·第 3 章·
类和对象 II

常见考点

- 常对象的使用方法。
- 常数据成员和常成员函数的使用方法。
- mutable 数据成员的特点。
- 构造函数中初始化列表的使用方法。
- explicit 关键字的作用。
- 子对象的使用方法。
- 使用类前向引用声明的情况。
- 子对象构造函数的设计和执行次序。
- 静态子对象的特点。
- 嵌套类和局部类的设计方法与作用。

3.1 常对象和常对象成员

3.1.1 要点归纳

▷▷▷ 1. 常对象

常对象是指对象常量，其一般定义格式如下：

```
类名 const 对象名;
```

或者

```
const 类名 对象名;
```

🕮 在使用常对象时需要注意以下几点：

❶ 在定义常对象时必须进行初始化。

❷ 常对象的数据成员不能被更新。

❸ 如果一个对象被定义为常对象，则不能调用该对象的非 const 成员函数，否则会报错。这样做是为了防止非 const 成员函数修改常对象中的数据成员值，因为 const 成员函数是不可以修改对象中的数据成员值的。

⋙ 2. 常对象成员

常对象成员包括常数据成员和常成员函数。

1 常数据成员

类的成员数据可以是常量和常引用，使用 const 定义的数据成员为**常数据成员**。如果在一个类中定义了 n 个常数据成员，那么如何给它们赋初值呢？只能通过构造函数，并且构造函数只能通过成员初始化列表来实现，其一般格式如下：

```
构造函数(参数表):常数据成员 1(参数 1),常数据成员 2(参数 2),…, 常数据成员 n(参数 n)
{ … }
```

其中，冒号后面是一个成员初始化列表，它包含一个初始化项，当有多个初始化项时要用逗号分隔开。在执行构造函数时自动将“常数据成员 1”赋值为“参数 1”的值、“常数据成员 2”赋值为“参数 2”的值、…、“常数据成员 n”赋值为“参数 n”的值。

2 对成员初始化列表的深入讨论

实际上，对于类的非常数据成员也可以通过构造函数的成员初始化列表来初始化，只不过非常数据成员还可以在构造函数体内赋值，成员初始化列表是直接调用拷贝构造函数来完成的。

带成员初始化列表的构造函数的执行顺序是这样的：先执行初始化列表，再执行函数体，对于含有多个初始化项的列表不是按照从左到右或者从右到左的顺序执行，而是按照数据成员在类中定义的顺序执行的。例如有以下程序：

```
#include <iostream.h>
class Test
{   int a,b;
    const int x;            //常数据成员
public:
    Test(int i,int j,int k):x(i),a(b),b(j) { b=k; }    //构造函数
    void display()
    {
        cout << "a=" << a << ",b=" << b << ",x=" << x << endl;
    }
};
void main()
{   Test s(1,2,3);
    s.display();
}
```

在定义 s 对象时调用构造函数，由于类 Test 中定义数据成员的顺序是 a、b、x（常数

据成员），先执行 a(b)，由于此时 b 没有初始化，为垃圾值，所以 a 被初始化为一个垃圾值，然后执行 b(j)，即 b=2，再执行 x(i)，即 x=1，最后执行函数体，置 b=3。输出结果如下：

```
a=-858993460（垃圾值）,b=3,x=1
```

无论是在构造函数成员初始化列表中初始化数据成员，还是在构造函数体中对它们赋值，最终结果是相同的。那么两者的性能有什么区别呢？

一般情况下，对于内置的数据类型（含指针和引用），在成员初始化列表和构造函数体内进行，在性能和结果上都是一样的；对于用户自定义的类类型（即子对象的情况），在性能上存在很大的差别，因为子对象在进入构造函数体后已经构造完成，也就是说在成员初始化列表处进行构造对象的工作，这时调用子对象的构造函数，在进入函数体之后进行的是对已经构造好的子对象的赋值，需要调用其赋值运算符才能完成（如果未提供，编译器会提供一个默认的赋值运算符函数按成员进行赋值），而函数调用是浪费资源的，所以在这种情况下采用成员初始化列表来初始化数据成员性能更优。

3 常成员函数

使用 const 关键字声明的函数为**常成员函数**，常成员函数的声明格式如下：

```
函数类型 函数名(参数表) const;
```

📖 在使用常成员函数时需要注意以下几点：

❶ const 是函数类型的一个组成部分，因此在实现部分也要带 const 关键字。

❷ 常成员函数不能更新对象的数据成员，也不能调用该类中的非常成员函数。

❸ 如果将一个对象定义为常对象，则通过该常对象只能调用它的常成员函数，不能调用其他非常成员函数。

❹ const 关键字可以参与区分重载函数。例如，如果在类中有声明：

```
void display();
void display() const;
```

则这是对 display 的有效重载。例如有以下程序：

```
#include <iostream.h>
class Test
{   const int n,m;                      //常数据成员
public:
    Test(int i,int j):n(i),m(j) { }     //构造函数
    void display()                      //重载成员函数
    {  cout << "n=" << n << ",m=" << m << endl; }
    void display() const                //常成员函数
    {  cout << "const n=" << n << ",m=" << m << endl; }
};
void main()
{   Test a(1,2);
```

```
    a.display();
    const Test b(3,4);              //常对象
    b.display();
}
```

在上述程序中类 Test 的 n、m 均为常数据成员，构造函数采用初始化列表给常数据成员赋值。display 成员函数有两个版本。在 main 中定义了对象 a 和常对象 b，前者调用 display 的非 const 版本，后者调用 display 的 const 版本。程序的执行结果如下：

```
n=1,m=2
const n=3,m=4
```

⋙ 3. mutable

在定义一个常对象后只能通过它调用 const 成员函数，而且 const 成员函数不能修改数据成员值。如果希望通过常对象修改某些数据成员值，只需要将其定义为 mutable（易变的），mutable 数据成员永远不会是 const 成员，即使它是一个 const 对象的数据成员。例如有以下程序：

```
#include <iostream.h>
class A
{   mutable int i;  //mutable 数据成员
    int j;          //非 mutable 数据成员
public:
    A(int x,int y) { i=x; j=y; }
    void add() const { i++; };
    void disp() const { cout << "i=" << i << ",j=" << j << endl; }
};
void main()
{   const A a(1,2);
    cout << "修改前:"; a.disp();
    a.add();
    cout << "修改后:"; a.disp();
}
```

类 A 中的 i 是 mutable 数据成员，所以可以在 const 成员函数中修改它的值，而且常对象 a 通过调用该 const 成员函数改变数据成员 i 的值。由于 j 不是 mutable 数据成员，不能进行像数据成员 i 那样的操作。程序的执行结果如下：

```
修改前:i=1,j=2
修改后:i=2,j=2
```

3.1.2 面试真题解析

【面试题 3-1】下面的说法正确的是（　　）。

A．C++中已有的任何运算符都可以重载

B．const 对象只能调用 const 类型成员函数

C．构造函数和析构函数都可以是虚函数

D．函数重载的返回值的类型必须相同

答：常对象只能调用它的常成员函数。答案为 B。

【面试题 3-2】X 是类名称，下面（　　）写法是错误的。

A．const X *x　　B．X const *x　　C．const X const *x　　D．X * const x

答：在 X * const x 中 const 修饰对象名 x，表示对象指针 x 是不能修改的，所以必须初始化。答案为 D。

【面试题 3-3】假设 A 是一个类，A* abc() const 是该类的一个成员函数的原型。若该函数返回 this 值，当用 x.abc()调用该成员函数后 x 的值（　　）。

A．可能被改变　　B．已经被改变

C．受到函数调用的影响　　D．不变

答：abc()是类 A 的一个常成员函数，只能用常对象调用，在这里调用为 x.abc()，说明 x 是类 A 的常对象，显然调用后 x 的值是不变的。答案为 D。

实际上这道面试题是不严谨的，由于只能用常对象调用常成员函数，所以在这种函数中 this 指针是常指针，而 abc()函数的类型为 A*是错误的，应该改为 A const * abc()。

【面试题 3-4】有如下代码，result 变量的输出结果是（　　）。

```
#include <iostream>
using namespace std;
int i=1;
class MyCls
{
public:
    MyCls():m_nFor(m_nThd),m_nSec(i++),m_nFir(i++),m_nThd(i++) { m_nThd=i; }
    void echo()
    {
        cout << "result:" << m_nFir+m_nSec+m_nThd+m_nFor << endl;
    }
private:
    int m_nFir;
    int m_nSec;
    int m_nThd;
    int &m_nFor;
```

```
};
int main()
{   MyCls oCls;
    oCls.echo();
    return 0;
}
```

A．10　　B．11　　C．9　　D．12　　E．8

答：在创建对象 oCls 时调用构造函数，全局变量 i 的初值为 1，先按 MyCls 类数据成员的顺序执行初始化列表，即 i=1，m_nFir(i++) ⇨ m_nFir=1，i=2；m_nSec(i++) ⇨ m_nSec=2，i=3；m_nThd(i++) ⇨ m_nThd =3，i=4；m_nFor(m_nThd) ⇨ m_nFor 为 m_nThd 的引用，也就是 m_nFor=3。最后执行函数体，m_nThd=i=4，而 m_nFor 为 m_nThd 的引用，所以 m_nFor=4。执行 oCls.echo()，result=m_nFir+m_nSec+m_nThd+m_nFor=1+2+4+4=11。答案为 B。

【面试题 3-5】有一个类 A，其数据成员如下：

```
class A
{
    …
private:
    int a;
public:
    const int b;
    float *&c;
    static const char *d;
    static double *e;
};
```

则构造函数中成员变量一定要通过初始化列表来初始化的是（　　）。

A．a b c　　B．b c　　C．b c d e　　D．b c d　　E．b　　F．c

答：常数据成员 b 必须通过初始化列表来初始化。c 是引用型指针变量，引用必须在定义的时候初始化，并且不能重新赋值，所以也要写在初始化列表中。d 和 e 都是静态指针变量，属于类变量，不能在构造函数中赋值。答案为 B。

【面试题 3-6】以下程序的输出结果是什么？

```
#include <iostream>
using namespace std;
class base
{
```

```
private:
    int m_i;
    int m_j;
public:
    base(): m_j(0), m_i(m_j) { }
    base(int i): m_j(i), m_i(m_j) { }
    int get_i() {  return m_i;  }
    int get_j() {  return m_j;  }
};
int main()
{   base obj(98);
    cout << obj.get_i() << "," << obj.get_j() <<endl;
    return 0;
}
```

答：在 main 函数中定义 obj 对象时调用重载构造函数。由于在 base 类中定义数据成员的顺序是 m_i、m_j，先执行 m_i(m_j)，而 m_j 没有赋值，将垃圾值赋给 m_i，再执行 m_j(i)，置 m_j=98，最后执行重载构造函数的函数体，其为空，所以输出结果是“-858993460（垃圾值）,98”。

【面试题 3-7】下面这个类的声明正确吗？为什么？

```
class A
{
  const int Size=0;
};
```

答：在类 A 中存在数据成员的问题，因为常数据成员必须在构造函数的初始化列表中初始化或者将其设置成 static，同时类数据成员不能在定义时初始化。正确的程序如下：

```
class A
{
    const int Size;
public:
    A():Size(0) {};
};
```

或者

```
class A
{
    static int Size;
};
int A::Size=0;
```

3.2 C++中的 explicit

3.2.1 要点归纳

先看以下程序：

```
#include <iostream.h>
class A
{   int n;
public:
    A(int m) { n=m; }
    void display() { cout << "n=" << n << endl; }
};
void main()
{   A a=1;
    a.display();
}
```

main 函数中的 A a=1 是什么意思呢？实际上 C++的构造函数默认是隐式的，该语句相当于 A a(1)，所以程序正确执行并输出 n=1。

如果不希望这种隐式调用，可以在构造函数前面加上 explicit，即将前面类 A 的重载构造函数改为如下：

```
explicit A(int m) { n=m; }
```

这样 A a=1 就不会调用构造函数，将整数 1 赋值给对象 a，此时程序出现“cannot convert from 'const int' to 'class A'”的编译错误。其只能显式调用构造函数，即采用形如 A a(1)的方式定义对象。

实际上 C++中的 explicit 关键字（其含义是明确的，不隐瞒的）只能用于修饰只有一个参数的类构造函数，它的作用是表明该构造函数是显式的，而非隐式的，从而阻止不允许的经过构造函数进行的隐式转换。类构造函数在默认情况下都是隐式的。

在 C++中一个参数的构造函数（或者除了第 1 个参数外其余参数都有默认值的多参构造函数）承担了两个角色，一个是构造器，二是默认且隐含的类型转换操作符。

所以在写形如 A a=n 这样的代码且恰好类 A 有单个参数的构造函数时编译器自动调用这个构造函数创建一个类 A 的对象 a，而且 explicit 不是重载区分符，如果类 A 中同时出现以下两行代码是不允许的：

```
    explicit A(int m) { n=m; }
    A(int m) { n=m; }
```

3.2.2 面试真题解析

【面试题 3-8】以下程序的输出结果是（　　）。

```
#include <string>       //包含 string 类
#include <iostream>
using namespace std;
class Number
{
public:
    string type;
    Number():type("void") { }
    explicit Number(short):type("short") { }
    Number(int):type("int") { }
};
void Show(const Number &n)
{
    cout  << n.type;
}
void f()
{   short s=42;
    Show(s);
}
void main()
{
    f();
}
```

A．void　　B．short　　C．int　　D．None of the above

答：在调用普通函数 f 时执行 s=42，调用普通 Show(s)，参数传递为 const Number &n=s，那么调用哪个构造函数创建 n 对象呢？由于 Number(short)是 explicit 构造函数，不能调用它，只能调用 Number(int)构造函数（s 由 short 转换为 int 类型），所以输出 int。答案为 C。

【面试题 3-9】有程序如下：

```
#include <iostream>
using namespace std;
class B
{
private:
    int data;
public:
```

```
    B() { cout << "defualt constructor" << endl; }
    B(int i):data(i) { cout << "constructed by parameter " << data << endl; }
    ~B() { cout << "destructed" << endl; }
};
B Play(B b)
{
    return b;
}
int main ()
{   B temp=Play(5);
    return 0;
}
```

回答以下问题：

（1）该程序的输出结果是什么？为什么会有这样的输出？

（2）B(int i) : data(i)这种用法的专业术语叫什么？

（3）对于 Play(5)，形参类型是类对象，而 5 是一个常量，这样写合法吗？为什么？

答：（1）该程序的输出结果如下。

```
constructed by parameter 5
destructed
destructed
```

在执行 B temp=Play(5)时调用普通函数 Play，实参数 5 通过隐含的类型转换调用类 B 的带参构造函数创建对象 b，其 data 数据成员设置为 5，并输出上述第 1 行。函数 Play 返回对象 b，并调用拷贝构造函数由对象 b 创建对象 temp。函数 Play 执行完毕调用析构函数销毁对象 b，输出上述第 2 行。main 执行完毕调用析构函数销毁对象 temp，输出上述第 3 行。

（2）在类 B 的带参构造函数中冒号后面的 B(int i):data(i)是成员变量初始化列表。

（3）合法。单个参数的构造函数如果不添加 explicit 关键字会进行隐式类型转换，如果添加 explicit 关键字会消除这种隐式转换。

3.3　子　对　象

3.3.1　要点归纳

当一个类的数据成员是另一个类的对象时，这个对象称为**子对象**。子对象可以像普通对象那样使用，唯一要考虑的是子对象构造函数和析构函数的执行次序。

1. has-a 关系

如果在类 A 的声明中将类 B 的对象作为数据成员，也就是说类 A 有一个类 B 的对象（即子对象），则称类 A 和类 B 之间是一种 has-a 关系，形如：

```
class B     //声明类 A
{
    …
};
class A     //声明类 B
{   …
    B obj;    //obj 是类 B 的对象、是类 A 的子对象
    …
};
```

这种 has-a 关系是一种组合关系，通常用图 3.1 表示，从类 B 的结点画一条线到类 A 的结点，并在类 A 的结点的一端标记一个实心小圆。

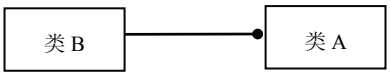

图 3.1　两个类的 has-a 关系图示

2. 子对象构造函数的设计和执行次序

当子对象的类 B 含有构造函数时，在声明类 A 中若定义有 n 个对象成员，则其构造函数的一般格式如下：

```
类名::类名(形参表):子对象名 1(实参表 1)[,子对象名 2(实参表 2),…,子对象名 n(实参表 n)]
{
    //构造函数体
}
```

其中，“子对象名 1(实参表 1)[,子对象名 2(实参表 2),…,子对象名 n(实参表 n)]”为子对象成员初始化表。

📖 对上述构造函数的几点说明如下：

❶ 形参必须带有类型声明，而实参是可计算值的表达式。

❷ 对象成员构造函数的调用顺序取决于这些子对象成员在类中的定义顺序，而与它们在成员初始化表中的位置无关。

❸ 在建立类对象时先调用各个对象成员的构造函数，初始化相应的子对象成员，然后才执行类的构造函数体，初始化本类中的其他数据成员。一般地析构函数的调用顺序与构

造函数正好相反。

▷▷▷ 3. 子对象析构函数的设计和执行次序

同样，当含有子对象的类存在析构函数时特别要注意析构函数的调用次序。在含有子对象的类 A 中设计析构函数如下：

```
~A() { 函数体; }
```

其执行次序是先执行函数体，再以子对象在类 A 中说明的相反次序调用各类的析构函数。例如有以下程序：

```
#include <iostream.h>
class B
{
private:
    int b;
public:
    B()
    {   b=0;
        cout << b << ",调用 B 默认的构造函数" << endl;
    }
    B(int i)
    {   b=i;
        cout << b << ",调用 B 的构造函数" << endl;
    }
    ~B() { cout << b << ",调用 B 的析构函数 "<< endl; }
};
class A
{
private:
    B one,two;
    int a;
public:
    A()
    {   a=0;
        cout << a << ",调用 A 默认的构造函数" << endl;
    }
    A(int i,int j,int k ):two(i+j),one(k)
    {   a=i;
        cout << a << ",调用 A 的构造函数" << endl;
    }
```

```
    ~A() {  cout << a << ",调用 A 的析构函数" << endl;  }
};
void main()
{
    A obj(1,2,4);
}
```

程序的执行结果如下：

```
4,调用 B 的构造函数
3,调用 B 的构造函数
1,调用 A 的构造函数
1,调用 A 的析构函数
3,调用 B 的析构函数
4,调用 B 的析构函数
```

因为在建立类的对象时先调用各个对象成员的构造函数（调用顺序取决于对象成员在类中的定义顺序），然后才执行类的构造函数，所以在本例中建立类 A 的对象 obj 时首先调用对象成员 one 的构造函数 B(int i)，然后调用对象成员 two 的构造函数 B(int i)，最后调用类 A 的构造函数 A()。析构函数的调用顺序与构造函数正好相反。

⋙ 4．静态子对象

在一个类中可以定义另外一个类的静态子对象，静态子对象需要在类外初始化。初始化格式如下：

```
子对象类名 所属类::静态子对象名[(参数表)]
```

如果调用默认的构造函数创建子对象，则不包含“(参数表)”，否则需要包含它。静态子对象属于类对象，和 int 等类型的静态数据成员类似，所不同的是在销毁时需要调用析构函数。例如有以下程序：

```
#include <iostream.h>
class A
{
public:
   A()  { cout << "A 构造函数" << endl; }
   ~A() { cout << "A 析构函数" << endl; }
};
class B
{    static A obj;  //静态子对象
public:
   B() {  cout << "B 构造函数" << endl; }
```

```
    ~B(){  cout << "B 析构函数" << endl; }
};
A B::obj;           //在类体外初始化静态子对象
void main()
{   cout << "main()开始" << endl;
    B b;
    cout << "main()结束" << endl;
}
```

程序的执行结果如下：

```
A 构造函数
main()开始
B 构造函数
main()结束
B 析构函数
A 析构函数
```

第 1 行是创建静态子对象 obj 调用默认构造函数的输出，最后一行是销毁静态子对象 obj 调用默认析构函数的输出，即类的静态子对象具有全局变量的某些特征。

⋙ 5. 前向引用声明

C++中的类通常是先声明后使用。有时大家会遇到两个类相互引用的情况，这时必然要有一个类在声明之前就被引用，解决这一问题的方法是使用前向引用声明。

前向引用声明是在引用未声明的类之前对该类进行预声明，它只是为程序引入一个代表该类的标识符，类的具体声明可以在程序的其他地方。例如有以下程序：

```
#include <iostream.h>
class B;                                    //前向引用声明
class A                                     //A 类的声明
{
public:
    void fa(B b);                           //以 B 类对象 b 为形参的成员函数声明
    void dispa() { cout << "class A" << endl; }
};
class B                                     //B 类的声明
{
public:
    void fb(A a) { a.dispa(); }             //以 A 类对象 a 为形参的成员函数声明
    void dispb() { cout << "class B" << endl; }
};
```

```
void A::fa(B b)
{
    b.dispb();
}
void main()
{   A a;
    B b;
    a.fa(b);
    b.fb(a);
}
```

程序的执行结果如下：

```
class B
class A
```

上述程序的第 2 行给出了类 B 的前向引用声明，表明 B 是一个类，它具有类的一切属性，具体的声明在其他地方。

但要注意的是，前向引用声明仅引入一个代表类的标识符，在没有遇到该类标识符的实际声明之前不能使用该类的任何成员。例如在上例中类 A 的 fa 函数以类 B 对象作为形参，而类 B 采用了前向引用声明，所以是正确的，但 fa 成员函数的实现必须放在类 B 的声明之后，否则会出现不能识别类 B 的错误。

3.3.2 面试真题解析

【面试题 3-10】两个互相独立的类 A 和 B 各自定义了非静态公有成员函数 PublicFunc()和非静态的私有成员函数 PrivateFunc()，现在要在类 A 中增加一个成员函数 A::AdditionalPunction(A a,B b)，则可以在其实现部分（函数体内部）出现的合法的表达式最全的是（　　）。

A．a.PrivateFunc()、a.PublicFunc()、b.PrivateFunc()、b.PublicFunc()

B．a.PrivateFunc()、a.PublicFunc()、b.PublicFunc()

C．a.PrivateFunc()、b.PrivateFunc()、b.PublicFunc()

D．a.PublicFunc()、b.PublicFunc()

答：在类 A 的成员函数 AdditionalPunction 中可以调用类 A 的任何成员函数，即 a.PrivateFunc()、a.PublicFunc()是合法的，也可以调用子对象 b 的公有成员函数，即 b.PublicFunc()是合法的，但不能调用子对象 b 的私有成员函数，即 b.PrivateFunc()是不合法的。答案为 B。

【面试题 3-11】以下代码执行时可能得到的结果是（　　）。

```
class A
{
    int i;
};
class B
{   A *p;
public:
    B() {  p=new A; }
    ~B() { delete p; }
};
void sayHello(B b)
{  }
void main()
{   B b;
    sayHello(b);
}
```

A．程序正常运行　　B．程序编译错误　　C．程序崩溃　　D．程序死循环

答：在 main 中定义类 B 的对象 b，调用类 B 的构造函数，为子对象指针 p 创建指向的实例。当执行 sayHello(b)时将实参 b 赋给形参 b，采用的是浅复制，形参 b 和实参 b 的 p 都指向相同的实例，sayHello 函数执行完毕，销毁形参 b 调用析构函数将 p 指向的实例释放。当 main 执行完毕，销毁实参 b 调用析构函数释放 p 指向的实例（前面已经被释放）时出现程序崩溃。答案为 C。

3.4　嵌套类和局部类

3.4.1　要点归纳

⋙ 1．嵌套类

用户可以在一个类声明中声明其他类，这种在其他类中声明的类称为**嵌套类**，包含嵌套类的类称为**外围类**。其层次结构如下：

```
class OuterClass        //称为外围类
{
public:
    …
private:
```

```
    class InnerClass    //称为嵌套类
    {
        …
    }
    …
};
```

📖 有关嵌套类和外围类的说明如下：

❶ 嵌套类可以是公有的，也可以是私有的。如果嵌套类是私有的，那么嵌套类就不能在其外围类之外使用；如果嵌套类是公有的，那么嵌套类可以在外围类之外使用，但此时嵌套类的类型应该是 OuterClass::InnerClass。

❷ 嵌套类可以通过定义引用、指针或者对象来访问外围类的成员，不管该成员是 public、private 还是 protected。

❸ 嵌套类仅是语法上的嵌入，嵌套类的成员不是外围类中对象的成员，反之亦然。嵌套类的成员函数对外围类的成员没有访问权，反之亦然。

例如有以下程序：

```
#include <iostream.h>
#include <string.h>
class A                 //外围类
{   class B             //嵌套类
    {   int no;
        char name[10];
    public:
        void setb(int n,char na[])
        {   no=n;
            strcpy(name,na);
        }
        void dispb() {  cout << "no:" << no << ",name:" << name << endl; }
    };
    B b;            //类 A 的数据成员 b 是类 B 的对象
    public:
        void seta(int n,char na[]) {  b.setb(n,na); }
        void dispa() {  b.dispb(); }
};
void main()
{   A a;
    a.seta(2,"Mary");
    a.dispa();
}
```

在上述程序中声明了一个外围类 A，其中包含一个嵌套类，类 A 的私有数据成员是类 B 的对象。程序的执行结果如下：

```
no:2 name:Mary
```

⋙ 2. 局部类

一个类的声明还可以在某一个函数定义中，这种类称为**局部类**，局部类的声明限制在函数定义中。局部类不包含静态成员，并且所有成员函数都必须定义在类体内。通常局部类很少使用。

例如有以下程序：

```
#include <iostream.h>
#include <string.h>
void fun(int n,char na[])
{
    class B                        //局部类
    {   int no;
        char name[10];
    public:
        B(int n,char na[])         //局部类构造函数
        {   no=n;
            strcpy(name,na);
        }
        void dispb() {  cout << "no:" << no << ",name:" << name << endl; }
    };
    B b(n,na);                     //定义局部类的对象
    b.dispb();                     //调用局部类的成员函数
}
void main()
{
    fun(2,"Mary");
}
```

在上述程序中定义了一个函数 fun，其中声明了一个局部类 B，并定义了类 B 的一个对象 b，通过对象 b 调用相应的成员函数。程序的执行结果如下：

```
no:2 name:Mary
```

3.4.2　面试真题解析

【面试题 3-12】说明 C++中局部类和嵌套类的区别。

答：在一个函数体内定义的类称为局部类。在局部类中只能使用它的外围作用域中的对象和函数进行联系，因为外围作用域中的变量与该局部类的对象无关。局部类的所有成员都必须定义在类体内。

在一个类中声明的类称为嵌套类，从作用域的角度看，嵌套类被隐藏在外围类之中，该类名只能在外围类中使用，如果在外围类的作用域内使用该类名，需要加名字限定。从访问权限的角度来看，嵌套类名与它的外围类的对象成员名具有相同的访问权限规则，不能访问嵌套类的对象中的私有成员函数，也不能对外围类的私有嵌套类建立对象。嵌套类中的成员函数可以在它的类体外定义。

【面试题 3-13】说明嵌套类的主要作用。

答：定义嵌套类的作用在于隐藏类名，减少全局的标识符，从而限制用户能否使用该类建立对象，这样可以提高类的抽象能力，并且强调了两个类（外围类和嵌套类）之间的主从关系。

【面试题 3-14】在软件开发中有一种单例模式，即在应用中仅允许创建类的一个实例。那么如何采用 C++实现单例模式？

答：单例模式也称为单件模式，设计它的意图是保证一个类仅有一个实例，并提供类的一个全局访问点，该实例被所有程序模块共享。

说明：由于在 VC++ 6.0 中嵌套类不能访问外围类的静态私有数据成员，而在 Dev C++中可以，本面试题的程序是在 Dev C++中调试执行的。

采用 C++实现单例模式有多种方式，这里采用嵌套类实现，其程序如下：

```
#include <iostream>
using namespace std;
class A                          //单例模式类 A
{
    //其他数据成员
public:
    static A *GetInstance()
    {   if(pinstance==NULL)
            pinstance=new A();
        return pinstance;        //返回唯一实例的指针
    }
private:
    static A *pinstance;         //指向类 A 实例的指针
    class B                      //嵌套类，它的唯一工作就是在析构函数中释放实例
```

```
    {
    public:
        ~B()
        {   if(A::pinstance!=NULL) //访问外围类的静态私有数据成员
                delete A::pinstance;//在 VC++ 6.0 中不允许，在 Dev C++中可以
            cout << "释放实例" <<endl;
        }
    };
    static B b;                    //定义一个子对象，在程序结束时会调用它的析构函数
};
A *A::pinstance=NULL;              //静态子对象指针初始化
A::B A::b;                         //静态子对象初始化
int main()
{   A *p=A::GetInstance();
    A *q=A::GetInstance();
    if (p==q) cout << "Same" << endl;
    return 0;
}
```

其中类 A 实现单例模式，对其说明如下：

- 它的构造函数是私有的，这样就不能从别处创建该类的实例。
- 它有一个唯一实例的静态对象指针 pinstance，且是私有的。
- 它有一个公有成员函数 GetInstance，可以获取这个唯一的实例，并在需要的时候创建该实例。
- 设计嵌套类的目的是为了定义它的静态子对象，在程序结束时会调用该子对象的析构函数以释放唯一的实例。如果采用在类 A 中设计析构函数来释放实例，则该析构函数必须是公用的，这在单例模式中是不恰当的。

上述程序的设计特征如下：

- 在单例类 A 的内部声明专门的私有嵌套类。
- 在嵌套类 B 内定义专门用于释放实例的析构函数。
- 利用程序在结束时销毁静态子对象的特性选择最终的释放时机。
- 使用 C++单例模式的代码不需要其他操作，不必关心对象的释放。

上述程序的执行结果如下：

```
Same
释放实例
```

从结果可以看出类 A 的两个对象指针指向同一个实例，说明只能创建唯一的实例，并且该唯一实例是自动销毁的。

3.5 自测题和参考答案

3.5.1 自测题

3-1 说明 explicit 关键字的含义和作用。

3-2 如果一个类的成员函数 print()不修改类的数据成员值，应将其声明为（　　）。

A．void print() const;　　B．const void print();

C．void const print();　　D．void print(const);

3-3 以下关于嵌套类的叙述中正确的是（　　）。

A．嵌套类可以像普通类那样在外围类之外使用，例如定义其对象

B．嵌套类的所有成员可以被看成是外围类的成员

C．嵌套类中的成员函数可以直接访问外围类的成员

D．以上都不对

3-4 阅读下面类的定义，找出程序中的错误，并说明错误原因。

```
#include <iostream.h>
class Sample
{   int x,y,c;
public:
    Sample(int i=0,int j=0);
    void f() const;
};
void Sample::f() const
{   cout << x << " " << y << endl;
    c++;
}
```

3-5 指出以下程序中存在的问题。

```
#include <iostream>
using namespace std;
class A
{   int *p;
    int size;
public:
    A():size(4),p(new int[size])
```

```
    {   for(int i=0;i<size;++i)
            p[i]=i;
    }
    void display()
    {   for(int i=0;i<size;i++)
            cout << p[i] << " ";
        cout << endl;
    }
};
int main()
{   A a;
    a.display();
    return 1;
}
```

3-6　以下 C++代码的输出是什么？

```
#include <iostream.h>
class A
{
private:
    int n1;
    int n2;
public:
    A(): n2(0),n1(n2+2) { }
    void Print() {  cout << "n1:" << n1 << ",n2:" << n2 << endl; }
};
void main()
{   A a;
    a.Print();
}
```

3-7　给出以下程序的执行结果。

```
#include <iostream.h>
class A
{
public:
    int n;
    A(int i) { cout << "A 构造函数" << endl; n=i; }
    ~A() {  cout << "A 析构函数" << endl; }
    void dispa() { cout << "n=" << n << endl; }
};
```

```
class B
{   A a,b;
    int m;
public:
    B(int i,int j):m(j),a(i),b(2*a.n) { cout << "B 构造函数" << endl; }
    void dispb()
    {   a.dispa();
        b.dispa();
        cout << "m=" << m << endl;
    }
    ~B() {  cout << "B 析构函数" << endl; }
};
void main()
{   B obj(1,3);
    obj.dispb();
}
```

3-8　给出以下程序的执行结果。

```
#include <iostream.h>
class A
{
private:
    int n;
public:
    A(int n)
    {   this->n=n;
        cout << "A constructor" << endl;
    }
    void show() {  cout << n << endl; }
    class B
    {
    public:
        B() {  cout << "B constructor" << endl; }
        void disp() {  cout << "B disp" << endl; }
    };
    B b;
};
void main()
{   A a=1;
    a.show();
    a.b.disp();
```

```
}
```

3-9　下面的代码实现了设计模式中的（　　）模式。

```
class A
{   A *pinstance;
    A() { }
public:
    static A *GetInstance()
    {   if (pinstance==NULL)
            pinstance=new A();
        return pinstance;
    }
    …
};
```

A．Factory　　B．Abstract Factory　　C．Singleton　　D．Builder

3.5.2　参考答案

3-1　**答**：explicit 关键字禁止单参数构造函数的隐式转换，防止程序员误操作。

3-2　**答**：将其声明为 const 成员函数。答案为 A。

3-3　**答**：只有公有嵌套类才可以在外围类之外使用。嵌套类的成员并不是外围类的成员。嵌套类可以通过定义引用、指针或者对象来访问外围类的成员。答案为 D。

3-4　**答**：类成员函数 Sample()被定义为常成员函数，因此它不能修改对象的数据成员，即 C++是不适合的。修改方法是将 Sample()常成员函数改为普通成员函数。

3-5　**答**：在程序中有两个问题，一是类 A 中私有数据成员的定义顺序是先 p 后 size，在调用构造函数时先执行成员初始化列表中的 p(new int[size])后执行 size(4)，前者执行时 size 为垃圾值，分配空间失败（尽管在 VC++ 6.0 和 Dev C++中测试没有问题，原则上讲这样是错误的），应该在类中将定义 p 和 size 的顺序倒过来；二是 p 指向的空间没有释放，应该设计一个含 delete p 语句的析构函数。

3-6　**答**：在 C++中类数据成员的初始化顺序与其定义顺序相同，而与它们在构造函数的成员初始化列表中的顺序无关，因此首先初始化 n1，但初始 n1 的实参 n2 还没有初始化，是一个垃圾值，所以 n1 就是一个垃圾值；在初始化 n2 时根据参数 0 对其初始化，故 n2=0，所以程序的输出结果是 n1 为一个垃圾值，n2 为 0。

3-7　**答**：类 B 有类 A 的两个子对象 a、b 作为数据成员，a 在前 b 在后，先后调用类 A 的构造函数创建它们，再执行类 B 的构造函数体。析构函数调用的顺序与之相反。程序的执行结果如下：

```
A 构造函数
A 构造函数
B 构造函数
n=1
n=2
m=3
B 析构函数
A 析构函数
A 析构函数
```

3-8　**答**：类 B 是类 A 的嵌套类，在本程序中仅用于定义类 A 的子对象 b，和将类 B 的定义放在类 A 的外面是相同的。在创建类 A 的对象 a 时先调用子对象的构造函数，再执行类 B 的构造函数体。程序的输出结果如下：

```
B constructor
A constructor
1
B disp
```

3-9　**答**：类 A 的构造函数是私有的，有唯一的对象指针 pinstance，且是私有的，有一个公有成员函数 GetInstance，可以获取这个唯一的实例。它属于典型的单例模式。答案为 C。