }
    void g()
    {
        cout<<" Excute A::g() "<<endl;
    }
private:
    int data1;
};

class B:public A
{
public:
    B(int num1,int num2);
    ~B()
    {
        cout<<" Destory B"<<endl;
    }
    void f( ) const
    {
        cout<<" Excute B::f() ";
        cout<<" Data1="<< data1;
        cout<<" Data2="<<data2<<endl;
    }
    void f(int n) const
    {
        cout<<" Excute B::f(int) ";
        cout<<" n="<<n;
        cout<<" Data1="<< data1;
        cout<<" Data2="<<data2<<endl;
    }
    void h()
    {
        cout<<" Excute B::h() "<<endl;
    }
private:
    int data2;
};
```

1)完成 B 类的构造函数，使得参数 num1 对应 data1，num2 对应 data2；

2)尝试在 main 函数中使用这两个类，编译程序看是否有编译错误？指出错误的原因。

3)将基类中的 private 改为 protected，再编译。理解 protected 访问权限，在 public 继承方式下的可访问性。

4)修改 main 函数，如下所示，看看哪些语句合法？为什么？执行的是基类的实现，还是派生类的实现？

```
int main()
{
	B b(1,2);
	b.f();
	b.g();
	b.f(3);
	b.h();
	return 0;
}
```

5)将继承 A 类的继承方式改为 private，编译能通过吗？再执行 4)中的 main 函数，看看哪些语句变得不合法了？为什么？

6)将继承 A 类的继承方式改回 public，并实现 B 类自定义的拷贝构造和赋值函数。

7)分别创建 A 和 B 类的两个对象 a 和 b，分别执行 a.f()，b.f()，a.g()，b.g()，a.f(1)，b.f(1)，a.h()，b.h()，请问哪些可以通过编译，执行结果如何？

8)增加代码 A * p=new B(1,2);，理解向上类型转换的安全性。

9)在 8)的基础上，执行 p->f()，输出是什么？与 B* p=new B(1,2); p->f();的结果一样吗？

10)在 8)的基础上，执行 p->f(1)，能通过编译吗？为什么？

11)在 8)的基础上，执行 p->g()和 p->h()，能行吗？为什么？

12)在 8)的基础上，执行 delete p;，输出是什么？B 类的析构函数执行了吗？

5.阅读代码，并按要求练习。

```
class A
{
public:
	A(int num):data(num) {}
	void AFuncs()
	{
		cout<<"This is A \'s public function!"<<endl;
	}
protected:
	int data;
};
class B
{
public:
	B(int num):value(num) {}
	void BFuncs()
	{
		cout<<"This is B \'s public function!"<<endl;
	}
protected:
	int value;
};
```

```
class C:public A,private B
{
public:
	C(int num1,int num2,int y);
	void MyFuncs()
	{
		BFuncs();
		cout<<"This function call B::BFuncs() !"<<endl;
	}
private:
	int yyy;
};
```

1)完成 C 类的构造函数。
2)在 main 函数中可以向 C 类对象发送哪些消息？
3)在不改变 C 类的功能的条件下，利用类的组合重新定义并实现 C 类，使其变成单继承。
4)实现新的 C 类的构造、析构、拷贝、赋值函数。

6.某同学设计开发一个游戏，游戏中有墙(Wall)和门(Door)，他给出了如下的类定义：

```
class Wall
{
public:
	Wall():color(0)
	{
		cout<<"构造一面墙"<<endl;
	}
	void Paint(int newColor)
	{
		color = newColor;
		cout<<"用新颜色粉刷墙"<<endl;
	}
	int GetColor() const
	{
		return color;
	}
private:
	int    color;
};

class Door
{
public:
	Door():openOrClose(false)
	{
		cout<<"构造一扇门"<<endl;
```

```
    }
    void Open()
    {
        if (!IsOpened( ))
        {
            openOrClose = true;
            cout<<"门被打开了"<<endl;
        }
        else
        {
            cout<<"门开着呢！"<<endl;
        }
    }
    void Close()
    {
        if ( IsOpened( ) )
        {
            openOrClose = false;
            cout<<"门被关上了"<<endl;
        }
        else
        {
            cout<<"门关着呢！"<<endl;
        }
    }
    bool IsOpened() const
    {
        return openOrClose;
    }
private:
    bool openOrClose;
};
```

1)请你用多重继承的方式，实现带有一扇门的墙(WallWithDoor)类。功能变更为：

a.当用红色粉刷墙时，关闭门；

b.当用绿色粉刷墙时，打开门；

c.当用其它颜色刷墙时，门的状态不变。

2)用单继承的方法，实现同样功能。

3)用水平关联的方式，实现同样功能。