

# 7.1-5

## 7.1

继承方式不同，派生类对基类的成员访问权限不同。

基类成员	公有派生	私有派生	保护派生
私有成员	不可访问	不可访问	不可访问
保护成员	保护，派生类可访问，外部函数无法访问	私有，派生类可访问，外部函数不可访问	保护，派生类可访问，外部函数无法访问
公有成员	公有，派生类与外部函数均可访问	私有，派生类可访问，外部函数不可访问	保护，派生类可访问，外部函数无法访问

## 7.2

先基类的构造函数，再调用成员对象的构造函数，最后调用派生类调用函数函数体内的语句。

## 7.3

使用域解析运算符

```
1 class B:public A
2 {
3     public:
4     funB()
5     {
6         fn1()//使用B重载之后的fn1
7         A::fn1()//调用A的fn1
8         fn2()//调用A的fn2
9     }
10 }
```

## 7.4

```
1 class classnameA:virtual public/private/protected basename;
2 //其中的basename就是虚基类
```

在菱形继承（某类的部分基类是从另一基类派生而来）中，会出现最终的派生类含有多个最初的基类拷贝的情况，他们都有相同的成员名称，有时这会使类的某些功能出问题，或者浪费内存，为了防止这种多重拷贝基类的情况，就有了虚基类技术，多个类共同的基类如果是虚基类，由这些类派生出来的派生类只含有一个该基类。

## 7.5

```
1  #include <iostream>
2  using namespace std;
3
4  class Mammal
5  {
6  public:
7      Mammal() { cout << "Mammal constructor" << endl; }
8      ~Mammal() { cout << "Mammal destructor" << endl; }
9  };
10 class Dog : public Mammal
11 {
12 public:
13     Dog() { cout << "Dog constructor" << endl; }
14     ~Dog() { cout << "Dog destructor" << endl; }
15 };
16
17 int main()
18 {
19     Dog d;
20     return 0;
21 }
```



```
E:\programs\Clions\week8\A.exe
Mammal constructor
Dog constructor
Dog destructor
Mammal destructor

进程已结束,退出代码0
```

如图，首先调用基类的构造函数，再调用派生类的，析构反过来，先调用派生类的，再调用基类的。

## 7.8-11

### 7.8

在派生类中重载fn1 fn2

```
1  #include <iostream>
2  using namespace std;
3  class base
4  {
5  public:
6      void fn1()
```

```

7      {
8          cout << "base fn1" << endl;
9      }
10     void fn2()
11     {
12         cout << "base fn2" << endl;
13     }
14 };
15 class derived:private base
16 {
17 public:
18     void fn1()
19     {
20         base::fn1();
21     }
22     void fn2()
23     {
24         base::fn2();
25     }
26 };
27
28 int main()
29 {
30     derived d;
31     d.fn1();
32     d.fn2();
33     return 0;
34 }

```

## 7.9

```

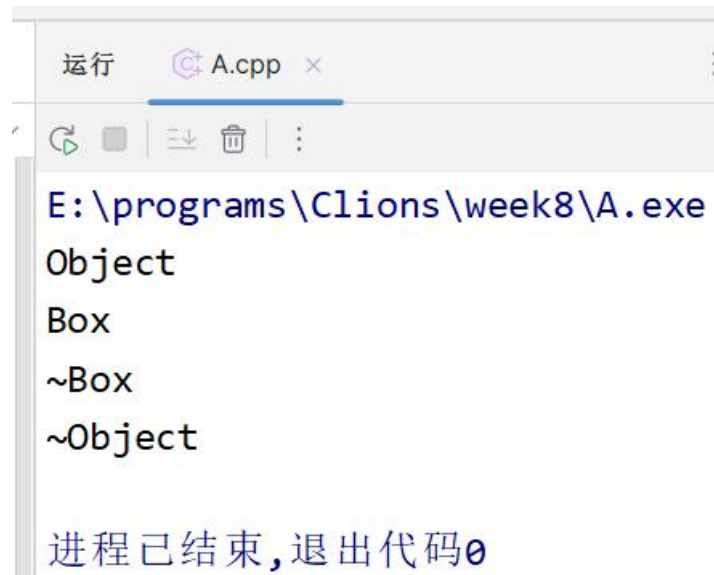
1  #include <iostream>
2  using namespace std;
3  class Object
4  {
5  private:
6      int weight;
7
8  public:
9      Object(int w) : weight(w) { cout << "Object" << endl; }
10     int getweight() { return weight; }
11     ~Object() { cout << "~Object" << endl; }
12 };
13 class Box : public Object
14 {
15 private:
16     int height;
17     int width;
18
19 public:

```

```

20     Box(int w, int h, int wd) : Object(w), height(h), width(wd) { cout << "Box"
    << endl; }
21     int getHeight() { return height; }
22     int getWidth() { return width; }
23     ~Box() { cout << "~Box" << endl; }
24 };
25
26 int main()
27 {
28     Box b(1, 2, 3);
29     return 0;
30 }

```



```

运行 A.cpp x
E:\programs\Clions\week8\A.exe
Object
Box
~Box
~Object

进程已结束,退出代码0

```

## 7.10

```

1  #include <iostream>
2  using namespace std;
3  class BaseClass
4  {
5  public:
6      void fn1()
7      {
8          cout << "BaseClass::fn1()" << endl;
9      }
10     void fn2()
11     {
12         cout << "BaseClass::fn2()" << endl;
13     }
14 };
15 class DerivedClass : public BaseClass
16 {
17 public:
18     void fn1()
19     {
20         cout << "DerivedClass::fn1()" << endl;

```

```

21     }
22     void fn2()
23     {
24         cout << "DerivedClass::fn2()" << endl;
25     }
26 };
27
28 int main()
29 {
30     DerivedClass d;
31     d.fn1();
32     d.fn2();
33     BaseClass *pb = &d;
34     pb->fn1();
35     pb->fn2();
36     DerivedClass *pd = &d;
37     pd->fn1();
38     pd->fn2();
39     return 0;
40 }

```



```

E:\programs\Clions\week8\A.exe
DerivedClass::fn1()
DerivedClass::fn2()
BaseClass::fn1()
BaseClass::fn2()
DerivedClass::fn1()
DerivedClass::fn2()

进程已结束,退出代码0

```

## 7.11

相同：组合、继承都是描述类的方式，以及是代码复用的方式。都使得旧类成为新类的一部分。

差异：组合是"has-a"关系，指的是B中存在A对象，是部分与整体的关系，如学校含有学生。

继承是"is-a"关系，指的是B是一种A，是包含与被包含的关系，如苹果是一种水果。