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

（2008年版）

编著：刘振安

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×20题 = 20分

程序改错题　4分×5题 = 20分

完成程序题　4分×5题 = 20分

程序分析题　5分×2题 = 10分

程序设计题 10分×1题 = 10分

第五章　特殊函数和成员

目录

第5章
特殊函数
和成员

§5.1 对象成员的初始化

一、对象成员的概念

可以在一个类中说明具有某个类的类型的数据成员，这些成员称为对象成员。在类A中说明对象成员的一般形式为：

```
class A{
    类名1 成员名1;
    类名2 成员名2;
……
    类名n 成员名n;
};
```

说明对象成员是在其所属类名之后给出对象成员的名字。

二、含对象成员的类的构造函数格式

为初始化这些对象成员，A类的构造函数需要分别调用这些对象成员所属类的构造函数。含有对象成员的类A的构造函数的定义形式为：

A::A(参数表0) :成员1 (参数表1)，成员2 (参数表2)，…，成员n (参数表n)

{…… //其他操作}

冒号“：”后由逗号隔开的项组成成员初始化列表，其中的参数表给出了为调用相应成员所在类的构造函数时应提供的参数，参数列表中的参数都来自于“参数表0”，可以使用任意复杂的表达式，其中可以有函数调用。

对象成员构造函数的调用顺序取决于这些对象成员在类A中说明的顺序，与它们在成员初始化列表中的顺序无关。

当建立A类的对象时，先调用对象成员的构造函数，初始化对象成员，然后才执行A类的构造函数，初始化A类中的其他成员。析构函数的调用顺序与构造函数正好相反。

例：含对象成员的类如何调用构造函数和析构函数

```
#include <iostream>
using namespace std;
class object{
private:
    int val;  //声明int型的object类的私有数据成员val
public:
    object( ):val(0) {cout<<"信息一"<<endl;}  //定义不带参数的object类的构造函数object为内联函数
    object(int i):val(i) {cout<<"信息二"<<val<<endl;}  //定义带一个参数的object类的构造函数object为内联函数
    ~object( ) {cout<<"信息三"<<val<<endl;}  //定义object类的析构函数为内联函数
};                              //类object的声明结束
class container{
private:
    object one;  //声明object类的对象one为container类的对象成员，初始化顺序为第一
    object two;  //声明object类的对象two为container类的对象成员，初始化顺序为第二
    int data;    //声明int型的container类的私有数据成员data，初始化顺序为第三
public:
    container( ):data(0) {cout<<"信息四"<<endl;}  //定义不带参数的container类的构造函数container为内联函数
    container(int i,int j,int k);  //声明带3个int型参数的container类的构造函数container
    ~container( ) {cout<<"信息六"<<data<<endl;}  //定义container类的析构函数为内联函数
};                          //类container的声明结束
container::container(int i,int j,int k):two(i),one(j)
{data=k;cout<<"信息五"<<data<<endl;}  //定义带3个int型参数的container类的构造函数
void main( ){
     container obj,anObj(5,6,10);}
```

```
信息一
信息一
信息四
信息二6
信息二5
信息五10
信息六10
信息三5
信息三6
信息六0
信息三0
信息三0
```

三、基本数据类型的成员和特殊成员的初始化

1、基本数据类型的成员的初始化

如上例中container类的基本数据类型成员data的初始化，可如上例中在函数体内通过语句“data=k；”初始化，也可在成员初始化列表中进行，如将语句改为:

```
container::container(int i,int j,int k):two(i),one(j),data(k)
{
    cout<<”信息五”<<data<<endl;
}
```

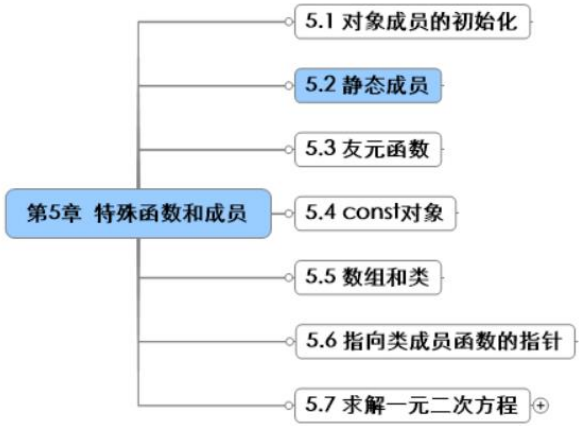
2、特殊成员的初始化

特殊成员如const成员、引用成员的初始化，则必须通过成员初始化列表进行，

如：

```
class exam{
private:
    const int num;
    int& ret;
public:
    exam(int n,int f):num(n),ret(f){ }
};
```

§5.2 静态成员



一、静态成员概念

简单成员函数是指声明中不含const、volatile、static关键字。

如果类的数据成员或成员函数使用关键字static修饰，这样的成员称为静态数据成员或静态成员函数，统称为静态成员。

例：含静态成员的程序

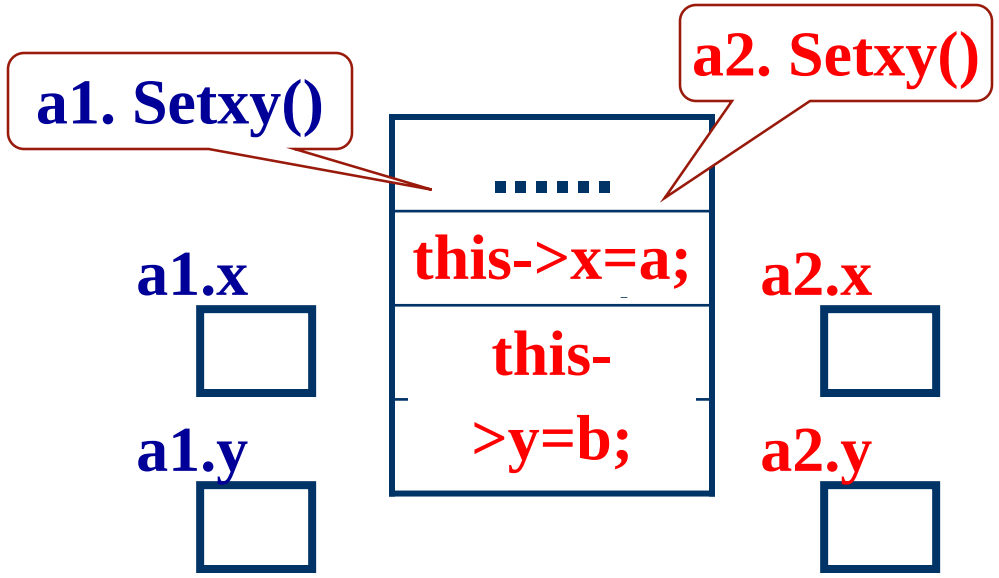
```
class Test{
    static int x;           //声明静态数据成员
    int n;
public:
    Test( ){ }      //定义无参数的Test类的构造函数
    Test(int a,int b){x=a;n=b;}  //定义含两个参数的Test类的构造函数Test为内联函数
    static int func( ){return x;}  //定义静态成员函数func为内联函数
    static void sfunc(Test&r,int a){r.n=a;}  //定义静态成员函数sfunc为内联函数,函数以Test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                    25 58 25 25 24 24 24
    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;
}
```

静态成员

通常，每当说明一个对象时，把该类中的有关成员数据拷贝到该对象中，即同一类的不同对象，其成员数据之间是互相独立的。

```
class A{
    int x,y;
public:
    void Setxy(int a, int b)
    {  x=a;   y=b;}
};
A   a1,a2;
```

```
a1.Setxy(1,2);
a2.Setxy(3,4);
```

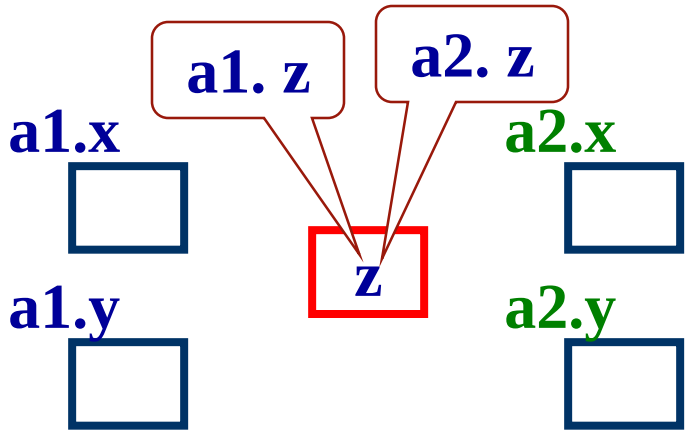


当我们将类的某一个数据成员的存储类型指定为静态类型时，则由该类所产生的所有对象，其静态成员均共享一个存储空间，这个空间是在编译的时候分配的。换言之，在说明对象时，并不为静态类型的成员分配空间。

在类定义中，用关键字static修饰的数据成员称为静态数据成员。

```
class A{
    int x,y;  static  int z;
public:
    void Setxy(int a, int b)
    { x=a; y=b;}
};
A   a1,a2;
```

不同对象，同一空间



有关静态数据成员的使用，说明以下几点：

1、类的静态数据成员是静态分配存储空间的，而其它成员是动态分配存储空间的（全局变量除外）。当类中没有定义静态数据成员时，在程序执行期间遇到说明类的对象时，才为对象的所有成员依次分配存储空间，这种存储空间的分配是动态的；而当类中定义了静态数据成员时，在编译时，就要为类的静态数据成员分配存储空间。

2、必须在文件作用域中，对静态数据成员作一次且只能作一次定义性说明。因为静态数据成员在定义性说明时已分配了存储空间，所以通过静态数据成员名前加上类名和作用域运算符，可直接引用静态数据成员。在C++中，静态变量缺省的初值为0，所以静态数据成员总有唯一的初值。当然，在对静态数据成员作定义性的说明时，也可以指定一个初值。

```
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

```
#include<iostream.h>
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( ){
        cout<<"A::x="<<A::x<<'\n';
        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
}
```

```
A::x=500
i=20    j=40    x=100
i=30    j=50    x=100
A::x=100
```

3、静态数据成员具有全局变量和局部变量的一些特性。静态数据成员与全局变量一样都是静态分配存储空间的，但全局变量在程序中的任何位置都可以访问它，而静态数据成员受到访问权限的约束。必须是public权限时，才可能在类外进行访问。

4、为了保持静态数据成员取值的一致性，通常在构造函数中不给静态数据成员置初值，而是在对静态数据成员的定义性说明时指定初值。

```
#include<iostream.h>
class  A{
        int  i;
        static int count;
public:
        A(int a=0){
                i=a; count++;
           cout<<"Number of Objects="<<count<<'\n';}
        ~A(){
           count--;
                cout<<"Number of Objects="<<count<<'\n';}
        void Show(){
                cout<<"i="<<i<<'\n';
           cout<<"count="<<count<<"\n"; }
};
int A::count;
void main( )
{
        A  a1(100);
        A  b[2];
        a1.Show();
}
```

```
Number of Objects=1
Number of Objects=2
Number of Objects=3
i=100
count=3
Number of Objects=2
Number of Objects=1
Number of Objects=0
```

静态成员函数

可以将类的成员函数定义为静态的成员函数。即使用关键字static来修饰成员函数 。

```
class A
{    float x, y;
public :
     A( ){  }
     static  void sum(void)  { ..... }
};
```

对静态成员函数的用法说明以下几点：

1、与静态数据成员一样，在类外的程序代码中，通过类名加上作用域操作符，可直接调用静态成员函数。

2、静态成员函数只能直接使用本类的静态数据成员或静态成员函数，但不能直接使用非静态的数据成员（可以引用使用）。这是因为静态成员函数可被其它程序代码直接调用，所以，它不包含对象地址的this指针。

```
class Tc {
private:int A;
        static int B;//静态数据成员
public:Tc(int a){A=a;  B+=a;}
        static void display(Tc  c)    //Tc的对象为形参
        {
              cout<<"A="<<c.A<<",B="<<B<<endl;
        }
};
int Tc::B=6;
void main(void)
{       Tc  a(2),b(4);
        Tc::display (a);
        Tc::display (b);
}
```

非静态成员，用对象名来引用

静态成员，直接引用

直接用类名来调用静态成员函数

A=2,B=12

A=4,B=12

3、静态成员函数的实现部分在类定义之外定义时，其前面不能加修饰词static。这是由于关键字static不是数据类型的组成部分，因此，在类外定义静态成员函数的实现部分时，不能使用这个关键字。

4、不能把静态成员函数定义为虚函数。静态成员函数也是在编译时分配存储空间，所以在程序的执行过程中不能提供多态性。

5、可将静态成员函数定义为内联的（inline），其定义方法与非静态成员函数完全相同。

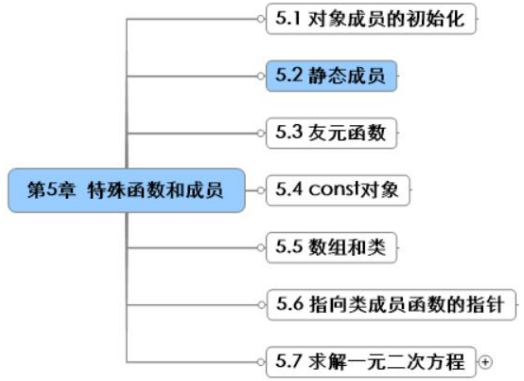
```
class Tc {
private:int A;
        static int B;//静态数据成员
public:Tc(int a){A=a;  B+=a;}
        static void display(Tc  c);  //Tc的对象为形参
};
void  Tc::display(Tc  c)    //不用static修饰
      {       cout<<"A="<<c.A<<",B="<<B<<endl;          }
int Tc::B=2;
void main(void)
{
        Tc  a(2),b(4);
        Tc::display (a);
        Tc::display (b);
}
```

函数原型

类外定义

A=2,B=8
A=4,B=8

二、静态成员函数与一般成员函数的不同



1、可以不指向某个具体的对象，只与类名连用；

2、在没有建立对象之前，静态成员就已存在；

3、静态成员是类的成员，不是对象的成员；

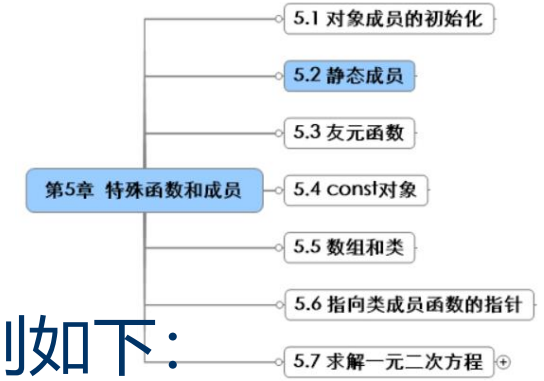
4、静态成员为该类的所有对象共享，它们被存储于一个公用内存中；

5、没有this指针，只能通过对象名或指向对象的指针访问类的数据成员；

6、静态成员函数不能被说明为虚函数；

7、静态成员函数不能直接访问非静态函数。

三、类的静态对象与普通对象的对比使用实例



静态对象是使用关键字static声明的类的对象，它与普通对象区别如下：

1、静态对象的构造函数在代码执行过程中，在第一次遇到它的变量定义并初始化时被调用，但直到整个程序结束之前仅调用一次；而普通对象则是遇到变量定义就被调用，遇到几次调用几次。

2、静态对象的析构函数在整个程序退出之前被调用，同样也只调用一次；而普通对象则是变量被定义几次，则析构几次。

```
#include <iostream>
using namespace std;
class test{
private:
    int n;    //声明test类的私有数据成员n
public:
    test(int i){ n=i;cout<<"构造:"<<i<<endl;}  //定义含一个整型参数的test类的内联构造函数为公有成员函数
    ~test( ){cout<<“析构:”<<n<<endl; }   //定义test类的内联析构函数
    int getn( ){return n;}     //定义test类的内联函数getn
    void inc( ){++n;}    //函数作用为：使用n之前，使n的值加1
};
void main( )
{
    cout<<"循环开始:"<<endl;
    for(int i=0;i<3;i++){
        static test a(3);    //定义test类的静态对象a并初始化
        test b(3);  //定义test类的普通对象b并初始化
        a.inc( );  //test类的静态对象a调用其同类的成员函数inc
        b.inc( );  //test类的普通对象b调用其同类的成员函数inc
        cout<<"a.n="<<a.getn( )<<endl;
        cout<<"b.n="<<b.getn( )<<endl; }
    cout<<"循环结束"<<endl;
    cout<<“退出主程序”<<endl;}
```

```
循环开始:
构造:3
构造:3
a.n=4
b.n=4
析构:4
构造:3
a.n=5
b.n=4
析构:4
构造:3
a.n=6
b.n=4
析构:4
循环结束
退出主程序
析构:6
```

§5.3 友元函数

可以在类A中通过关键字friend声明或定义某个独立函数或另一个类B的某个成员函数或另一个类B为类A的友元函数，友元函数可以无限制的存取类A的成员（包括私有、公有和保护成员）。

定义形式：

friend 函数类型 函数所在类名::函数名（参数列表）；

友元函数可在类中的私有或公有部分通过关键字friend说明或定义，但如在类中声明，而在类外定义，就不能再在类外使用friend关键字。友元函数作用域的开始点在它的说明点，结束点和类的作用域相同。

§5.3 友元函数

友元函数应被看作类的接口的一部分，使用它的主要目的是提高效率，因为它可以直接访问对象的私有成员，从而省去调用类的相应成员函数的开销。

友元函数的另一个优点是：类的设计者不必在考虑好该类的各种可能使用情况之后再设计这个类，而是可以根据需要，通过使用友元来增加类的接口。

有时两个概念上相近的类要求其中一个类可以无限制地存取另一个类的成员。友元函数解决了这类难题。友元函数可以存取私有成员、公有成员和保护成员。其实友元函数可以是一个类或函数，尚未定义的类也可以作为友元引用。

友元函数不是成员函数，用法也与普通的函数完全一致，只不过它能访问类中所有的数据。友元函数破坏了类的封装性和隐蔽性，使得非成员函数可以访问类的私有成员。

一个类的友元可以自由地用该类中的所有成员。

```
class A{
        float  x,y;
public:
A(float a, float b){  x=a; y=b;}
float  Sum(){  return  x+y;  }
friend  float Sum(A &a){     return   a.x+a.y;      }
};
void main(void)
{  A   t1(4,5),t2(10,20);
    cout<<t1.Sum()<<endl;
   cout<<Sum(t2)<<endl;
}
```

友元函数只能用对象名引用类中的数据。

成员函数

友元函数

私有数据

成员函数的调用，利用对象名调用

友元函数的调用，直接调用

有关友元函数的使用，说明如下：

友元函数不是类的成员函数

友元函数近似于普通的函数，它不带有this指针，因此必须将对象名或对象的引用作为友元函数的参数，这样才能访问到对象的成员。

一、类外的独立函数作为类的友元函数

虽然在类中说明友元函数，但它并不是类的成员函数，所以可以在类外面像普通函数那样定义这个函数、这个独立函数其实就是一个普通的函数，仅有的不同点是：它在类中被说明为类的友元函数，可以访问该类所有对象的私有成员。

```
#include <iostream>
#include <cmath>
using namespace std;
class Point {
private:
    double X,Y;
public:
    Point( double xi,double yi){X=xi,Y=yi;}  //类Point的构造函数
    double GetX( ){return X;}
    double GetY( ){return Y;}
    friend double distances( Point&, Point&);  //声明友元函数
};
double distances( Point& a, Point& b)  //像普通函数一样定义友元函数
{   double dx=a.X-b.X;  //因是友元函数，所以可以直接访问对象的私有数据成员
    double dy=a.Y-b.Y;  //因是友元函数，所以可以直接访问对象的私有数据成员
    return sqrt( dx*dx + dy*dy );
}
void main( ){
    Point p1(3.5,5.5),p2(4.5,6.5);
    cout<<"距离是"<<distances(p1,p2)<<endl;
}
```

距离是1.41421

二、类B的成员函数作为类A的友元函数

一个类的成员函数（包括构造函数和析构函数）可以通过使用friend说明为另一个类的友元函数。例如将类One的成员函数func说明为类Two的友元，可在类Two中用语句：“friend void One::func(Two&);”实现，其中因func属于类One，所以要用限定符说明出处。这样One的对象就可以通过友元函数func（Two&）访问类Two的所有成员，因为是访问类Two，所以应使用类Two对象的引用作为传递参数。

类One的对象通过友元函数访问类Two的对象的私有数据

```
#include <iostream>
using namespace std;
class Two;  //先声明类Two，以便类One引用Two&
class One {
private:
    int x;
public:
    One(int a){x=a;}  //定义类One的构造函数为内联公有函数
    int Getx( ){return x;}
    void func(Two&);  //声明本类的成员函数，参数为类Two的引用
};          //类One声明结束
class Two{
private:
    int y;
public:
    Two(int b){y=b;}  //定义类Two的构造函数为内联公有函数
    int Gety( ){return y;}
    friend void One::func(Two&);  //声明类One的成员函数为本类的友元函数
};          //类Two声明结束
void One::func(Two& r) {r.y=x;}
void main( ){
    One Obj1(5);    //生成类One的对象Obj1
    Two Obj2(8);    //生成类Two的对象Obj2
    cout<<Obj1.Getx( )<<" "<<Obj2.Gety( )<<endl;
//输出5 8
    Obj1.func(Obj2);
    cout<<Obj1.Getx( )<<" "<<Obj2.Gety( )<<endl;
//输出5 5
}
```

```
5 8
5 5
```

三、将一个类说明为另一个类的友元

可以将一个类One说明为另一个类Two的友元，这时，整个类One的成员函数均是类Two的友元函数，声明语句简化为：“friend class 类名；”。

```
#include <iostream>
using namespace std;
class Two{
    int y;
public:
    friend class One;  //声明类One为类Two的友元
};
class One{                  //类One的成员函数均是类Two的友元函数
    int x;
public:
    One(int a,Two&r,int b){x=a;r.y=b;}  //利用类One的构造函数给本类及类Two的对象赋值
    void Display(Two&);     //声明类One的成员函数，它能访问类Two的成员
};
void One::Display(Two&r){cout<<x<<" "<<r.y<<endl;}
void main( ){
    Two Obj2;
    One Obj1(23,Obj2,55);
    Obj1.Display(Obj2);
}
```

23 55

§5.4 const对象

可以在类中使用const关键字定义数据成员和成员函数或修饰一个对象。一个const对象只能访问const成员函数，否则将产生编译错误。

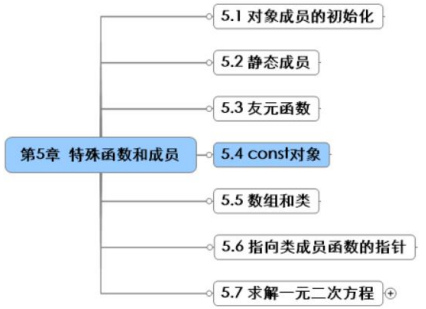
一、常量成员

常量成员包括静态常数据成员、常数据成员和常引用，其中前者仍保留静态成员特征，需要在类外初始化，后两者则只能通过初始化列表来获得初值。

```
#include <iostream>
using namespace std;
class Base{
private:
    int x;
    const int a;          //常数据成员只能通过初始化列表来获得初值
    static const int b;  //静态常数据成员需在类外初始化
    const int& r;        //常引用只能通过初始化列表来获得初值
public:
    Base(int,int);  //声明含两个整型参数的Base类的构造函数
    void Show( ){cout<<x<<","<<a<<","<<b<<","<<r<<endl;}
    void Display(const double& r){cout<<r<<endl;}
};
const int Base::b=125;    //静态常数据成员在类外初始化
Base::Base(int i,int j):x(i),a(j),r(x){ }  //初始化列表

void main( ){
    Base a1(104,118),a2(119,140);
    a1.Show( );                                          104,118,125,104
    a2.Show( );                                          119,140,125,119
    a2.Display(3.14159);                                 3.14159
}
```

二、常对象



在对象名前使用const声明的对象就是常对象，常对象必须在声明的同时进行初始化，而且不能被更新，形式为：

类名 const 对象名（参数表）；　　//必须在声明的同时进行初始化

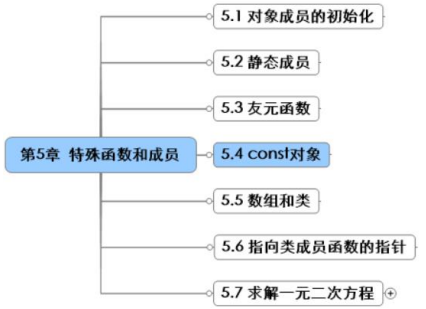
例如上例中定义一个Base类的常对象，语句为：Base const a(24,35);

三、常成员函数

为防止覆盖函数改变数据成员的值，可以将一个成员函数声明为const函数，const函数不能更新对象的数据成员，也不能调用该类中没有用const修饰的成员函数。

注意用const声明static成员函数没有什么作用，另外，在C++中声明构造函数和析构函数时使用const关键字是非法的，而volatile关键字的使用方法与const类似，因使用很少，故不再介绍。

三、常成员函数



对于一般对象，它可以调用所有成员函数，包括常成员函数，而对于const对象，它只能调用const函数，不能调用非const函数。故可以通过常成员函数参与对重载函数的区分。

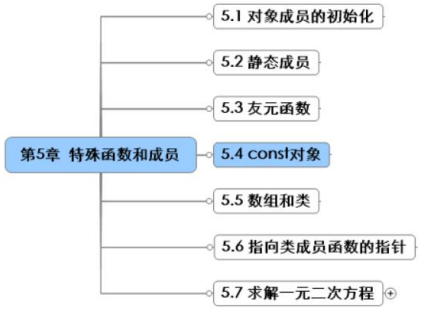
声明常成员函数的格式：类型标识符 函数名（参数列表） const;

定义格式：类型标识符 类名::函数名（参数列表） const {……//函数体}

在类中用内联函数定义const函数：

类型标识符 函数名（参数列表） **const** {……//函数体}

const函数的声明及定义形式



（1）在类体内定义const函数为内联函数时的形式

类型标识符 函数名（参数列表） const {……//函数体}

（2）在类体内声明，类体外定义时的形式

声明形式：类型标识符 函数名（参数列表） const;

定义形式：类型标识符 类名::函数名（参数列表） const {……//函数体}

const对象调用const函数的实例

```
#include <iostream>
using namespace std;
class Base{
private:
    double x,y;
    const double p;
public:
    Base(double m,double n,double d):p(d) {x=m;y=n;}
        //常数据成员只能通过初始化列表来获得初值
    void Show( );
    void Show( ) const;    //声明常成员函数
};
    void Base::Show( ){cout<<x<<","<<y<<" p="<<p<<endl;}
    void Base::Show( ) const {cout<<x<<","<<y<<" const p="<<p<<endl;}

void main( ){
    Base a(8.9,2.5,3.1416);    //调用语句1进行初始化
    const Base b(2.5,8.9,3.14);      //调用语句1进行初始化
    b.Show( );       //输出"2.5,8,9 const p=3.14"
    a.Show( );       //输出"2.5,8,9 p=3.1416"
}
```

2.5,8.9 const p=3.14
8.9,2.5 p=3.1416

const函数返回的常量对象与其他常量对象一起使用的实例

```
#include <iostream>
using namespace std;
class ConstFun{
public:
    int f5( ) const{return 5;}    //常成员函数，返回常量对象
    int Obj( ){return 45;}        //一般成员函数
};
void main( ){
    ConstFun s;  //声明ConstFun类的一般对象s
    const int i=s.f5( );  //对象s使用常成员函数f5( )初始化常整数i
    const int j=s.Obj( ); //对象s使用一般成员函数Obj( )初始化常整数j
    int x=s.Obj( );  //对象s使用一般成员函数Obj( )初始化整数x
    int y=s.f5( );  //对象s使用常成员函数f5( )初始化整数y
    cout<<i<<" "<<j<<" "<<x<<" "<<y<<endl;  //输出"5 45 45 5"
    ConstFun const d;  //声明ConstFun类的常对象d
    int k=d.f5( );    //常对象d只能调用常成员函数f5( )
    cout<<"k="<<k<<endl;    //输出"k=5"
}
```

```
5 45 45 5
k=5
```

§5.5 数组和类

类的引入产生了除C语言可以声明的数组类型外的一系列新的数组类型，下面简单介绍类对象数组和类对象指针数组。

ANSI C可以声明的数字类型都适用于C++，但类的引入产生了一系列新的数组类型。

一、使用类对象数组和指针的实例

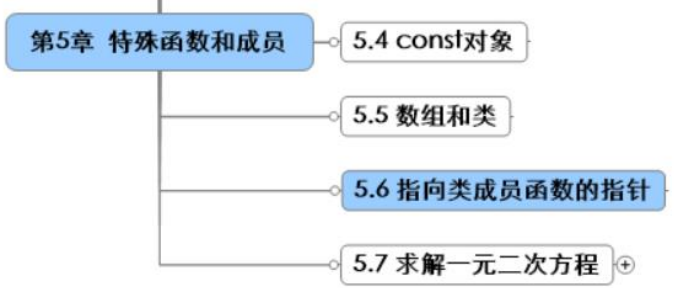
```
class Test{
    int num;
    double f1;
public:
    Test (int n){num=n;}  //语句1，一个参数的构造函数
    Test(int n,double f){num=n;f1=f;}  //语句2，两个参数的构造函数
    int GetNum( ){return num;}
    float GetF( ){return f1;}
};
#include <iostream>
using namespace std;
void main( ){
    Test one[2]={2,4},*p;
//定义含两个元素的Test类对象数组one和一个指向Test类的对象指针p，one的两个元素调用语句1初始化
    Test two[2]={Test(1,3.2),Test(5,9.5)};
//定义含两个元素的Test类的对象数组two，two的两个元素调用语句2初始化
    for (int i=0;i<2;i++)
        cout<<"one["<<i<<"]="<<one[i].GetNum( )<<",";
    p=two;  //因数组名即代表数组首地址，所以可以用数组名给指针p赋值
    for(i=0;i<2;i++,p++)
        cout<<"two["<<i<<"]=("<<p->GetNum( )<<","<<p->GetF( )<<"),";
}
```

one[0]=2,one[1]=4,two[0]=(1,3.2),two[1]=(5,9.5),

二、仍然使用类Test，但主程序改用对象指针数组演示


```
void main( ){
    Test *one[2]={new Test(2),new Test(4)};
    Test *two[2]={new Test(1,3.2),new Test(5,9.5)};
    for (int i=0;i<2;i++)
    cout<<” one[ “<<i<<” ]=” <<one[i]->GetNum( )<<” ,” ;
    for (i=0;i<2;i++,p++)
    cout<<” two[ “<<i<<” ]=(” <<two[i]->GetNum( )<<” ,”
        <<two[i]->GetF( )<<” ),” ;
}
```

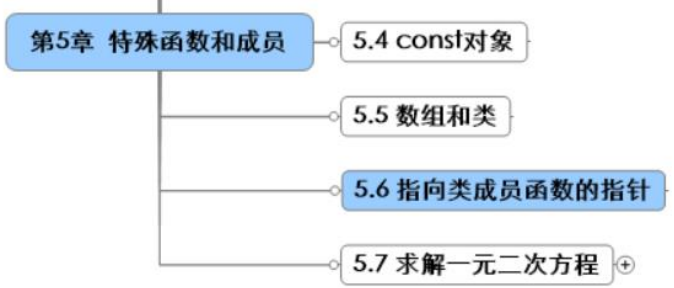
例中one和two都是直接用动态分配的对象初始化的，编译器自动调用构造函数语句1和语句2来产生one和two的每一个分量。



§5.6 指向类的成员函数的指针

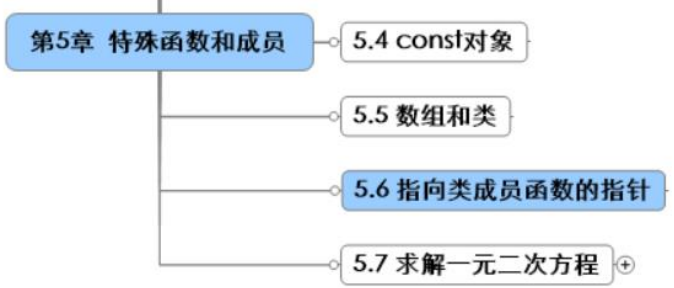
C++除普通指针外，还包含了能指向类的成员的指针。普通指针指向对象，而指向类的成员的指针则可指向某个特定类的对象的数据成员或成员函数。

C++既包含指向类数据成员的指针，又包含指向成员函数的指针。它提供一种特殊的指针类型，指针指向类的成员，而不是指向该类的一个对象中该成员的一个实例，这种指针称为指向类成员的指针。不过，指向类数据成员的指针要求数据成员是公有的，用途不大。



一、指向类的数据成员的指针

该指针要求数据成员必须是公有的，用途不大，故不详细介绍。

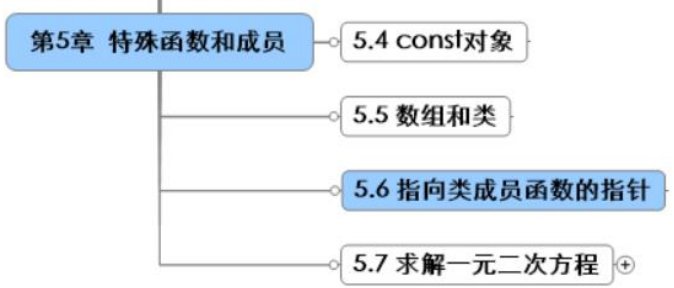


二、指向类的成员函数的指针

如类A的成员函数的参数类型列表为list，返回类型为type，则指向该函数的指针pointer的声明形式为“type（A::*pointer）(list);” ，此声明可读做：pointer是一个指针，指向类A的成员函数，此成员函数参数类型列表为list，返回值类型为type。

如果类A的成员函数fun的原型和pointer指向的函数的原型一样，则语句“pointer = A::fun;”将函数fun的地址（不是真实地址，而是在类A中所有对象的偏移）置给指针pointer，即指针pointer指向函数fun（好比数组的情况，数组名即是数组的首地址）。

在使用指向类成员函数的指针访问类的某个成员函数时，必须指定一个对象，通过对象名、引用或指向对象的指针调用该成员函数，其中前两者使用运算符“.*”，后者使用运算符“->”。



二、指向类的成员函数的指针

注意：程序中(obj.*pfun)或(pc->*pfun)必须用括号括起来。类似于指向对象的指针，指向类成员函数的指针也可看作存储的是类成员函数地址的指针变量，好比pfun为指向类成员函数的指针，指向类成员函数value，则有：

*pfun=value　　但和对象的p=&x;不同，这里是pfun=A::value，这和数组名即为数组首地址相似，value即是函数名，也看作是函数首地址。

```
#include <iostream>
using namespace std;
class A{
private:
    int val;
public:
    A(int i){val=i;}    //含一个整型参数的类A的构造函数
    int value(int a){return val+a;}
};
void main( ){
    int(A::*pfun)(int);    //声明指向类A的成员函数的指针pfun
    pfun=A::value;    //指针指向A的具体的成员函数value
    A obj(10);    //创建类A的对象obj，并调用构造函数初始化
    cout<<(obj.*pfun)(15)<<endl;  //对象obj调用指针指向的成员函数value，输出25
    A *pc=&obj;    //创建类型为A的指针pc，该指针指向类A的对象obj
    cout<<(pc->*pfun)(15)<<endl;  //指针pc调用成员函数value，输出25
}
```

25
25

§5.7 求解一元二次方程

设计代表方程的类：

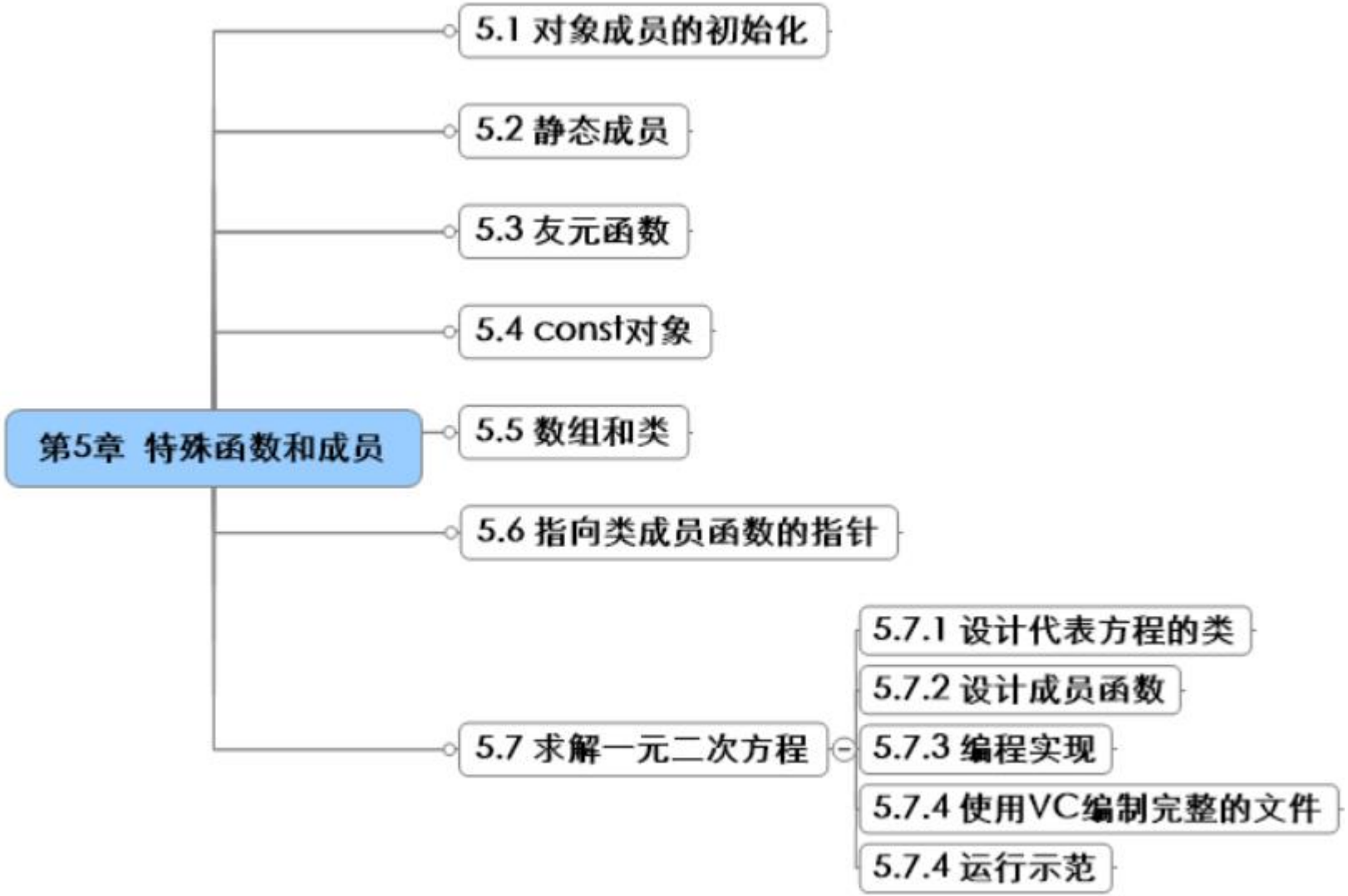
FindRoot
a:float b:float c:float d:float x1:double x2:double
FindRoot:FindRoot Find:void Display:void

类示意图

Obj:FindRoot
a=1 b=-3 c=2 d=1 x1=2 x2=1
FindRoot Find Display

obj 对象图

本章总结


第5章 特殊函数和成员
5.1 对象成员的初始化
5.2 静态成员
5.3 友元函数
5.4 const对象
5.5 数组和类
5.6 指向类成员函数的指针
5.7 求解一元二次方程
5.7.1 设计代表方程的类
5.7.2 设计成员函数
5.7.3 编程实现
5.7.4 使用VC编制完整的文件
5.7.4 运行示范

完成程序，使其输出结果为

n=2，sum=2

n=3，sum=5

n=5，sum=10

```
#include <iostream>
using namespace std;
class Sample{
   int n;
   static int sum;
public:
   Sample(int x){
      n=x;
   }
   void add(){
      sum+=n
   }
   void disp(){
      cout<<"n="<<n<<",sum="<<sum<<endl;
   }
};
int Sample::sum=0;
void main(){
   Sample a(2),b(3),c(5);
   a.add();
   a.disp();
   b.add();
   b.disp();
   c.add();
   c.disp();
}
```

祝大家顺利通过考试！