广州大学

GU CN Computer Networks Institute

面向对象程序设计

Object-Oriented Programming

主讲人：王国军 教授

计算机科学与网络工程学院

csgjwang@gzhu.edu.cn

http://trust.gzhu.edu.cn/

办公室：行政西楼前座532室

根据《C++程序设计基础》（第6版）和《Visual C++面向对象与可视化程序设计》（第4版）制作，仅供广州大学计算机、软件工程、网络工程、网络空间安全、人工智能2023级讲课使用。

第8章 继承

8.1 类之间的关系

8.2 基类和派生类

8.3 基类的初始化

8.4 继承的应用实例

8.5 多继承

MOOC视频与简评

清华郑莉、李超老师-1-导学（8分钟）：

简评：

1、大型程序开发：软件重用（扩充，升级）

2、继承来的东西怎么用？（公有，私有，保护）

3、继承来的东西如何初始化？（派生类传给基类）

4、组合（has-a）和继承（is-a, is-a kind of）

5、使用（uses-a）（Windows程序设计：事件驱动）

MOOC视频与简评

清华郑莉老师-2-继承的基本概念和语法（7分钟）：

简评：

1、继承与派生（文化自信：“没有文明的继承和发展，没有文化的弘扬和繁荣，就没有中国梦的实现”）

2、基类（父类）和派生类（子类）

3、直接基类和间接基类（DAG图，类格）

4、单继承与多继承

5、派生类的构成：吸收基类成员（构造函数C++11“using B::B;”），改造（覆盖）基类成员（第4章作用域、可见性、生存期），添加新的成员

MOOC视频与简评

清华郑莉老师-3-继承方式简介及公有继承（11分钟）：

简评：

1、基类+继承方式：派生类成员（内部）、派生类对象（外部）

公有继承(public)

- 继承的访问控制
  - 基类的public和protected成员：访问属性在派生类中保持不变；
  - 基类的private成员：不可直接访问。
- 访问权限
  - 派生类中的成员函数：可以直接访问基类中的public和protected成员，但不能直接访问基类的private成员；
  - 通过派生类的对象：只能访问public成员。

Point.h × Rectangle.h 7_1.cpp

7_1 Point

```
//Point.h
#ifndef _POINT_H
#define _POINT_H

class Point { //基类Point类的定义
public: //公有函数成员
    void initPoint(float x = 0, float y = 0) { this->x = x; this->y = y;}
    void move(float offX, float offY) { x += offX; y += offY; }
    float getX() const { return x; }
    float getY() const { return y; }
private://私有数据成员
    float x, y;
};

#endif //_POINT_H
```

Point.h | Rectangle.h | 7_1.cpp

7_1 | Rectangle

```
#include "Point.h"

class Rectangle: public Point{ //派生类定义部分
public: //新增公有函数成员
    void initRectangle(float x, float y, float w, float h) {
        initPoint(x, y); //调用基类公有成员函数
        this->w = w;
        this->h = h;
    }
    float getH() const { return h; }
    float getW() const { return w; }
private: //新增私有数据成员
    float w, h;
};
```

Point.h Rectangle.h 7_1.cpp

7_1 (全局范围)

```
#include "Rectangle.h"
using namespace std;

int main() {
    Rectangle rect; //定义Rectangle类的对象
    rect.initRectangle(2, 3, 20, 10); //设置矩形的数据
    rect.move(3,2); //移动矩形位置
    cout << "The data of rect(x,y,w,h): " << endl;
    cout << rect.getX() <<", "    //输出矩形的特征参数
        << rect.getY() << ", "
        << rect.getW() << ", "
        << rect.getH() << endl;
    return 0;
}
```

MOOC视频与简评

清华郑莉老师-4-私有继承和保护继承（10分钟）：

简评：

1、私有继承

私有继承(private)

- 继承的访问控制
  - 基类的public和protected成员：都以private身份出现在派生类中；
  - 基类的private成员：不可直接访问。
- 访问权限
  - 派生类中的成员函数：可以直接访问基类中的public和protected成员，但不能直接访问基类的private成员；
  - 通过派生类的对象：不能直接访问从基类继承的任何成员。

Point.h × Rectangle.h × 7_2.cpp

7_2 | Point

```
//Point.h
#ifndef _POINT_H
#define _POINT_H

class Point { //基类Point类的定义
public: //公有函数成员
    void initPoint(float x = 0, float y = 0) { this->x = x; this->y = y;}
    void move(float offX, float offY) { x += offX; y += offY; }
    float getX() const { return x; }
    float getY() const { return y; }
private://私有数据成员
    float x, y;
};

#endif //_POINT_H
```

Point.h | Rectangle.h | 7_2.cpp

7_2 | Rectangle

```
//Rectangle.h
#ifndef _RECTANGLE_H
#define _RECTANGLE_H
#include "Point.h"

class Rectangle: private Point { //派生类定义部分
public: //新增公有函数成员
    void initRectangle(float x, float y, float w, float h) {
        initPoint(x, y); //调用基类公有成员函数
        this->w = w;
        this->h = h;
    }
    void move(float offX, float offY) { Point::move(offX, offY); }
    float getX() const { return Point::getX(); }
    float getY() const { return Point::getY(); }
```

100 %

Point.h Rectangle.h 7_2.cpp

7_2 (全局范围)

```
#include "Rectangle.h"
using namespace std;

int main() {
    Rectangle rect; //定义Rectangle类的对象
    rect.initRectangle(2, 3, 20, 10); //设置矩形的数据
    rect.move(3,2); //移动矩形位置
    cout << "The data of rect(x,y,w,h): " << endl;
    cout << rect.getX() <<", "    //输出矩形的特征参数
        << rect.getY() << ", "
        << rect.getW() << ", "
        << rect.getH() << endl;
    return 0;
}
```

Point.h | Rectangle.h | 7_2.cpp

7_2 | Rectangle

```
public: //新增公有函数成员
    void initRectangle(float x, float y, float w, float h) {
        initPoint(x, y); //调用基类公有成员函数
        this->w = w;
        this->h = h;
    }
    void move(float offX, float offY) { Point::move(offX, offY); }
    float getX() const { return Point::getX(); }
    float getY() const { return Point::getY(); }
    float getH() const { return h; }
    float getW() const { return w; }
private: //新增私有数据成员
    float w, h;
};
```

100 %

MOOC视频与简评

清华郑莉老师-4-私有继承和保护继承（10分钟）：

简评：

2、保护继承

保护继承(protected)

- 继承的访问控制
  - 基类的public和protected成员：都以protected身份出现在派生类中
  - 基类的private成员：不可直接访问。
- 访问权限
  - 派生类中的成员函数：可以直接访问基类中的public和protected成员，但不能直接访问基类的private成员；
  - 通过派生类的对象：不能直接访问从基类继承的任何成员。

MOOC视频与简评

清华郑莉老师-4-私有继承和保护继承（10分钟）：

简评：

2、保护继承

protected 成员的特点与作用

- 对建立其所在类对象的模块来说，它与 private 成员的性质相同。
- 对于其派生类来说，它与 public 成员的性质相同。
- 既实现了数据隐藏，又方便继承，实现代码重用。

MOOC视频与简评

```
class A {
protected:
   int x;
};

int main() {
   A a;
   a.x = 5;    //错误
}
```

```
class A {
protected:
   int x;
};
class B: public A{
public:
   void function();
};
void B::function() {
   x = 5;   //正确
}
```

MOOC视频与简评

清华郑莉老师-4-私有继承和保护继承（10分钟）：

简评：

3、多继承举例（多基类、不同继承方式）

```
class A {
public:
    void setA(int);
    void showA() const;
private:
    int a;
};
class B {
public:
    void setB(int);
    void showB() const;
private:
    int b;
};
class C : public A, private B {
public:
    void setC(int, int, int);
    void showC() const;
private const:
    int c;
};
```

MOOC视频与简评

```
void  A::setA(int x) {
    a=x;
}
void B::setB(int x) {
    b=x;
}
void C::setC(int x, int y, int z) {
    //派生类成员直接访问基类的
    //公有成员
    setA(x);
    setB(y);
    c = z;
}
//其他函数实现略
```

```
int main() {
    C obj;
    obj.setA(5);
    obj.showA();
    obj.setC(6,7,9);
    obj.showC();
// obj.setB(6);  错误
// obj.showB(); 错误
    return 0;
}
```

MOOC视频与简评

清华郑莉老师-5-基类与派生类类型转换（7分钟）：

简评：

类型转换

- 公有派生类对象可以被当作基类的对象使用，反之则不可。
  - 派生类的对象可以隐含转换为基类对象；
  - 派生类的对象可以初始化基类的引用；
  - 派生类的指针可以隐含转换为基类的指针。
- 通过基类对象名、指针只能使用从基类继承的成员。

```
#include <iostream>
using namespace std;
class Base1 { //基类Base1定义
public:
    void display() const {
        cout << "Base1::display()" << endl;
    }
};
class Base2: public Base1 { //公有派生类Base2定义
public:
    void display() const {
        cout << "Base2::display()" << endl;
    }
};
class Derived: public Base2 { //公有派生类Derived定义
public:
    void display() const {
        cout << "Derived::display()" << endl;
    }
};
```

```
void fun(Base1 *ptr) {          //参数为指向基类对象的指针
    ptr->display();             //"对象指针->成员名"
}
int main() {    //主函数
    Base1 base1;        //声明Base1类对象
    Base2 base2;        //声明Base2类对象
    Derived derived;    //声明Derived类对象

    fun(&base1);        //用Base1对象的指针调用fun函数
    fun(&base2);        //用Base2对象的指针调用fun函数
    fun(&derived);     //用Derived对象的指针调用fun函数

    return 0;
}
```

例7-3 类型转换规则举例

```
void fun(Base1 *ptr) {          //参数为指向基类对象的指针
    ptr->display();             //"对象指针->成员名"
}
int main() {    //主函数
    Base1 base1;        //声明Base1类对象
    Base2 base2;        //声明Base2类对象
    Derived derived;    //声明Derived类对象

    fun(&base1);        //用Base1对象的指针调用fun函数
    fun(&base2);        //用Base2对象的指针调用fun函数
    fun(&derived);     //用Derived对象的指针调用fun函数

    return 0;
}
```

运行结果：

```
Base1::display()
Base1::display()
Base1::display()
```


我们会期望这个效果

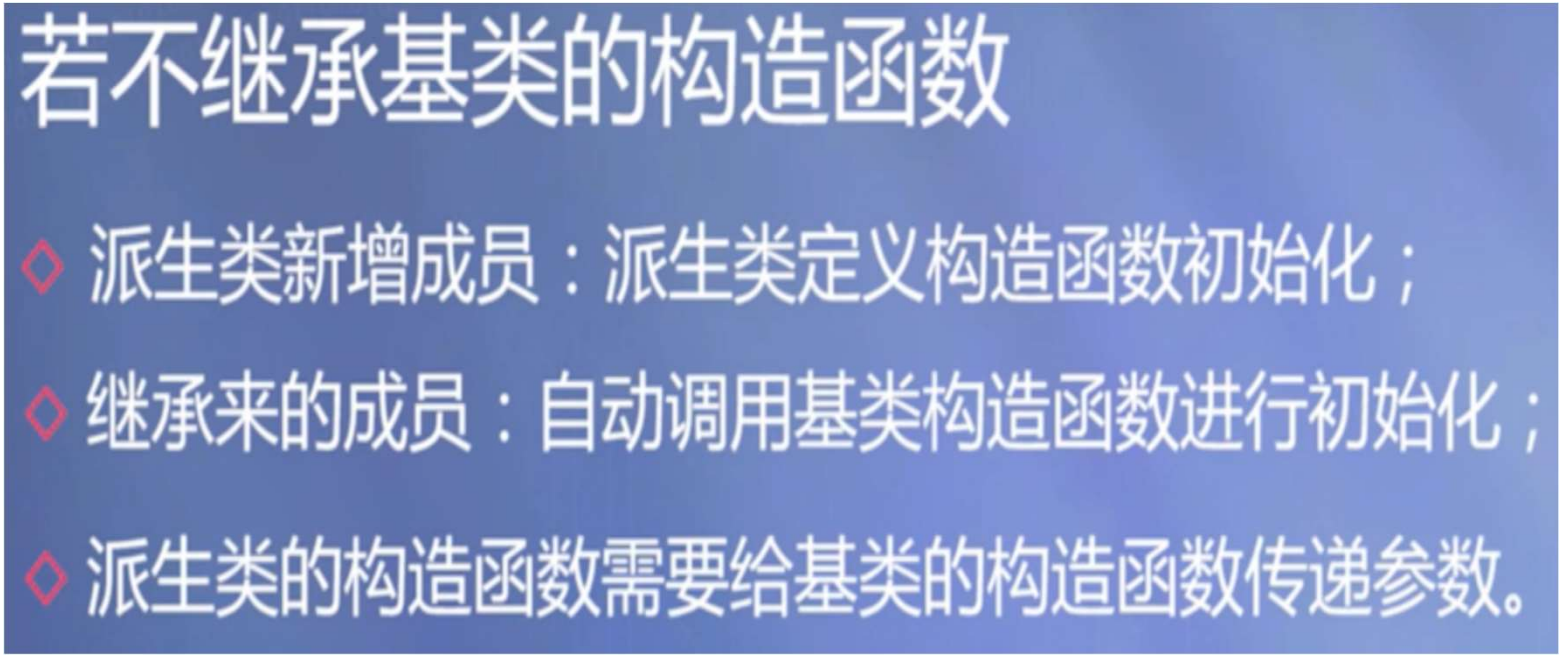
MOOC视频与简评

清华郑莉老师-6-派生类的构造函数（12分钟）：

简评：

1、C++11：using B::B;

2、如果不继承基类的构造函数



MOOC视频与简评

清华郑莉老师-6-派生类的构造函数（12分钟）：

3、单继承情形

单继承时构造函数的定义语法：

- 派生类名::派生类名(基类所需的形参，本类成员所需的形参):
  基类名(参数表), 本类成员初始化列表

```
{
    //其他初始化；
};
```

MOOC视频与简

清华郑莉老师-6-

4、多继承情形

多继承时构造函数的定义语法：

- 派生类名::派生类名(参数表)：
  基类名1(基类1初始化参数表),
  基类名2(基类2初始化参数表),
  ...
  基类名n(基类n初始化参数表),
  本类成员初始化列表
  {
  　　//其他初始化；
  };

MOOC视频与简评

清华郑莉老师-6-派生类的构造函数（12分钟）：

4、多继承情形

派生类与基类的构造函数

- 当基类有默认构造函数时
  - 派生类构造函数可以不向基类构造函数传递参数。
  - 构造派生类的对象时，基类的默认构造函数将被调用。
- 如需执行基类中带参数的构造函数
  - 派生类构造函数应为基类构造函数提供参数。

MOOC视频与简评

清华郑莉老师-6-派生类的构造函数（12分钟）：

5、多继承+类的组合

多继承且有对象成员时派生的构造函数定义语法

- 派生类名::派生类名(形参表):
  基类名1(参数), 基类名2(参数), ..., 基类名n(参数),
  本类成员（含对象成员）初始化列表
  {
  　　//其他初始化
  };

MOOC视频与简评

清华郑莉老师-6-派生类的构造函数（12分钟）：

5、多继承+类的组合

构造函数的执行顺序

1. 调用基类构造函数。
   - 顺序按照它们被继承时声明的顺序（从左向右）。
2. 对初始化列表中的成员进行初始化。
   - 顺序按照它们在类中定义的顺序。
   - 对象成员初始化时自动调用其所属类的构造函数。
     - 由初始化列表提供参数。
3. 执行派生类的构造函数体中的内容。

MOOC视频与简评

清华郑莉老师-7-派生类的构造函数举例（6分钟）：

简评：

```
class Base1 { //基类Base1，构造函数有参数
public:
    Base1(int i) { cout << "Constructing Base1 " << i << endl; }
};

class Base2 { //基类Base2，构造函数有参数
public:
    Base2(int j) { cout << "Constructing Base2 " << j << endl; }
};

class Base3 { //基类Base3，构造函数无参数
public:
    Base3() { cout << "Constructing Base3 *" << endl; }
```

MOOC视频与简评

清华郑莉老师-7-派生类的构造函数举例（6分钟）：

```
class Derived: public Base2, public Base1, public Base3 {
//派生新类Derived，注意基类名的顺序
public: //派生类的公有成员
    Derived(int a, int b, int c, int d): Base1(a), member2(d), member1(c), Base2(b) { }
    //注意基类名的个数与顺序，注意成员对象名的个数与顺序
private://派生类的私有成员对象
    Base1 member1;
    Base2 member2;
    Base3 member3;
};
```

MOOC视频与简评

清华郑莉老师-7-派生类的构造函数举例（6分钟）：

简评：

7_4.cpp

7_4 | Derived

```
public: //派生类的公有成员
    Derived(int a, int b, int c, int d): Base1(a), member2(d), member1(c), Base2(b) { }
    //注意基类名的个数与顺序，注意成员对象名的个数与顺序
private://派生类的私有成员对象
    Base1 member1;
    Base2 member2;
    Base3 member3;
};

int main() {
    Derived obj(1, 2, 3, 4);
    return 0;
}
```

MOOC视频与简评

清华郑莉老师-8-派生类的复制构造函数（3分钟）：

简评：

若派生类没有声明复制构造函数

- 编译器会在需要时生成一个隐含的复制构造函数；
- 先调用基类的复制构造函数；
- 再为派生类新增的成员执行复制。

MOOC视频与简评

清华郑莉老师-8-派生类的复制构造函数（3分钟）：

简评：

若派生类定义复制构造函数

- 一般都要为基类的复制构造函数传递参数。
- 复制构造函数只能接受一个参数，既用来初始化派生类定义的成员，也将被传递给基类的复制构造函数。
- 基类的复制构造函数形参类型是基类对象的引用，实参可以是派生类对象的引用。
- 例如:

```
C::C(const C &c1): B(c1) {...}
```

MOOC视频与简评

清华郑莉老师-9-派生类的析构函数（5分钟）：

简评：

- 析构函数不被继承，派生类如果需要，要自行声明析构函数。
- 声明方法与无继承关系时类的析构函数相同。
- 不需要显式地调用基类的析构函数，系统会自动隐式调用。
- 先执行派生类析构函数的函数体，再调用基类的析构函数。

7_5.cpp

7_5 (全局范围)

```cpp
//7_5.cpp
#include <iostream>
using namespace std;

class Base1 { //基类Base1，构造函数有参数
public:
    Base1(int i) { cout << "Constructing Base1 " << i << endl; }
    ~Base1() { cout << "Destructing Base1" << endl; }
};

class Base2 { //基类Base2，构造函数有参数
public:
    Base2(int j) { cout << "Constructing Base2 " << j << endl; }
    ~Base2() { cout << "Destructing Base2" << endl; }
};
```

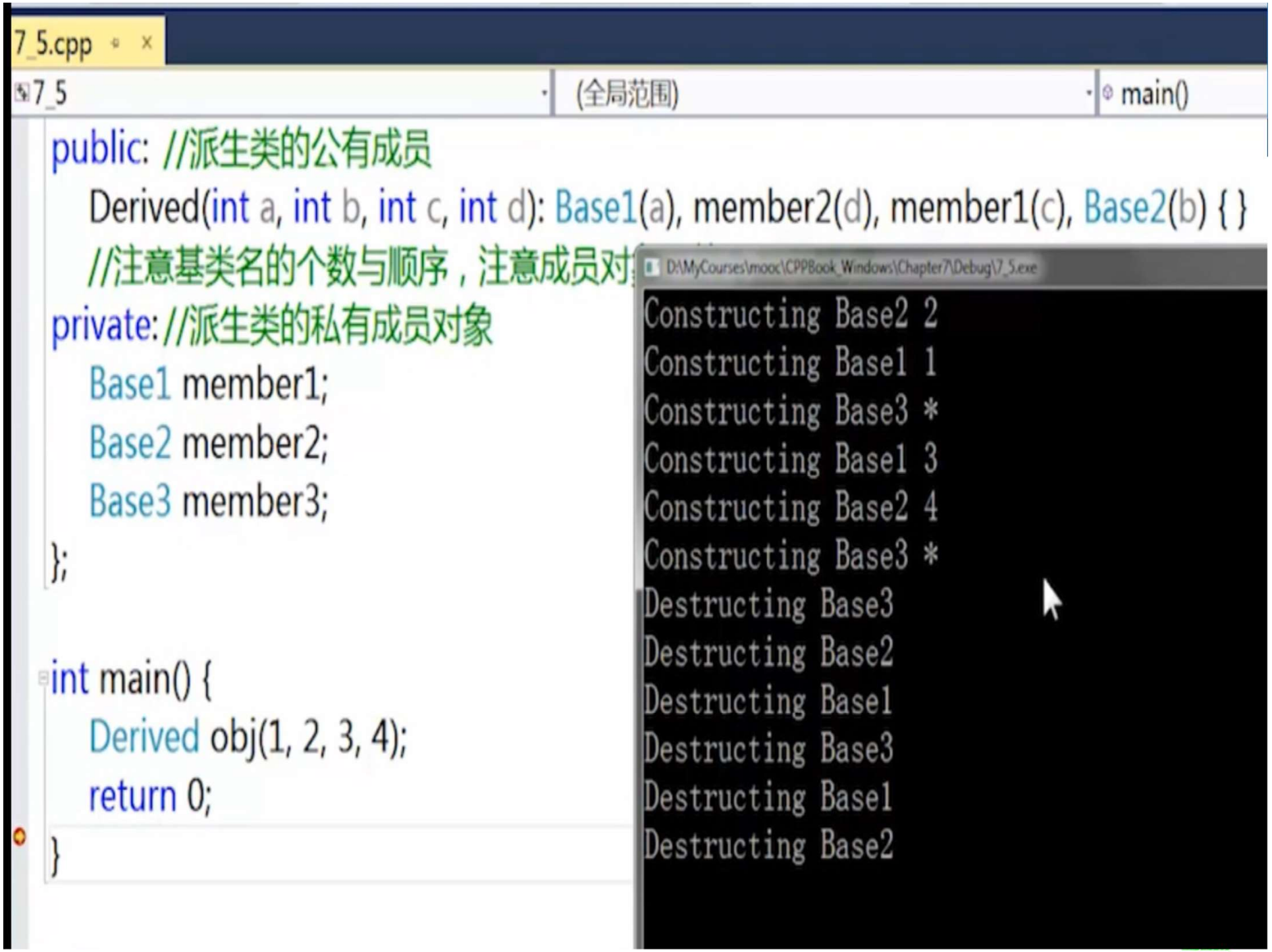
7_5.cpp

7_5 | Base2 | ~Base2()

```
class Base3 {//基类Base3，构造函数无参数
public:
    Base3() { cout << "Constructing Base3 *" << endl; }
    ~Base3() { cout << "Destructing Base3" << endl; }
};

class Derived: public Base2, public Base1, public Base3 {
//派生新类Derived，注意基类名的顺序
public: //派生类的公有成员
    Derived(int a, int b, int c, int d): Base1(a), member2(d), member1(c), Base2(b) { }
    //注意基类名的个数与顺序，注意成员对象名的个数与顺序
private://派生类的私有成员对象
    Base1 member1;
    Base2 member2;
```

100 %


7_5.cpp
7_5
(全局范围)
main()
public: //派生类的公有成员
Derived(int a, int b, int c, int d): Base1(a), member2(d), member1(c), Base2(b) { }
//注意基类名的个数与顺序，注意成员对
private://派生类的私有成员对象
Base1 member1;
Base2 member2;
Base3 member3;
};
int main() {
Derived obj(1, 2, 3, 4);
return 0;
}
D:\MyCourses\mooc\CPPBook_Windows\Chapter7\Debug\7_5.exe
Constructing Base2 2
Constructing Base1 1
Constructing Base3 *
Constructing Base1 3
Constructing Base2 4
Constructing Base3 *
Destructing Base3
Destructing Base2
Destructing Base1
Destructing Base3
Destructing Base1
Destructing Base2

MOOC视频与简评

清华郑莉老师-10-访问从基类继承的成员（8分钟）：

简评：

当派生类与基类中有相同成员时：

- 若未特别限定，则通过派生类对象使用的是派生类中的同名成员。
- 如要通过派生类对象访问基类中被隐藏的同名成员，应使用基类名和作用域操作符（::）来限定。

MOOC视频与简评

清华郑莉老师-10-访问从基类继承的成员（8分钟）：

简评：

二义性问题

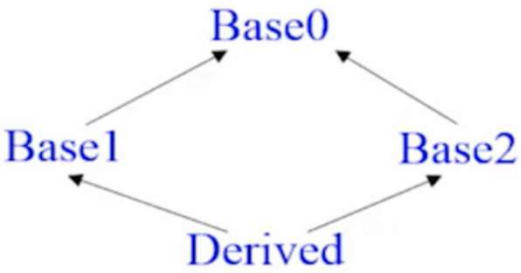
- 如果从不同基类继承了同名成员，但是在派生类中没有定义同名成员，“派生类对象名或引用名.成员名”、“派生类指针->成员名”访问成员存在二义性问题
- 解决方式：用类名限定。

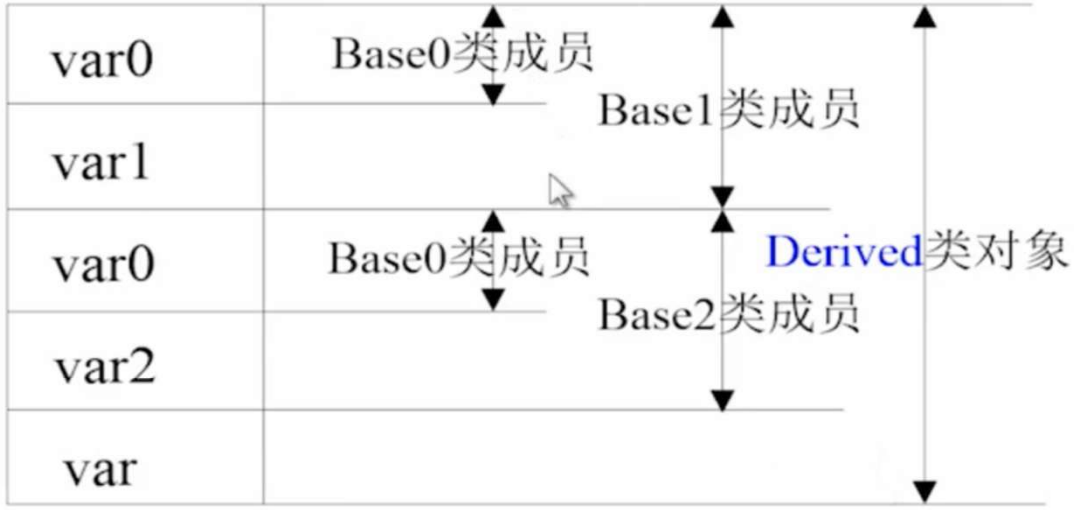
MOOC视频与简评

清华郑莉老师-10-访问从基类继承的成员（8分钟）：

简评：

1、多继承的二义性和冗余问题





有二义性：
d.var0
d.Base0::var0

无二义性：
d.Base1::var0
d.Base2::var0

```cpp
//7_7.cpp
#include <iostream>
using namespace std;
class Base0 {    //定义基类Base0
public:
    int var0;
    void fun0() { cout << "Member of Base0" << endl; }
};
class Base1: public Base0 {        //定义派生类Base1
public: //新增外部接口
    int var1;
};
class Base2: public Base0 {        //定义派生类Base2
public: //新增外部接口
    int var2;
};
```

```
class Derived: public Base1, public Base2 {
public:
    int var;
    void fun()
  { cout << "Member of Derived" << endl; }
};

int main() {    //程序主函数
    Derived d;
    d.Base1::var0 = 2;
    d.Base1::fun0();
    d.Base2::var0 = 3;
    d.Base2::fun0();
    return 0;
}
```

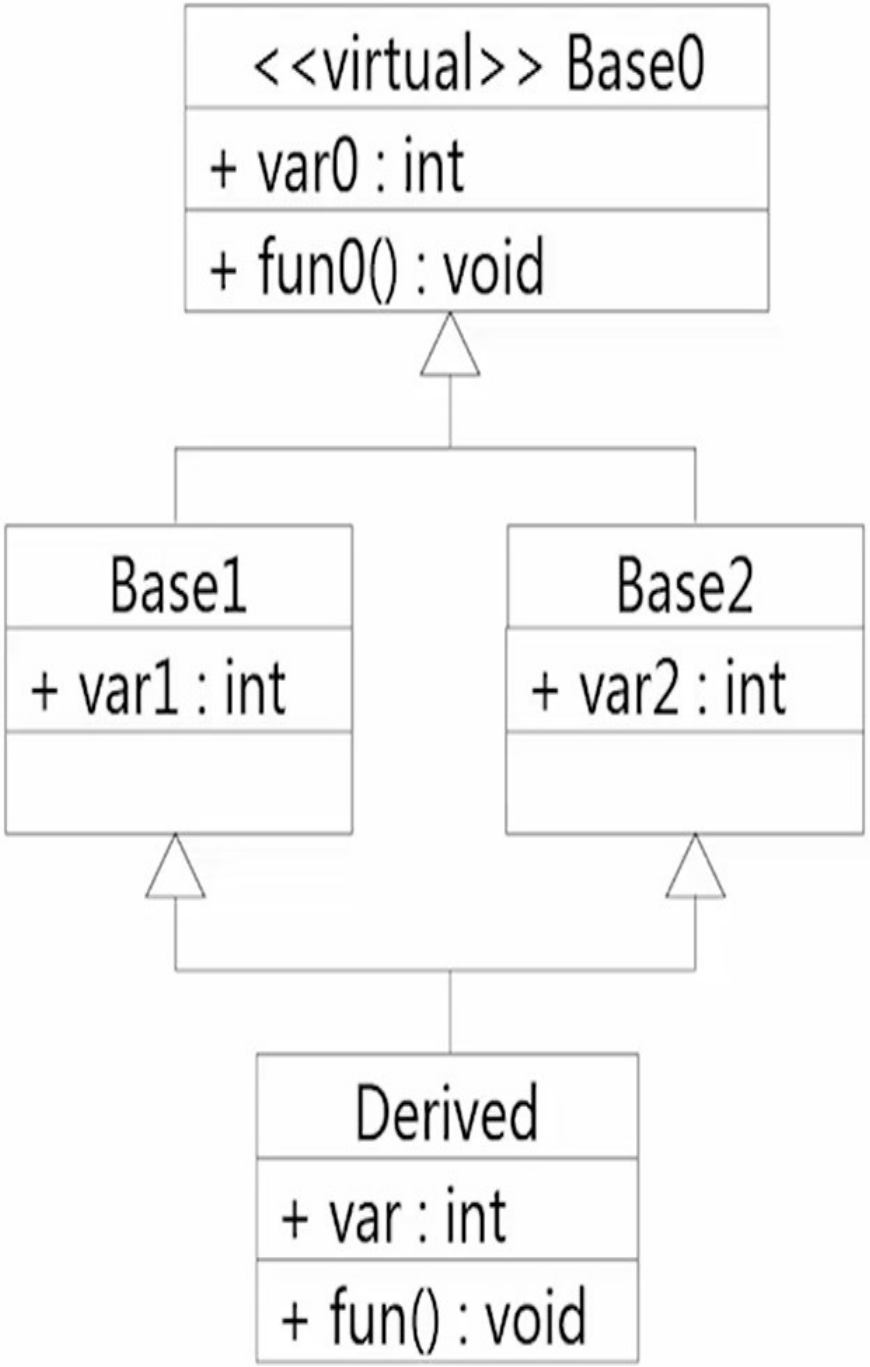
MOOC视频与简评

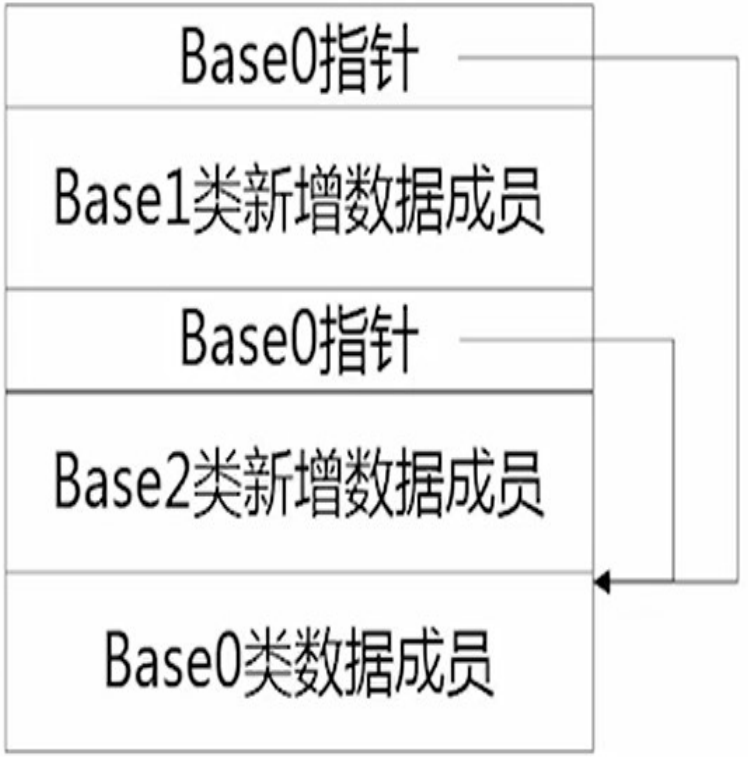
清华郑莉老师-11-虚基类（8分钟）：

- 需要解决的问题
  - 当派生类从多个基类派生，而这些基类又有共同基类，则在访问此共同基类中的成员时，将产生冗余，并有可能因冗余带来不一致性。
- 虚基类声明
  - 以virtual说明基类继承方式

例：class B1:virtual public B

- 作用
  - 主要用来解决多继承时可能发生的对同一基类继承多次而产生的二义性问题；
  - 为最远的派生类提供唯一的基类成员，而不重复产生多次复制。
- 注意：
  - 在第一级继承时就要将共同基类设计为虚基类。


<<virtual>> Base0
+ var0 : int
+ fun0() : void
Base1
+ var1 : int
Base2
+ var2 : int
Derived
+ var : int
+ fun() : void


Base0指针
Base1类新增数据成员
Base0指针
Base2类新增数据成员
Base0类数据成员

Derived对象

```
#include <iostream>
using namespace std;
class Base0 {
public:
    int var0;
    void fun0() { cout << "Member of Base0" << endl; }
};
class Base1: virtual public Base0 {
public:
    int var1;
};
class Base2: virtual public Base0 {
public:
    int var2;
};
```

```
class Derived: public Base1, public Base2 {
//定义派生类Derived
public:
    int var;
    void fun() {
        cout << "Member of Derived" << endl;
    }
};

int main() {
    Derived d;
    d.var0 = 2; //直接访问虚基类的数据成员
    d.fun0();      //直接访问虚基类的函数成员
    return 0;
}
```

MOOC视频与简评

虚基类及其派生类构造函数

- 建立对象时所指定的类称为最远派生类。
- 虚基类的成员是由最远派生类的构造函数通过调用虚基类的构造函数进行初始化的
- 在整个继承结构中，直接或间接继承虚基类的所有派生类，都必须在构造函数的成员初始化表中为虚基类的构造函数列出参数。如果未列出，则表示调用该虚基类的默认构造函数。
- 在建立对象时，只有最远派生类的构造函数调用虚基类的构造函数，其他类对虚基类构造函数的调用被忽略。

```
class Base0 {
public:
    Base0(int var) : var0(var) { }
    int var0;
    void fun0() { cout << "Member of Base0" << endl; }
};
class Base1: virtual public Base0 {
public:
    Base1(int var) : Base0(var) { }
    int var1;
};
class Base2: virtual public Base0 {
public:
    Base2(int var) : Base0(var) { }
    int var2;
};
```

直接的派生类Base1

```
class Derived: public Base1, public Base2 {
public:
    Derived(int var) : Base0(var), Base1(var), Base2(var)
   { }
    int var;
    void fun()
   { cout << "Member of Derived" << endl; }
};

int main() {     //程序主函数
    Derived d(1);
    d.var0 = 2; //直接访问虚基类的数据成员
    d.fun0();     //直接访问虚基类的函数成员
    return 0;
}
```

MOOC视频与简评

清华郑莉老师-12-小结（5分钟）：

本章主要内容

- 继承与派生的基本概念
- 单继承与多继承
- 类成员的访问控制
- 派生类对象的构造和析构
- 派生类与基类对象的类型转换
- 类成员的标识与访问
- 虚继承

作　　业

习题

第8章 测验题

单选题 1分

设置

（8.1）一个大的应用程序，通常由多个类构成，类与类之间互相协同工作，它们之间有三种主要关系。下列不属于类之间关系的是（　）。

A gets-a

B has-a

C uses-a

D is-a

提交

单选题 1分

设置

（8.1）在C++中，类之间的继承关系具有（　）。

A 自反性

B 传递性

C 对称性

D 反对称性

提交

单选题 1分

设置

（8.1）下列关于类之间关系的描述，正确的是（　）。

A has-a表示一个类部分地使用另一个类

B uses-a表示类的包含关系

C is-a关系具有对称性

D is-a机制称为“继承”

提交

单选题 1分

设置

（8.1）下列关于类的描述，正确的是（ ）。

A uses-a表示类的继承机制

B 一个类只能从一个类继承

C is-a关系具有传递性

D 父类具有子类的特征

提交

GU CN Computer Networks Institute

单选题 1分

设置

（8.1）下列关于使用有向无环图（DAG）表示的类之间关系的描述，错误的是（　）。

A DAG表示类之间关系，称为“类格”

B DAG中每个结点是一个类定义，它有且仅有一个前驱结点

C DAG中每个结点是一个类定义，它的后继结点称为派生类

D DAG中每个结点是一个类定义，它的前驱结点称为基类

提交

单选题 1分

设置

（8.1）下列关于类的继承描述中，正确的是（　）。

A 派生类公有继承基类时，可以访问基类的所有数据成员，调用所有成员函数。

B 派生类也是基类，所以它们是等价的。

C 派生类对象不会建立基类的私有数据成员，所以不能访问基类的私有数据成员。

D 一个基类可以有多个派生类，一个派生类可以有多个基类。

提交

单选题 1分

设置

（8.2）当一个派生类公有继承一个基类时，基类中的所有公有成员成为派生类的（　）。

A public成员

B protected成员

C private成员

D 友元

提交

GU CN Computer Networks Institute

单选题 1分

设置

（8.2）当一个派生类私有继承一个基类时，基类中的所有公有成员和保护成员成为派生类的（　）。

A public成员

B protected成员

C private成员

D 友元

提交

单选题 1分

设置

（8.2）当一个派生类保护继承一个基类时，基类中的所有公有成员和保护成员成为派生类的（　）。

A public成员

B protected成员

C private成员

D 友元

提交

GU CN Computer Networks Institute

单选题 1分

设置

（8.2）不论派生类以何种方式继承基类，都不能直接使用基类的（　）。

A public成员

B protected成员

C private成员

D 所有成员

提交

GU CN Computer Networks Institute

单选题 1分

设置

（8.2）在C++中，不加说明，则默认的继承方式是（　）。

A public

B protected

C private

D public或protected

提交

单选题 1分

设置

（8.2）某公有派生类的成员函数不能直接访问基类中继承来的某个成员，则该成员一定是基类中的（　）。

A 公有成员

B 保护成员

C 私有成员

D 保护成员或私有成员

提交

单选题 1分

设置

（8.2）下列关于类层次中重名成员的描述，错误的是（　）。

A　C++允许派生类的成员与基类成员重名

B　在派生类中访问重名成员时，屏蔽基类的同名成员

C　在派生类中不能访问基类的同名成员

D　如果要在派生类中访问基类的同名成员，可以显式地使用作用域符指定

提交

单选题 1分

设置

（8.2）下列关于类层次中静态成员的描述，正确的是（ ）。

A 在基类中定义的静态成员，只能由基类的对象访问

B 在基类中定义的静态成员，在整个类体系中共享

C 在基类中定义的静态成员，不管派生类以何种方式继承，在类层次中具有相同的访问性质

D 一旦在基类中定义了静态成员，就不能在派生类中再定义

提交

单选题 1分

设置

（8.3）在C++中，可以被派生类继承的函数是（ ）。

A 构造函数

B 析构函数

C 成员函数

D 友元函数

提交

单选题 1分

设置

（8.3）下列关于派生类对象的初始化，叙述正确的是（　）。

A 是由基类的构造函数实现的

B 是由派生类的构造函数实现的

C 是系统自动完成的，不需要程序设计者干预

D 是由基类和派生类的构造函数实现的

提交

单选题 1分

设置

（8.3）在创建派生类对象时，构造函数的执行顺序是（　）。

A 对象成员构造函数—基类构造函数—派生类本身的构造函数

B 派生类本身的构造函数—基类构造函数—对象成员构造函数

C 基类构造函数—派生类本身的构造函数—对象成员构造函数

D 基类构造函数—对象成员构造函数—派生类本身的构造函数

提交

单选题 1分

设置

（8.3）在具有继承关系的类层次体系中，析构函数执行的顺序是（　）。

A 对象成员析构函数—基类析构函数—派生类本身的析构函数

B 派生类本身的析构函数—对象成员析构函数—基类析构函数

C 基类析构函数—派生类本身的析构函数—对象成员析构函数

D 基类析构函数—对象成员析构函数—派生类本身的析构函数

提交

单选题 1分

设置

（8.3）在创建派生类对象时，类层次中构造函数的执行顺序是由（　）。

A 派生类的参数初始式列表的顺序决定的

B 是由类的书写顺序决定的

C 系统规定的

D 是任意的

提交

单选题 1分

设置

（8.5）当不同的类具有相同的间接基类时，（　）。

A 各派生类无法按继承路线产生自己的基类版本

B 为了建立唯一的间接基类版本，应该声明间接基类为虚基类

C 为了建立唯一的间接基类版本，应该声明派生类虚继承基类

D 一旦声明虚继承，基类的性质就改变了，不能再定义新的派生类

提交

单选题 1分

设置

（8.5）下列关于多继承的描述，错误的是（　　）。

A 一个派生类对象可以拥有多个直接或间接基类的成员

B 在多继承时不同的基类可以有同名成员

C 对于不同基类的同名成员，派生类对象访问它们时不会出现二义性

D 对于不同基类的不同名成员，派生类对象访问它们时不会出现二义性

提交

单选题 1分

设置

（8.5）下面关于基类和派生类的描述，正确的是（　　）。

A 一个类可以被多次说明为一个派生类的直接基类，可以不止一次地成为间接基类

B 一个类不能被多次说明为一个派生类的直接基类，可以不止一次地成为间接基类

C 一个类不能被多次说明为一个派生类的直接基类，且只能成为一次间接基类

D 一个类可以被多次说明为一个派生类的直接基类，但只能成为一次间接基类

提交

单选题 1分

设置

（8.5）下列关于虚继承的说明形式的描述，正确的是（　　）。

A 在派生类类名前添加关键字virtual

B 在基类类名前添加关键字virtual

C 在基类类名后添加关键字virtual

D 在派生类类名后，类继承的关键字之前添加关键字virtual

提交

单选题 1分

设置

（8.5）设置虚基类的目的是（　　）。

A 消除二义性

B 简化程序

C 提高运行效率

D 减少目标代码

提交