C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

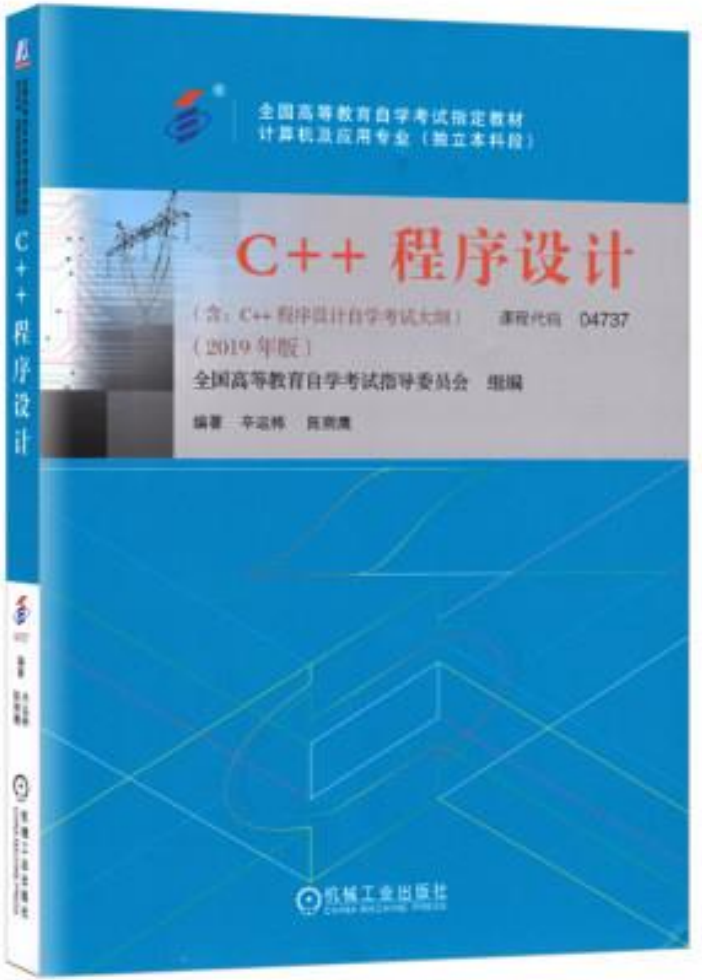
教材介绍

C++程序设计

（2019年版）

编著：辛运帷 陈朔鹰

机械工业出版社



考试题型

单选题　　1分×20题 = 20分

填空题　　1分×15题 = 15分

程序填空题　4分×5题 = 20分

程序分析题　6分×5题 = 30分

程序设计题　　2题 = 15分

（第一题5分，第二题10分）

第三章　类和对象进阶

本章主要内容

- 构造函数
- 析构函数
- 类的静态成员
- 变量及对象的生存期和作用域
- 常量成员和常引用成员
- 成员对象和封闭类
- 友元
- this指针

3.1 构造函数

对于C++中基本数据类型的变量，可以声明全局变量和函数内部的局部变量。

变量	初始化
全局变量	如果程序员在声明变量时没有进行初始化，则系统自动为其初始化为0。这个工作在程序启动时完成。
局部变量	系统不进行自动初始化，所以它的初值需要靠程序员给定。如果程序员没有设定，则是一个随机值。

3.1 构造函数

- 为了对对象进行初始化，C++提供了一种称为构造函数的机制，用于对对象进行初始化，实际上是用来为成员变量赋初值的。
- 构造函数是类中的特殊成员函数，它属于类的一部分。给出类定义时，由程序员编写构造函数。如果程序员没有编写类的任何构造函数，则由系统自动添加一个不带参数的构造函数。
- 声明对象后，可以使用new运算符为对象进行初始化，此时调用的是对象所属类的构造函数。构造函数的作用是完成对象的初始化工作，用来保证对象的初始状态是确定的。在对象生成时，系统自动调用构造函数，用户在程序中不会直接调用构造函数。

3.1.2 构造函数的定义

定义一个类时，需要为类定义相应的构造函数。构造函数的函数名与类名相同，**没有返回值**。一个类的构造函数可以有多个，即**构造函数允许重载**。同一个类的多个构造函数的参数表一定不能完全相同。

构造函数的声明格式如下：

类名(形参1, 形参2, …， 形参n);

3.1.2 构造函数的定义

- 在声明类的构造函数时可以同时给出函数体，这样的构造函数称为内联函数。也可以在类体外给出构造函数的定义。构造函数的声明中，形参的个数可以为0，即参数表为空。
- 当类中没有定义任何构造函数时，系统会自动添加一个参数表为空、函数体也为空的构造函数，称为默认构造函数。所以**任何类都可以保证至少有一个构造函数**。
- 如果程序员在程序中已经定义了构造函数，则系统不会再添加默认构造函数。

假设类的成员变量是x1，x2，…，xn，则在类体外定义构造函数时通常有如下3种形式：

形式一：

```
类名::类名(形参1,形参2,…,形参n):x1(形参1), x2(形参2), …, xn(形参n){}
```

形式二：

```
类名::类名(形参1,形参2,…,形参n)
{
    x1=形参1;
    x2=形参2;
    ……
    xn=形参n;
}
```

形式三：

```
类名::类名( )
{
    x1=初始化表达式1;
    x2=初始化表达式2;
    ……
    xn=初始化表达式n;
}
```

例如，程序2-1中为类myDate定义了两个构造函数，分别如下：

```
myDate::myDate( )
{
    year = 1970;
    month = 1;
    day = 1;
}
myDate::myDate(int y, int m, int d)
{
    year = y;
    month = m;
    day = d;
}
```

例3-1构造函数的定义

```
myDate::myDate():year(1970)，month(1)，day(1){ }
```

或是

```
myDate::myDate(int y, int m, int d):year(y), month(m), day(d){}
```

第一个是使用固定值在初始化列表中为各成员变量赋初值，第二个是使用带入的参数值通过初始化列表为各成员变量赋初值。

构造函数也可以在类体外定义。假定类myDate中已经声明了下列4个构造函数:

```
myDate( );                                  //不带参数
myDate(int);                                //带1个参数
myDate(int, int);                           //带2个参数
myDate(int, int, int);                      //带3个参数
```

则类体外，构造函数的定义如例3-2所示。

例3-2在类体外定义构造函数

```
myDate::myDate():year(1970),month(1),day(25){ }     //不带参数
myDate::myDate(int d):year(1970),month(1)           //带1个参数
{       day = d;
}
myDate::myDate(int m, int d):year(1970)             //带2个参数
{       month = m;
        day = d;
}
myDate::myDate(int y, int m, int d)                 //带3个参数
{       year = y;
        month = m;
        day = d;
}
```

例3-4单项选择题

构造函数不具备的特征是【　】

A.构造函数的函数名与类名相同

B.构造函数可以重载

C.构造函数可以设置默认参数

D.构造函数必须指定返回类型

例3-4单项选择题

构造函数不具备的特征是【　】

A.构造函数的函数名与类名相同　　B.构造函数可以重载

C.构造函数可以设置默认参数　　D.构造函数必须指定返回类型

答案：D。

【分析】构造函数的函数名与类名相同（选项A正确），没有返回类型（选项D错误），可以像普通函数那样在参数表中使用默认参数（选项C正确）。构造函数可以重载，只要参数列表不同即可（选项B正确）。

例3-5单项选择题

以下选项中，自动调用类的构造函数的时机是【　】

A.定义类的成员函数时

B.定义类的对象时

C.定义类的成员对象时

D.定义类的友元函数时

例3-5单项选择题

以下选项中，自动调用类的构造函数的时机是【　】

A.定义类的成员函数时　　B.定义类的对象时

C.定义类的成员对象时　　D.定义类的友元函数时

答案：B。

【分析】定义类的成员函数、成员对象及友元函数时，均不调用类的构造函数。仅当定义类的对象时，才由系统自动调用类的构造函数。

真题演练

声明一个没有初始化参数的对象，需调用（　）。

A:指定参数构造函数

B:拷贝构造函数

C:初始化函数

D:默认构造函数

真题演练

声明一个没有初始化参数的对象，需调用（　）。

A:指定参数构造函数

B:拷贝构造函数

C:初始化函数

D:默认构造函数

答案：D

3.1.3 构造函数的使用

C++语言规定，创建类的任何对象时都一定会调用构造函数进行初始化。对象需要占据内存空间，生成对象时，为对象分配的这段内存空间的初始化由构造函数完成。

（定义对象时，类的构造函数会被自动调用）

特别地，如果程序中声明了对象数组，即数组的每个元素都是一个对象，则一定要为对象所属的这个类定义一个无参的构造函数。因为数组中每个元素都需要调用无参的构造函数进行初始化，所以必须要有一个不带参数的构造函数。

例3-7使用构造函数的默认参数创建对象

假设myDate类中仅定义了如下的构造函数：

```
myDate::myDate(int y =1970,int m =2,int d =14)  //3个参数均有默认值
{
        year = y;
        month = m;
        day = d;
}
```

则创建对象时，可以使用下列形式：

```
myDate d0;
myDate d1(1980);
myDate d2(1990,3);
myDate d3(2000,4,18);
```

输出这4个对象的值，如下所示:

```
1970/2/14
1980/2/14
1990/3/14
2000/4/18
```

例3-8单项选择题假定一个类的构造函数如下：

```
A(int k = 4, int j = 0)
{      a = k;
       b = j;
}
```

则执行“A x(1);”语句后，x.a和x.b的值分别是【　】

A. 1和0　　B. 1和4　　C. 4和0　　D. 4和1

例3-8单项选择题假定一个类的构造函数如下：

```
A(int k = 4, int j = 0)
{      a = k;
       b = j;
}
```

则执行“A x(1);”语句后，x.a和x.b的值分别是【　】

A. 1和0　　B. 1和4　　C. 4和0　　D. 4和1

答案：A。

【分析】调用函数时，给定的实参自左至右与形参进行一一匹配。如果实参的个数少于形参的个数，则不足的形参使用默认值进行初始化。这条规则也适用于构造函数的调用情况。本题中，代入的实参1赋给参数k，参数j的值为默认值0。

例3-9使用构造函数创建对象指针

仍假设已有例3-2中定义的构造函数，则创建对象时，可以使用下列形式：

```
myDate *pd=new myDate( );                    pd->printDate( );
myDate *pd1=new myDate(55);                  pd1->printDate( );
myDate *pd2=new myDate(66,77);               pd2->printDate( );
myDate *pd3=new myDate(12,30,50);            pd3->printDate( );
```

使用new创建对象时，还可以写成：

```
myDate *pd = new myDate;                     //未加括号
```

如果程序中声明了对象数组，即数组的每个元素都是一个对象，则一定要为对象所属的这个类定义一个无参的构造函数。因为数组中每个元素都需要调用无参的构造函数进行初始化，所以必须要有一个不带参数的构造函数。

仍以类myDate为例，如声明了对象数组A，即

myDate A[3];

此时系统要调用无参的构造函数，为数组A的3个元素进行初始化。

也有特例的情况。如果声明数组A时同时给各元素赋了初值，例如，有下列语句:

myDate A[3]={myDate(1), myDate(10,25), myDate(1980,9,10)};

在构造函数中，需要为类中声明的所有的成员变量赋初值。对于基本数据类型的成员变量，如果程序中没有进行显式的初始化，则系统使用0进行初始化。

例3-10单项选择题

AB是一个类，那么执行语句

AB a(4), b[3], *p;

时，调用的构造函数的次数是【】

A.2　　B.3　　C.4　　D.5

在构造函数中，需要为类中声明的所有的成员变量赋初值。对于基本数据类型的成员变量，如果程序中没有进行显式的初始化，则系统使用0进行初始化。

例3-10单项选择题

AB是一个类，那么执行语句

AB a(4), b[3], *p;

时，调用的构造函数的次数是【】

A.2　　B.3　　C.4　　D.5

答案：C。

【分析】创建对象a时，调用一次构造函数。创建对象b时，因为这是一个含有3个元素的数组，所以需要调用3次构造函数。而对于指针p，仅是说明了这个指针，并未与对象相关，所以并不调用构造函数。

课堂练习

My是一个类，则执行语句My a[3], *p[2];之后，自动调用构造函数次数为（ ）。

A:2

B:3

C:4

D:5

课堂练习

My是一个类，则执行语句My a[3], *p[2];之后，自动调用构造函数次数为（ ）。

A:2

B:3

C:4

D:5

答案：B

3.1.4 复制构造函数与类型转换构造函数

复制构造函数是构造函数的一种，也称为**拷贝构造函数**。它的作用是使用一个已存在的对象去初始化另一个正在创建的对象。例如，类对象间的赋值是由复制构造函数实现的。

复制构造函数只有一个参数，参数类型是**本类的引用**。复制构造函数的参数可以是const引用，也可以是非const引用。一个类中可以写两个复制构造函数，一个函数的参数是const引用，另一个函数的参数是非const引用。这样，当调用复制构造函数时，既能以常量对象（初始化后值不能改变的对象）作为参数，也能以非常量对象作为参数去初始化其他对象。对于类A而言，复制构造函数的原型如下（格式一）：

A::A(const A&)

或是如下（格式二）

A::A(A &)

设有程序2-1和程序2-2中定义的类myDate和类Student,则主函数中可以有如下的语句:

```
Student stud;
Student ss[2] = {stud, Student()};
```

第二条语句也可以写为如下的三条语句:

```
Student ss[2];
ss[0] = Student(stud);
ss[1] = Student();
```

创建数组ss时，具体来说，是创建ss[0]中的对象时，用到了默认复制构造函数。

可以在类中定义自己的复制构造函数。

例3-14自定义复制构造函数在类Student中声明复制构造函数的原型:

```
Student(const Student &s);
```

复制构造函数的函数体如下:

```
Student::Student(const Student &s)
{
    name = s.name;
    birthday = s.birthday;
}
```

则在主函数中调用自定义的复制构造函数的语句如下:

```
Student ss[2] = {stud, Student()};
stud.printStudent();
```

自动调用复制构造函数的情况有以下3种：

1)当用一个对象去初始化本类的另一个对象时，会调用复制构造函数。例如，使用下列形式的说明语句时，即会调用复制构造函数。

类名 对象名2(对象名1);

类名 对象名2=对象名1;

2)如果函数F的参数是类A的对象，那么当调用F时，会调用类A的复制构造函数。换句话说，作为形参的对象，是用复制构造函数初始化的，而且调用复制构造函数时的参数，就是调用函数时所给的实参。

3)如果函数的返回值是类A的对象，那么当函数返回时，会调用类A的复制构造函数。也就是说，作为函数返回值的对象是用复制构造函数初始化的，而调用复制构造函数时的实参，就是retrun语句所返回的对象。

注意，在复制构造函数的参数表中，加上const是更好的做法。这样复制构造函数才能接收常量对象作为参数，即才能以常量对象作为参数去初始化别的对象。

课堂练习

假定一个类的构造函数为A(int aa,int bb){a=aa++;b=a*++bb;}，则执行Ax(4,5);语句后，x.a和x.b的值分别为（ ）。

A:4和5

B:4和20

C:4和24

D:20和5

课堂练习

假定一个类的构造函数为A(int aa,int bb){a=aa++;b=a*++bb;}，则执行Ax(4,5);语句后，x.a和x.b的值分别为（ ）。

A:4和5

B:4和20

C:4和24

D:20和5

答案：C

课堂练习

以下选项中，自动调用类的构造函数的时机是（　）。

A:定义类的成员函数时

B:定义类的对象时

C:定义类的成员对象时

D:定义类的友元函数时

课堂练习

以下选项中，自动调用类的构造函数的时机是（　）。

A:定义类的成员函数时

B:定义类的对象时

C:定义类的成员对象时

D:定义类的友元函数时

答案：B

课堂练习

假定一个类的构造函数如下：

```
A(int k = 4, int j = 0)
{   a = k;
    b = j;
}
```

则执行“A x(1);”语句后，x.a和x.b的值分别是（　）

A:1和0

B:1和4

C:4和0

D:4和1

课堂练习

假定一个类的构造函数如下：

```
A(int k = 4, int j = 0)
{   a = k;
    b = j;
}
```

则执行“A x(1);”语句后，x.a和x.b的值分别是（　）

A:1和0

B:1和4

C:4和0

D:4和1

答案：A

真题演练

下列说法中，有关构造函数是正确的是（　）。

A:任何一个类必定有构造函数

B:可定义没有构造函数的类

C:构造函数不能重载

D:任何一类必定有缺省的构造函数

真题演练

下列说法中，有关构造函数是正确的是（　）。

A:任何一个类必定有构造函数

B:可定义没有构造函数的类

C:构造函数不能重载

D:任何一类必定有缺省的构造函数

答案：A

真题演练

对于拷贝构造函数和赋值操作的关系，正确的描述是（　）。

A:拷贝构造函数和赋值操作是完全一样的操作

B:进行赋值操作时，不会调用类的构造函数

C:当调用拷贝构造函数时，类的对象正在被建立并被初始化

D:拷贝构造函数和赋值操作不能在同一个类中被同时定义

真题演练

对于拷贝构造函数和赋值操作的关系，正确的描述是（　）。

A:拷贝构造函数和赋值操作是完全一样的操作

B:进行赋值操作时，不会调用类的构造函数

C:当调用拷贝构造函数时，类的对象正在被建立并被初始化

D:拷贝构造函数和赋值操作不能在同一个类中被同时定义

答案：C

真题演练

类构造函数定义的位置是（　）。

A:类体内或体外

B:只是在类体内

C:只在类体外

D:在类的成员函数中

真题演练

类构造函数定义的位置是（　）。

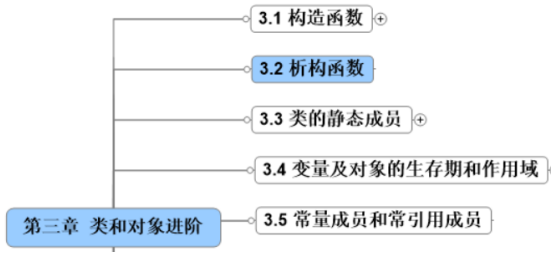
A:类体内或体外

B:只是在类体内

C:只在类体外

D:在类的成员函数中

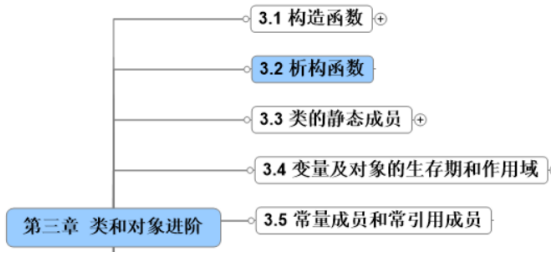
答案：A



3.2 析构函数

- 与构造函数一样，析构函数也是成员函数的一种，它的名字也与类名相同，但要在类名前面加一个“~”字符，以区别于构造函数。析构函数没有参数，也没有返回值。一个类中有且仅有一个析构函数，如果程序中没有定义析构函数，则编译器自动生成默认的析构函数。析构函数不可以多于一个，不会有重载的析构函数。默认析构函数的函数体为空。
- 创建对象时自动调用构造函数，那么，什么时候调用析构函数呢？可想而知，在对象消亡时自动调用析构函数。析构函数的作用是做一些善后处理的工作。例如，如果在创建对象时使用new运算符动态分配了内存空间，则在析构函数中应该使用delete释放掉这部分占用的空间，保证空间可再利用。

3.2 析构函数



当使用new运算符生成对象指针时，自动调用本类的构造函数。使用delete删除这个对象时，首先为这个动态对象调用本类的析构函数，然后再释放这个动态对象占用的内存。

例3-16单项选择题

下面对析构函数的叙述中，正确的是【】

A.系统在任何情况下都能正确析构对象

B.用户必须定义类的析构函数

C.析构函数没有参数，也没有返回值

D.析构函数可以设置默认参数

例3-16单项选择题

下面对析构函数的叙述中，正确的是【】

A.系统在任何情况下都能正确析构对象

B.用户必须定义类的析构函数

C.析构函数没有参数，也没有返回值

D.析构函数可以设置默认参数

答案：C。

【分析】通常析构函数用来在对象消亡时析构对象。但如果程序中析构函数编写不正确，或是出现其他问题，则对象不能正确析构。选项A是错误的。如果程序中没有定义析构函数，则系统会自动添加一个默认析构函数。选项B是错误的。析构函数中没有参数，所以也不可以设置默认参数。选项D是错误的。

例18使用delete语句

现在修改类myDate和类Student的无参构造函数和析构函数如下：

```
myDate::myDate(): year(1970), month(1), day(10)
{     cout<<”myDate 构造函数"<<endl;
}
myDate::~myDate()
{     cout<<"myDate 析构函数"<<endl;
}
Student::Student() : name("Noname"), birthday(myDate())
{     cout<<"Student 构造函数"<<endl;
}
Student: :~Student()
{     cout<<"Stuclent 析构函数"<<endl;
}
```

如果主函数中执行以下语句：

```
Student *stud = new Student();
delete stud;
```

则得到的信息如下:

myDate构造函数
Student构造函数
Student析构函数
myDate析构函数

对于对象数组，要为它的每个元素调用一次构造函数和析构函数。全局对象数组的析构函数在程序结束之前被调用。

例3-19对象数组与delete语句

```
Student *ss = new Student[2];
delete [ ]ss;
```

表达式new Student[2]首先分配2个Student类的对象所需的内存，然后为这2个对象各调用一次构造函数。当使用delete释放动态对象数组时，通过“[ ]”告诉编译器ss是对象数组，所以也为这2个对象各调用一次析构函数。

执行这两行语句得到的显示信息如下：

```
my Date构造函数
Student构造函数
myDate构造函数
Student构造函数
Student析构函数
my Date析构函数
Student析构函数
myDate析构函数
```

下面的语句创建了对象指针数组，消亡时，要分别释放空间。

```
Student *ss[2] = {new Student(), new Student()};
delete ss[0];
delete ss[1];
```

析构函数的调用执行顺序与构造函数刚好相反。

将下列程序补充完整。

```
#include <iostream>
using namespace std;
class Samp
{
public:
    void Setij(int a, int b)
    {
            i = a;
            j = b;
    }
    ~Samp()
   {       cout<<"析构.."<<i<<endl;
   }
   int GetMuti()
   {       return i*j;
   }
protected:
   int i;
   int j;
};
```

```
int main()
{
    Samp *p;
    p = new Samp[5];
    if(!p)
    {    cout<<"内存分配错误\n";
        return 1;
    }
    for(int j = 0;j<5;j++)
        p[j].Setij(j,j);
    for(int k = 0; k < 5; k++)
        cout<<"Muti["<<k<<"]值是:"<<p[k]._____<<endl;
    _____;
    return 0;
}
```

将下列程序补充完整。

```
#include <iostream>
using namespace std;
class Samp
{
public:
    void Setij(int a, int b)
    {
            i = a;
            j = b;
    }
    ~Samp()
   {       cout<<"析构.."<<i<<endl;
   }
   int GetMuti()
   {       return i*j;
   }
protected:
   int i;
   int j;
};
int main()
{
    Samp *p;
    p = new Samp[5];
    if(!p)
    {    cout<<"内存分配错误\n";
        return 1;
    }
    for(int j = 0;j<5;j++)
        p[j].Setij(j,j);
    for(int k = 0; k < 5; k++)
        cout<<"Muti["<<k<<"]值是:"<<p[k]. GetMuti() <<endl;
    delete [ ] p;
    return 0;
}
```

课堂练习

假定有类AB，有相应的构造函数定义，能正确执行"AB a(4)，b(5)，c[3]，*p[2]={&a，&b}；"语句，请问执行完此语句后共调用该类析构函数的次数为（ ）。

A:14

B:5

C:3

D:1

课堂练习

假定有类AB，有相应的构造函数定义，能正确执行"AB a(4)，b(5)，c[3]，*p[2]={&a，&b}；"语句，请问执行完此语句后共调用该类析构函数的次数为（ ）。

A:14

B:5

C:3

D:1

答案：B

真题演练

下面对析构函数的正确描述是（　）。

A:系统不能提供默认的析构函数

B:析构函数必须由用户定义

C:析构函数没有参数

D:析构函数可以设置默认参数

真题演练

下面对析构函数的正确描述是（　）。

A:系统不能提供默认的析构函数

B:析构函数必须由用户定义

C:析构函数没有参数

D:析构函数可以设置默认参数

答案：C

析构函数的参数0个，个数1个，不能指定返回类型，哪怕是void都不行。

真题演练

使用delete[ ]删除对象数组时，描述正确的是（　）。

A:数组中各元素都调用析构函数

B:数组中各元素都调用构造函数

C:不调用析构函数

D:只有首元素调用析构函数

真题演练

使用delete[ ]删除对象数组时，描述正确的是（　）。

A:数组中各元素都调用析构函数

B:数组中各元素都调用构造函数

C:不调用析构函数

D:只有首元素调用析构函数

答案：A

真题演练

析构函数的参数个数为（ ）。

A:0个

B:1个

C:至少1个

D:多于1个

真题演练

析构函数的参数个数为（ ）。

A:0个

B:1个

C:至少1个

D:多于1个

答案：A

真题演练

定义析构函数时，应该注意（　）。

A:其名与类名完全相同

B:返回类型是void类型

C:无形参，也不可重载

D:函数体中必须有delete语句

真题演练

定义析构函数时，应该注意（　）。

A:其名与类名完全相同

B:返回类型是void类型

C:无形参，也不可重载

D:函数体中必须有delete语句

答案：C

3.3 类的静态成员

- 与C语言一样，可以使用static说明自动变量。根据定义的位置不同，分为静态全局变量和静态局部变量。
- 全局变量是指在所有花括号之外声明的变量，其作用域范围是全局可见的，即在整个项目文件内都有效。使用static修饰的全局变量是静态全局变量，其作用域有所限制，仅在定义该变量的源文件内有效，项目中的其他源文件中不能使用它。
- 块内定义的变量是局部变量，从定义之处开始到本块结束处为止是局部变量的作用域。使用static修饰的局部变量是静态局部变量，即定义在块中的静态变量。静态局部变量具有局部作用域，但却具有全局生存期。
- 静态局部变量具有局部作用域，但却具有全局生存期。也就是说，静态局部变量在程序的整个运行期间都存在，它占据的空间一直到程序结束时才释放，但仅在定义它的块中有效，在块外并不能访问它。
- 静态变量均存储在全局数据区，静态局部变量只执行一次初始化。如果程序未显式给出初始值，则相当于初始化为0；如果显式给出初始值，则在该静态变量所在块第一次执行时完成初始化。

3.3 类的静态成员

例3-22单项选择题

在函数中声明的静态变量【 】

A.在函数体中可见，函数执行结束时释放占用的空间

B.在函数体中可见，程序结束时释放占用的空间

C.在程序中可见，函数执行结束时释放占用的空间

D.在程序中可见，程序结束时释放占用的空间

3.3 类的静态成员

例3-22单项选择题

在函数中声明的静态变量【 】

A.在函数体中可见，函数执行结束时释放占用的空间

B.在函数体中可见，程序结束时释放占用的空间

C.在程序中可见，函数执行结束时释放占用的空间

D.在程序中可见，程序结束时释放占用的空间

答案：B。

【分析】函数中声明的静态变量一直到整个程序结束时才释放占用的空间，但它并不是在程序的所有位置都可见。函数中声明的静态变量仅在函数体中可见。

```
#include <iostream>
using namespace std;
static int glos=100;
void f( )
{
    int a=1;              //局部自动变量a
    static int fs=1;      //静态局部变量fs，完成初始化。
    cout<<"在f中:a(自动)="<<a<<"  fs( 静态)="<<fs<<"  glos(静态)="<<glos<<endl;
    a+=2;
    fs+=2;
    glos+=10;
    cout<<"在f中:a(自动)="<<a<<"  fs( 静态)="<<fs<<"  glos(静态)="<<glos<<endl<<endl;
   //cout<<“ms=”<<ms<<endl;    //该行错误，变量ms不可见。
}
int main( )
{
     static int ms =10;
     for(int i=1;i<=3;i++)  f();
     //cout<<“fs=”<<fs<<endl;     //该行错误，变量fs不可见。
     cout<<"ms="<<ms<<endl;
     cout<<"glos="<<glos<<endl;
     return 0;
}
```

3.3.2 类的静态成员

- 类的静态成员有两种：静态成员变量和静态成员函数。在类体内定义类的成员时，在前面添加static关键字后，该成员即成为静态成员。
- 类的静态成员被类的所有对象共享，不论有多少对象存在，静态成员都只有一份保存在公用内存中。对于静态成员变量，各对象看到的值是一样的。
- 定义类静态成员变量时，在类定义中声明静态成员变量，然后必须在类体外定义静态成员变量的初值。这个初值不能在类体内赋值。
- 给静态成员变量赋初值的格式如下：

  类型 类名::静态成员变量=初值;

- 注意，在类体外为静态成员变量赋初值时，前面不能加static关键字，以免和一般的静态变量相混淆。在类体外定义成员函数时，前面也不能加static关键字。

3.3.2 类的静态成员

访问静态成员时，成员前面既可以用类名作前缀，也可以使用对象名或对象指针作前缀。这与访问类成员时仅能使用对象名或对象指针作前缀是不同的。

访问类静态成员的一般格式如下：

类名::静态成员名

或是

对象名.静态成员名

或是

对象指针->静态成员名

类的静态成员函数没有this指针，**不能在静态成员函数内访问非静态的成员**，即通常情况下，类的静态成员函数只能处理类的静态成员变量。静态成员数内函也不能调用非静态成员函数。

3.3.2 类的静态成员

例3-23单项选择题

已知类A中的两个成员函数f1( )和f2( )，如果在f1( )中不能直接调用f2( )，则下列选项中，正确的是【 】

A.f1( )和f2( )都是静态函数　　B.f1( )不是静态函数，f2( )是静态函数

C.f1( )是静态函数，f2( )不是静态函数　　D.f1( )和f2( )都不是静态函数

3.3.2 类的静态成员

例3-23单项选择题

已知类A中的两个成员函数f1( )和f2( )，如果在f1( )中不能直接调用f2( )，则下列选项中，正确的是【 】

A.f1( )和f2( )都是静态函数　　B.f1( )不是静态函数，f2( )是静态函数

C.f1( )是静态函数，f2( )不是静态函数　D.f1( )和f2( )都不是静态函数

答案：C。

【分析】在通常情况下，除一种情况外，类的成员函数之间是允许互相调用的。具体来说，静态函数与静态函数之间、非静态函数与非静态函数之间是可以相互调用的，非静态成员函数内可以调用静态成员函数，但静态成员函数内不能调用非静态成员函数。

3.3.2 类的静态成员

例3-24单项选择题

下列关于静态成员变量的描述中，正确的是【 】

A.静态成员变量是类的所有对象所共有的

B.静态成员变量要在构造函数内初始化

C.类的每个对象有自己的静态成员变量

D.静态成员变量不能通过类的对象访问

3.3.2 类的静态成员

例3-24单项选择题

下列关于静态成员变量的描述中，正确的是【 】

A.静态成员变量是类的所有对象所共有的

B.静态成员变量要在构造函数内初始化

C.类的每个对象有自己的静态成员变量

D.静态成员变量不能通过类的对象访问

答案：A。

【分析】类的静态成员变量只有一个拷贝（选项C是错误的），是类的所有对象所共享的（选项A是正确的）。类的每个对象都可以访问静态成员变量，成员变量前面既可以用类名作前缀，也可以使用对象名或对象指针作前缀（选项D是错误的）。**静态成员变量的初始化是在类体外进行的，所以不是在构造函数内初始化的（选项B是错误的）**。

3.3.2 类的静态成员

对于普通成员变量，每个对象有各自的一份，而**静态成员变量只有一份，被同类所有对象共享。**普通成员函数一定是作用在某个对象上的，而静态成员函数并不具体作用在某个对象上。访问普通成员时，要通过“对象名.成员名”等方式，指明要访问的成员变量是属于哪个对象的，或要调用的成员函数作用于哪个对象；访问静态成员时，则可以通过“类名::成员名”的方式访问，不需要指明被访问的成员属于哪个对象或作用于哪个对象。因此，甚至可以在还没有任何对象生成时就访问一个类的静态成员。非静态成员的访问方式其实也适用于静态成员，也就是可以通过“对象名.成员名”的方式访问，效果和“类名::成员名”这种访问方式没有区别。

课堂练习

下列关于静态数据成员的特性叙述中，错误的是（　）

A:说明静态数据成员时，使用关键字static进行修饰

B:静态数据成员要在类外进行初始化

C:引用静态数据成员时，要在静态数据成员名前加<类名>和作用域运算符

D:静态数据成员是所有对象的共享成员

课堂练习

下列关于静态数据成员的特性叙述中，错误的是（　）

A:说明静态数据成员时，使用关键字static进行修饰

B:静态数据成员要在类外进行初始化

C:引用静态数据成员时，要在静态数据成员名前加<类名>和作用域运算符

D:静态数据成员是所有对象的共享成员

答案：D

课堂练习

下列关于对静态数据成员的描述中，正确的是（　　）

A:静态数据成员不能用public控制符修饰

B:静态数据成员可以直接用类名或者对象名来调用

C:静态数据成员不可以被类的对象调用

D:静态数据成员不能用private控制符修饰

课堂练习

下列关于对静态数据成员的描述中，正确的是（　　）

A:静态数据成员不能用public控制符修饰

B:静态数据成员可以直接用类名或者对象名来调用

C:静态数据成员不可以被类的对象调用

D:静态数据成员不能用private控制符修饰

答案：B

课堂练习

在函数中声明的静态变量（　）

A:在函数体中可见，函数执行结束时释放占用的空间

B:在函数体中可见，程序结束时释放占用的空间

C:在程序中可见，函数执行结束时释放占用的空间

D:在程序中可见，程序结束时释放占用的空间

课堂练习

在函数中声明的静态变量（　）

A:在函数体中可见，函数执行结束时释放占用的空间

B:在函数体中可见，程序结束时释放占用的空间

C:在程序中可见，函数执行结束时释放占用的空间

D:在程序中可见，程序结束时释放占用的空间

答案：B

真题演练

下列描述错误的是（　）。

A:在创建对象前，静态成员不存在

B:静态成员是类的成员

C:静态成员不能是虚函数

D:静态成员函数不能直接访问非静态成员

真题演练

下列描述错误的是（　）。

A:在创建对象前，静态成员不存在

B:静态成员是类的成员

C:静态成员不能是虚函数

D:静态成员函数不能直接访问非静态成员

答案：A

真题演练

下面对静态数据成员的描述中，正确的是（）

A:类的不同对象有不同的静态数据成员值

B:类的每个对象都有自己的静态数据成员

C:静态数据成员是类的所有对象共享的数据

D:静态数据成员不能通过类的对象调用

真题演练

下面对静态数据成员的描述中，正确的是（）

A:类的不同对象有不同的静态数据成员值

B:类的每个对象都有自己的静态数据成员

C:静态数据成员是类的所有对象共享的数据

D:静态数据成员不能通过类的对象调用

答案：C

```
class Test{
    static int x;           //声明静态数据成员
    int n;
public:
    Test( ){ }      //定义无参数的Test类的构造函数
    Test(int a,int b){x=a;n=b;}  //定义含两个参数的Test类的构造函数Test为内联函数
    static int func( ){return x;}  //定义静态成员函数func为内联函数
    static void sfunc(Test&r,int a){r.n=a;}  //定义静态成员函数sfunc为内联函数,函数以Test类的引用r和整形数a为参数
    int Getn( ){return n;}  //定义成员函数Getn为内联函数
};                  //类Test的声明结束
int Test::x=25;    //初始化静态数据成员
#include <iostream>
using namespace std;
void main( )
{
    cout<<Test::func( );    //x在对象产生之前就存在，输出"25"
    Test b,c;       //利用无参数的构造函数产生Test类的对象b和c
    b.sfunc(b,58);  //设置对象b的数据成员n，n值为58，r为b的引用
    cout<<" "<<b.Getn( );  //输出" 58"
    cout<<" "<<b.func( );  //x属于所有对象，输出" 25"
    cout<<" "<<c.func( );  //x属于所有对象，输出" 25"
    Test a(24,56);  //利用含两个参数的构造函数产生Test类的对象a，并将x的值改为24，给a的私有数据成员n赋值56
    cout<<" "<<a.func( )<<" "<<b.func( )<<" "<<c.func( )<<endl;
}
```

25 58 25 25 24 24 24

```
class  A{
        int  i,j;
        static int x,y;//定义静态成员
public:  A(int a=0,int b=0,int c=0, int d=0) {i=a;j=b;x=c;y=d;}
        void Show(){
            cout << "i="<<i<<'\t'<<"j="<<j<<'\t';
            cout << "x="<<x<<'\t'<<"y="<<y<<"\n";}
};
int A::x=0; //必须对静态成员作一次定义性说明
int A::y=0;
void  main(void ){
        A  a(2,3,4,5);
        a.Show();
        A  b(100,200,300,400);
        b.Show();
        a.Show();
}
```

a.x 和b.x在内存中占据一个空间

a.y 和b.y在内存中占据一个空间

```
i=2     j=3     x=4     y=5
i=100   j=200   x=300   y=400
i=2     j=3     x=300   y=400
```

```
#include<iostream.h>
class  A
{     int  i,j;
public:
      static int x,y;//定义静态成员
public:
      A(int a=0,int b=0,int c=0, int d=0) {i=a;j=b;x=c;y=d;}
      void Show( )
      {      cout << "i="<<i<<'\t'<<"j="<<j<<'\t';
             cout << "x="<<x<<'\t'<<"y="<<y<<"\n";}
};
int A::x=1000; //必须对静态成员作一次定义性说明
int A::y=1000;
void  main(void )
{    cout<<"A::x="<<A::x<<" A::y="<<A::y<<endl;
      A  a(2,3,4,5);
      a.Show();
      A  b(100,200,300,400);
      b.Show();
      a.Show();
      cout<<"A::x="<<A::x<<" A::y="<<A::y<<endl;
}
```

```
A::x=1000 A::y=1000
i=2     j=3     x=4     y=5
i=100   j=200   x=300   y=400
i=2     j=3     x=300   y=400
A::x=300 A::y=400
```

```
class  A{
        int  i,j;
public:    static int x;
public:    A(int a=0,int b=0,int c=0){  i=a ;  j=b ; x=c;      }
        void Show(){
                cout << "i="<<i<<'\t'<<"j="<<j<<'\t';
                cout << "x="<<x<<"\n";
                }
};
int A::x=500;  //int A::x
void  main(void )
{       A a(20,40,10),b(30,50,100);
        a.Show ();
        b.Show ();
        cout <<“A::x=”<<A::x<<‘\n’; //可以直接用类名引用
}
```

在类外重新定义

3.4 变量及对象的生存期和作用域

- 变量的生存期是指变量所占据的内存空间由分配到释放的时期。变量有效的范围称为其作用域。全局变量是程序中定义在所有函数（包括main函数）之外的任何变量，其作用域是程序从变量定义到整个程序结束的部分。这意味着全局变量可以被所有定义在全局变量之后的函数访问。全局变量及静态变量分配的空间在全局数据区，它们的生存期为整个程序的执行期间。
- 而局部变量，如在函数内或程序块内说明的变量，被分配到局部数据区，如栈区等。这种分配是临时的，一旦该函数体或程序块运行结束，所分配的空间就会被撤销。局部变量的生存期从被说明处开始，到所在程序块结束处结束。
- 对于静态变量，如果没有进行初始化，系统会自动初始化为0。局部变量如果没有进行初始化，则其值是不确定的。
- 使用new运算符创建的变量具有动态生存期。从声明处开始，直到用delete运算符释放存储空间或程序结束时，变量生存期结束。

3.4.2 类对象的生存期和作用域

类的对象在生成时调用构造函数，在消亡时调用析构函数，在这两个函数调用之间即是对象的生存期。

真题演练

下列说法不正确的是（　　）。

A:主函数main中定义的变量在整个文件或程序中有效

B:不同函数中，可以使用相同名字的变量

C:形式参数是局部变量

D:在一个函数内部，可以在复合语句中定义变量，这些变量只在复合语句中有效

真题演练

下列说法不正确的是（　　）。

A:主函数main中定义的变量在整个文件或程序中有效

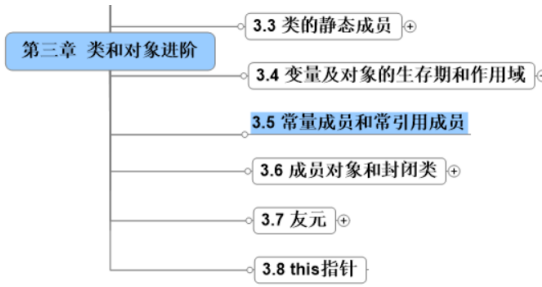
B:不同函数中，可以使用相同名字的变量

C:形式参数是局部变量

D:在一个函数内部，可以在复合语句中定义变量，这些变量只在复合语句中有效

答案：A

主函数main中定义的变量只在主函数中有效。



3.5 常量成员和常引用成员

- 在类中，也可以使用const关键字定义成员变量和成员函数，甚至是类的对象。由关键字const修饰的类成员变量称为类的常量成员变量。类的常量成员变量必须进行初始化，而且只能通过构造函数的成员初始化列表的方式进行。使用const修饰的函数称为常量函数。定义类的对象时如果在前面添加const关键字，则该对象称为常量对象。定义常量对象或常量成员变量的一般格式如下：

  const 数据类型 常量名=表达式;

- 定义常量函数的格式如下:

  类型说明符 函数名（参数表）const;

- 在对象被创建以后，其常量成员变量的值就不允许被修改，只可以读取其值。对于常量对象，只能调用常量函数。总之，常量成员变量的值不能修改，常量对象中的各个属性值均不能修改。

说明常量对象后，不能通过常量对象调用普通成员函数，见例3-26。

例3-26 使用常量对象不能调用非常量成员函数

```
class CDemo
{
public:
    void SetValue(){}        //非常量成员函数
};
```

如果主函数中有下列语句：

```
const CDemo Obj;             //Obj 是常量对象
Obj.SetValue();              //错误！
```

3.6 成员对象和封闭类

一个类的成员变量如果是另一个类的对象，则该成员变量称为“成员对象”。这两个类为包含关系。包含成员对象的类叫作封闭类。

例如，有类A和类B,在类B中定义了一个成员变量v，v的类型是类A；或者在类B中定义了一个函数，返回值类型是类A，则类A和类B是包含关系，更确切地说，类B包含类A，类B即是封闭类。

程序2-2中定义的类Student和类myDate，就是包含关系的类，类Student是封闭类。

3.6.1 封闭类构造函数的初始化列表

当生成封闭类的对象并进行初始化时，它包含的成员对象也需要被初始化，需要调用成员对象的构造函数。在定义封闭类的构造函数时，需要添加初始化列表，指明要调用成员对象的哪个构造函数。在封闭类构造函数中添加初始化列表的格式如下：

封闭类名::构造函数名(参数表): 成员变量1(参数表),成员变量2(参数表),...

{...}

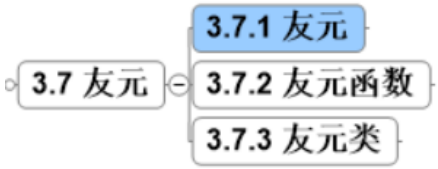
初始化列表中的成员变量既可以是成员对象，也可以是基本数据类型的成员变量。对于成员对象，初始化列表的“参数表”中列出的是成员对象构造函数的参数（它指明了该成员对象如何初始化）。

先调用成员对象的构造函数，再调用封闭类对象的构造函数。

*3.6.2 封闭类的复制构造函数

如果封闭类的对象是用默认复制构造函数初始化的，那么它包含的成员对象也会用复制构造函数初始化。

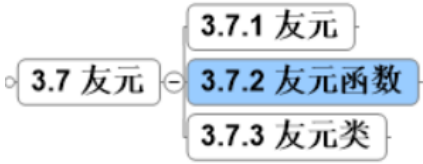
3.7 友元



友元实际上并不是面向对象的特征，而是为了兼顾C语言程序设计的习惯与C++友元的概念破坏了类的封装性和信息隐藏，但有助于数据共享，**能够提高程序执行的效率。**信息隐藏的特点，而特意增加的功能。这是一种类成员的访问权限。

友元使用关键字friend标识。在类定义中，当friend出现在函数说明语句的前面时，表示该函数为类的友元函数。一个函数可以同时说明为多个类的友元函数，一个类中也可以有多个友元函数。当friend出现在类名之前时，表示该类为类的友元类。

3.7.2 友元函数



在定义一个类的时候，可以把一些函数（包括全局函数和其他类的成员函数）声明为“友元”，这样那些函数就成为本类的友元函数。在友元函数内部可以直接访问本类对象的私有成员。在类定义中，将一个全局函数声明为本类友元函数的格式如下：

friend 返回值类型 函数名(参数表);

当有某类A的定义后，将类A的成员函数说明为本类的友元函数的格式如下:

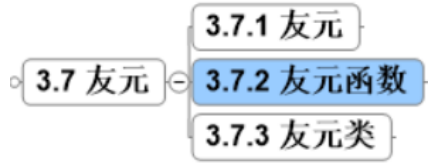
friend 返回值类型 类A::类A的成员函数名(参数表);

不能把其他类的私有成员函数声明为友元函数。

友元函数不是类的成员函数，但允许访问类中的所有成员。在函数体中访问对象成员时，必须使用“对象名.对象成员名”的方式。

友元函数不受类中的访问权限关键字限制，可以把它放在类的公有、私有、保护部分，结果是一样的。

3.7.2 友元函数



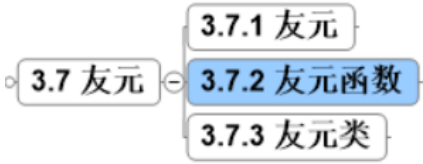
例3-29单项选择题

以下选项中，C++语言增加友元函数的目的是【 】

A.让其成为类的成员　　B.保证了数据的安全

C.能够访问类的私有成员　　D.破坏类访问的安全性

3.7.2 友元函数



例3-29单项选择题

以下选项中，C++语言增加友元函数的目的是【 】

A.让其成为类的成员　　B.保证了数据的安全

C.能够访问类的私有成员　　D.破坏类访问的安全性

答案：C

【分析】友元函数不是类的成员，在类中声明友元函数，确实破坏了类访问的安全性，但这不是增加友元函数的目的，而是它的副作用。声明友元函数的目的，是为了兼容C语言的特点，提供访问的方便性。在友元函数中可以访问类的私有成员，这是它的目的。

```
#include<iostream>
#include<cmath>
using namespace std;
class Pixel;    //前向引用声明
class Test
{
public:
    void printX(Pixel p); //用到了类Pixel
};
class Pixel
{
private:
    int x,y;
public:
    Pixel(int x0, int y0)
    {
        x=x0;
        y=y0;
    }
    void printxy()
    {   cout<<"pixel:("<<x<<","<<y<<")"<<endl;}
    friend double getDist(Pixel p1,Pixel p2);
    friend void Test::printX(Pixel p);
};
void Test::printX(Pixel p)
{
    cout<<"x="<<p.x<<"\ty="<<p.y<<endl;  //访问类Pixel的私有成员
    return;
}
double getDist(Pixel p1,Pixel p2)   //友元函数在类体外定义
{
    double xd=double(p1.x-p2.x);    //使用类Pixel的私有成员x
    double yd=double(p1.y-p2.y);    //使用类Pixel的私有成员y
    return sqrt(xd*xd+yd*yd);       //两点间距离
}

int main()
{
    Pixel p1(0,0),p2(10,10);
    p1.printxy();
    p2.printxy();
    cout<<"(p1,p2)间距离="<<getDist(p1,p2)<<endl;
    Test t;
    cout<<"从友元函数中输出--"<<endl;
    t.printX(p1); //通过对象调用类的成员函数
    t.printX(p2);//通过对象调用类的成员函数
    return 0;
}
```

真题演练

下列描述错误的是（　）。

A:友元是本类的成员函数

B:静态成员是类的成员，不是对象成员

C:静态成员不能是虚函数

D:静态成员函数不能直接访问非静态成员

真题演练

下列描述错误的是（　）。

A:友元是本类的成员函数

B:静态成员是类的成员，不是对象成员

C:静态成员不能是虚函数

D:静态成员函数不能直接访问非静态成员

答案：A

真题演练

下列关于友元的描述错误的是（　）。

A:成员函数不可作友元

B:类可以作友元

C:普通函数可以作友元

D:静态函数可以作友元

真题演练

下列关于友元的描述错误的是（　）。

A:成员函数不可作友元

B:类可以作友元

C:普通函数可以作友元

D:静态函数可以作友元

答案：A

真题演练

下列关于友元函数的描述，正确的是（　）。

A:友元函数可以存取私有成员、公有成员和保护成员

B:友元函数不可以是一个类

C:友元函数的作用之一是实现数据的隐藏性

D:在类中说明的友元函数，函数的定义不可在类体之外

真题演练

下列关于友元函数的描述，正确的是（　）。

A:友元函数可以存取私有成员、公有成员和保护成员

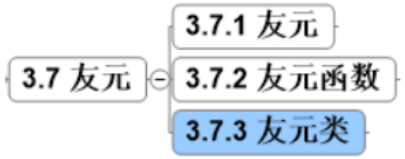
B:友元函数不可以是一个类

C:友元函数的作用之一是实现数据的隐藏性

D:在类中说明的友元函数，函数的定义不可在类体之外

答案：A

3.7.3 友元类



如果将一个类B说明为另一个类A的**友元类**，则类B中的所有函数都是类A的友元函数，在类B的所有成员函数中都可以访问类A中的所有成员。在类定义中声明友元类的格式如下：

friend class 类名;

友元类的关系是**单向**的。若说明类B是类A的友元类，不等于类A也是类B的友元类。友元类的关系不能传递，即若类B是类A的友元类，而类C是类B的友元类，不等于类C是类A的友元类。

除非确有必要，一般不把整个类说明为友元类，而仅把类中的某些成员函数说明为友元函数。

课堂练习

下面关于友元的描述中，错误的是（　）

A:友元函数可以直接访问该类的私有成员

B:一个类的友元类中的所有成员函数都是这个类的友元函数

C:利用友元可以提高程序的运行效率但却破坏了封装性

D:友元关系不能被继承，是双向可交换的

课堂练习

下面关于友元的描述中，错误的是（　）

A:友元函数可以直接访问该类的私有成员

B:一个类的友元类中的所有成员函数都是这个类的友元函数

C:利用友元可以提高程序的运行效率但却破坏了封装性

D:友元关系不能被继承，是双向可交换的

答案：D

课堂练习

已知类M是类N的友元类，类N是类K的友元类，则下列选项中，描述正确的是（　）。

A:类M一定是类K的友元类

B:类K一定是类M的友元类

C:类K的成员函数中可以访问类N的对象的任何成员

D:类M的成员函数中可以访问类N的对象的任何成员

课堂练习

已知类M是类N的友元类，类N是类K的友元类，则下列选项中，描述正确的是（　）。

A:类M一定是类K的友元类

B:类K一定是类M的友元类

C:类K的成员函数中可以访问类N的对象的任何成员

D:类M的成员函数中可以访问类N的对象的任何成员

答案：D

课堂练习

已知类A是类B的友元，类B是类C的友元，则（　）

A:类A一定是类C的友元

B:类C一定是类A的友元

C:类C的成员函数可以访问类B的对象的任何成员

D:类A的成员函数可以访问类B的对象的任何成员

课堂练习

已知类A是类B的友元，类B是类C的友元，则（　）

A:类A一定是类C的友元

B:类C一定是类A的友元

C:类C的成员函数可以访问类B的对象的任何成员

D:类A的成员函数可以访问类B的对象的任何成员

答案：D

真题演练

如果类A被声明成类B的友元，则（ ）。

A:类A的成员即类B的成员

B:类B的成员即类A的成员

C:类A的成员函数不得访问类B的成员

D:类B不一定是类A的友元

真题演练

如果类A被声明成类B的友元，则（ ）。

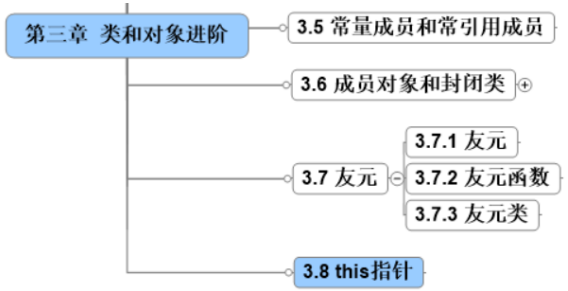
A:类A的成员即类B的成员

B:类B的成员即类A的成员

C:类A的成员函数不得访问类B的成员

D:类B不一定是类A的友元

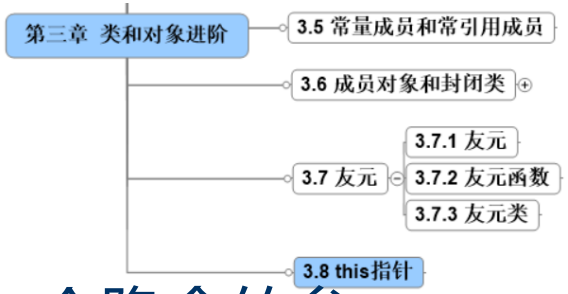
答案：D



3.8 this指针

```
class myDate
{
public:
        myDate();
        myDate(int, int, int);
        ...
private:
        int year, month, day;
};
myDate::myDate()
{       year = 1970;
        month = 1;
        day = 1;
}
myDate::myDate(int y, int m, int d)
{       year = y;
        month = m;
        day = d;
}
...
```

```
myDate::myDate(int year, int month, int day)
{
    this->year = year;
    this->month = month;
    this->day = day;
}
```



3.8 this指针

- C++语言规定，当调用一个成员函数时，系统自动向它传递一个隐含的参数。该参数是一个指向调用该函数的对象的指针，称为this指针，从而使成员函数知道对哪个对象进行操作。
- C++规定，在非静态成员函数内部可以直接使用this关键字，this就代表指向该函数所作用的对象的指针。
- 在一般情况下，在不引起歧义时，**可以省略“this->”**，系统采用默认设置。
- 静态成员是类具有的属性，不是对象的特征，this表示的是隐藏的对象的指针，所以**静态成员函数没有this指针**。

课堂练习

下面对友元的描述中，错误的是【 】

A.关键字friend用于声明友元

B.一个类中的成员函数可以是另一个类的友元

C.友元函数访问对象的成员不受访问特性影响

D.友元函数通过this指针访问对象成员

课堂练习

下面对友元的描述中，错误的是【 】

A.关键字friend用于声明友元

B.一个类中的成员函数可以是另一个类的友元

C.友元函数访问对象的成员不受访问特性影响

D.友元函数通过this指针访问对象成员

答案：D。

【分析】使用friend修饰的函数，既可以是其他类的成员函数，也可以是全局函数，但不是本类的成员函数。因为是友元，所以可以访问本类内的所有成员，包括私有成员，故友元函数访问对象的成员不受访问特性影响。this指针指向的是成员函数作用的对象，也就是调用成员函数的对象。友元函数不通过对象调用，所以没有this指针。

课堂练习

this指针存在的目的是（　）。

A:保证基类公有成员在子类中可以被访问

B:保证每个对象拥有自己的数据成员，但共享处理这些数据成员的代码

C:保证基类保护成员在子类中可以被访问

D:保证基类私有成员在子类中可以被访问

课堂练习

this指针存在的目的是（　）。

A:保证基类公有成员在子类中可以被访问

B:保证每个对象拥有自己的数据成员，但共享处理这些数据成员的代码

C:保证基类保护成员在子类中可以被访问

D:保证基类私有成员在子类中可以被访问

答案：B

课堂练习

以下关于this指针的叙述中，正确的是（　）。

A:任何与类相关的函数都有this指针

B:类的成员函数都有this指针

C:类的友元函数都有this指针

D:类的非静态成员函数才有this指针

课堂练习

以下关于this指针的叙述中，正确的是（　）。

A:任何与类相关的函数都有this指针

B:类的成员函数都有this指针

C:类的友元函数都有this指针

D:类的非静态成员函数才有this指针

答案：D

课堂练习

不同对象调用同一成员函数时，this指针指向（　）。

A:不同对象

B:相同对象

C:无对象

D:不确定

课堂练习

不同对象调用同一成员函数时，this指针指向（　）。

A:不同对象

B:相同对象

C:无对象

D:不确定

答案：D

课堂练习

关于this指针的说法错误的是（　）。

A:this指针必须显式声明

B:当创建一个对象后，this指针就指向该对象

C:动态成员函数拥有this指针

D:静态成员函数不拥有this指针

课堂练习

关于this指针的说法错误的是（　）。

A:this指针必须显式声明

B:当创建一个对象后，this指针就指向该对象

C:动态成员函数拥有this指针

D:静态成员函数不拥有this指针

答案：A

本章总结


第三章 类和对象进阶
3.1 构造函数
3.1.1 构造函数的作用
3.1.2 构造函数的定义
3.1.3 构造函数的使用
3.1.4 复制构造函数与类型转换构造函数
3.2 析构函数
3.3 类的静态成员
3.3.1 静态变量
3.3.2 类的静态成员
3.4 变量及对象的生存期和作用域
3.4.1 变量的生存期和作用域
3.4.2 类对象的生存期和作用域
3.5 常量成员和常引用成员
3.6 成员对象和封闭类
3.6.1 封闭类构造函数的初始化列表
3.6.2 封闭类的复制构造函数
3.7 友元
3.7.1 友元
3.7.2 友元函数
3.7.3 友元类
3.8 this指针

祝大家顺利通过考试！