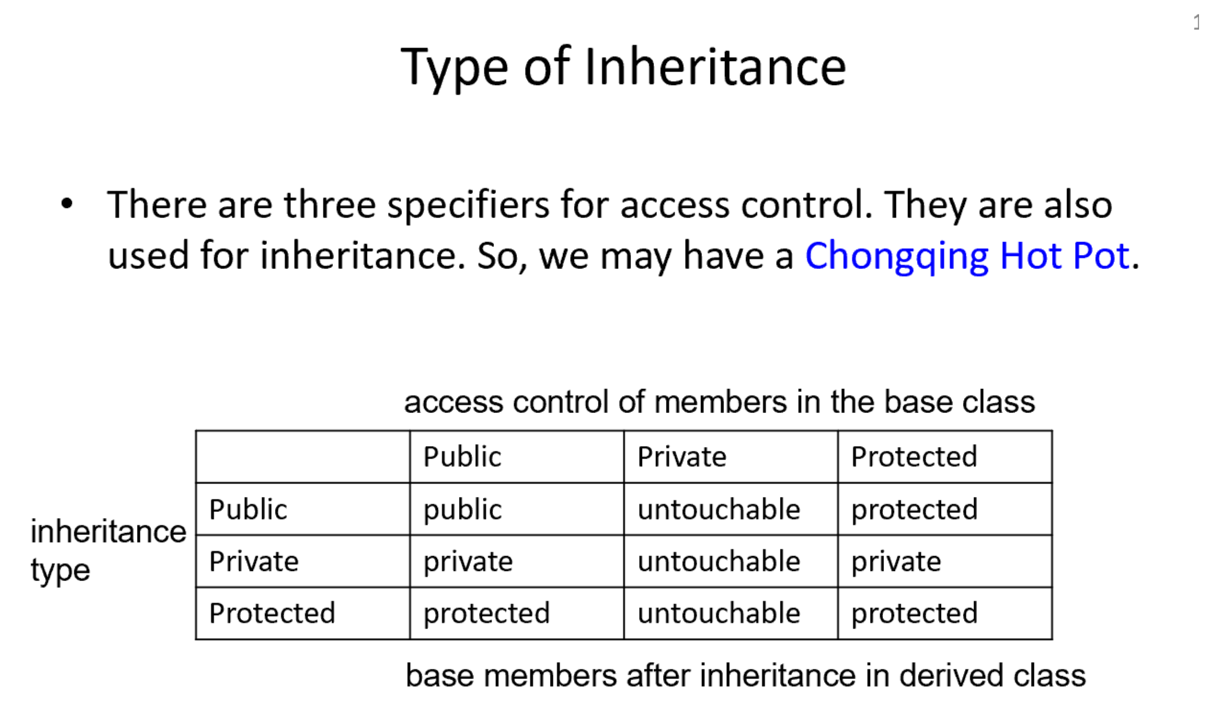
inheritance

```
class DerivedName : public BaseName {
    //just like normal class declaration
}
```

关于访问权限继承表



1

Type of Inheritance

- There are three specifiers for access control. They are also used for inheritance. So, we may have a Chongqing Hot Pot.

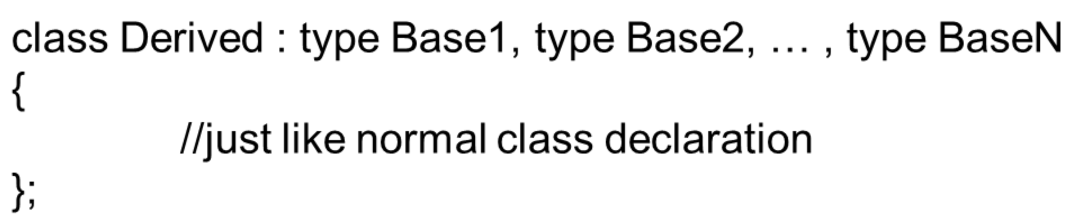
access control of members in the base class

inheritance type		Public	Private	Protected
	Public	public	untouchable	protected
	Private	private	untouchable	private
	Protected	protected	untouchable	protected

base members after inheritance in derived class

注意写继承的时候别写private，这玩意继承不了，写public和protected都行

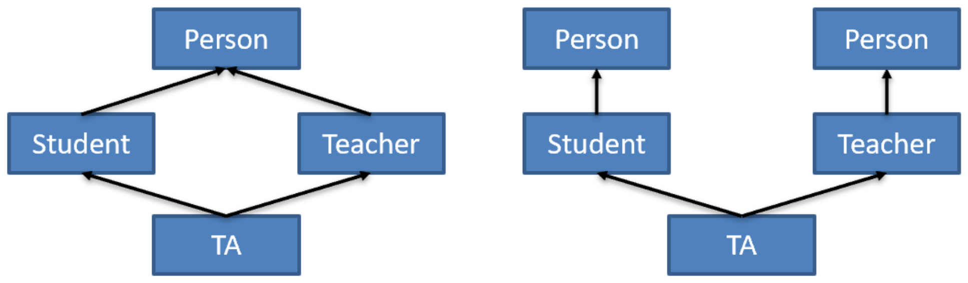
多继承



```
class Derived : type Base1, type Base2, … , type BaseN
{
            //just like normal class declaration
};
```

关于多继承有一个非常重要的概念也就是避免歧义

- Example



我们一般来说正常的逻辑就是以左边的形式来继承，毕竟基类一般只写一个

好，那如果照着之前的方式和顺序那样写下去会发生一些问题

当你在TA中想要调用到Person的一些函数时，程序对于那个函数会创建出两个对于那个函数的路径，一条是Student的，一条是Teacher的

首先这个函数会先继承给st和te，当你在底层派生类要调用时，这个时候st和te都会各自创建出一个person的实例，这样就出现了问题

如何避免这个问题呢，以下是代码

```
class Person {
public:
    void identify() {
        cout << "Person" << endl;
    }
};

class Student : virtual public Person {//别写错位置了
    // Student-specific functionality
};

class Teacher : virtual public Person {//在中间两个类前面加上virtual
    // Teacher-specific functionality
};

class TA : public Student, public Teacher {
    // TA-specific functionality
};

int main() {
    TA ta;
    ta.identify(); // Without virtual inheritance, this call would be ambiguous.
    return 0;
}
```

在中间两个类前面加上virtual关键字来保证只有一个person会被调用

这个时候st和te两个类就成为virtual base class

•After using virtual inheritance, there will be only one base object. The base object is shared between all objects in the inheritance tree and it is only constructed once.

其实如果你实在没有写virtual，你在下面调用的时候为了避免报错也可以这样

```
ta.Teacher::identify();
ta.Student::identity();
```

关于继承中的构造函数写法

```
// 基类
```

```cpp
class Base {
protected:
    int baseValue1;
    int baseValue2;

public:
    Base(int value1, int value2) : baseValue1(value1), baseValue2(value2) {
        cout << "Base constructor called with values " << value1 << " and " <<
value2 << endl;
    }
};

// 派生类
class Derived : public Base {
private:
    int derivedValue;

public:
    Derived(int baseVal1, int baseVal2, int derivedVal)
        : Base(baseVal1, baseVal2), derivedValue(derivedVal) {
        cout << "Derived constructor called with base values " << baseVal1 << ",
" << baseVal2 << " and derived value " << derivedVal << endl;
    }

    void printValues() {
        cout << "Base values: " << baseValue1 << ", " << baseValue2 << " |
Derived value: " << derivedValue << endl;
    }
};

int main() {
    Derived d(5, 15, 25);
    d.printValues();  // 输出基类和派生类的成员变量值
    return 0;
}
```