复习

- 构造函数与析构函数的概念与应用
- 拷贝构造函数

  用类的一个对象去初始化该类的另一个对象时；

  函数的形参是类的对象，调用函数进行形参和实参的结合时；

  函数的返回值是类的对象，函数执行完返回调用者时。
- 类的静态数据成员的应用

  解决数据共享；

  均属于类的静态成员而非对象成员；

  定义类的对象前进行静态数据成员初始化 。
- 类的静态成员函数

  静态成员函数和静态数据成员一样，它们都属于类的静态成员，它们都不是对象成员。

8.6　类的友元

友元可以是一个函数，该函数被称为友元函数；友元也可以是一个类，该类被称为友元类。

使用友元函数或友元类的成员函数可以访问类的私有成员。

8.6.1 友元函数

友元函数的特点：

（1）友元函数的特殊性在于：友元函数拥有访问类的私有数据成员的权力，普通性在于它是一个普通的函数，除了上述特殊性以外，与其他函数没有任何不同。

（2）友元函数在声明时，前面加以关键字friend，友元函数不是成员函数。

（3）友元函数的作用在于：提高编程的灵活性，但是破坏了数据的封装性和隐藏性。

声明友元函数的格式如下（在类中声明）：

```
friend 类型  函数名（）；
```

注意：

（1）一个函数可以是多个类的友元函数，友元函数的代码可以定义在类内或类外，但**声明必须在类内**。友元函数不属于类，所以不需要在函数名的前面用类名加以限制。

（2）友元函数在对类成员进行访问时，在参数表中需显式地指明访问对象。

（3）**不允许将构造函数、析构函数、虚函数作为友元函数。**

（4）当一个函数需要访问多个类时，友元函数非常有用，普通成员函数只能访问其所属的类，但是多个类的友元函数能访问相应的所有类的数据。

例8.22 在同一个类中访问不同对象之间的私有数据，可以使用一般的函数作为友元函数.

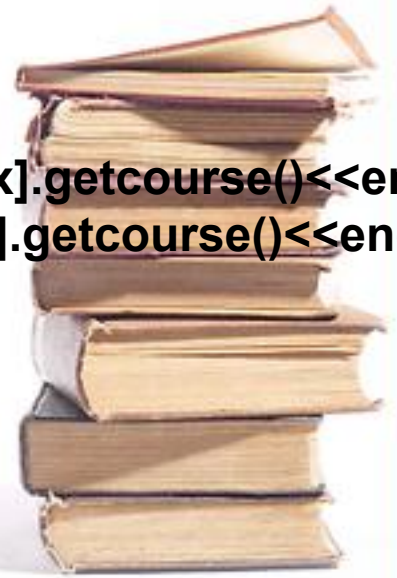
有一个学生类student如下：

```
class  student
{
private:
      char name[10];
      int course;
public:
      student(char *n,int sc){ strcpy(name,n);course=sc;}
      char *getname(){return  name;}
      int  getcourse(){return  course;}
};
```

比较不同对象之间的私有数据，程序设计如下：

```
class  student
{
private:
     char name[10];
     int course;
public:
     student(char *n,int sc){ strcpy(name,n);course=sc;}
     char *getname(){return  name;}
     int  getcourse(){return  course;}
     friend int compare(const student &s1,const student &s2);
};
int compare(const student &s1,const student &s2)
//友元函数在定义时不用类名限制
{
   if(s1.course>s2.course)
      return 1;
   else if(s1.course==s2.course)
      return 0;
   else
      return -1;
}
```

```
void main()
{
        student st[]={student("中华",87),student("孙权",67),
                      student("刘备",80),student("西施",97)};   //对象数组
        int i,min=0,max=0;
        for(i=0;i<4;i++)
        {
           if(compare(st[max],st[i])==-1)
               max=i;
           else if(compare(st[min],st[i])==1)
               min=i;
         }
       cout<<"比较结果："<<endl;
       cout<<"最高分者："<<st[max].getname()<<" "<<st[max].getcourse()<<endl;
       cout<<"最低分者："<<st[min].getname()<<" "<<st[min].getcourse()<<endl;
}
```



8.6.2 友元成员

除了一般的函数可以作为某个类的友元外，一个类的成员函数也可以作为另一个类的友元，这种成员函数不仅可以访问自己所在类对象中的所有成员，还可以访问friend声明语句所在类对象中的所有成员。

例8.24 成员函数作为友元的例子。

```
class Time;  //预先声明
class Date
{
public:
     Date(int month,int day,int year)
          {mm=month;dd=day;yy=year;}
     ~Date() { };
     void showDateTime(const Time &xTime);  //需要利用Time类内的信息
private:
     int mm,dd,yy;
};

class Time
{
public:
     Time(int hour,int minute,int second)
          {hrs=hour; mins=minute; secs=second;}
     ~Time() { };
     friend void Date::showDateTime(const Time &xTime);
//声明类Date的showDateTime()成员函数为类Time的友元函数
private:
     int hrs,mins,secs;
};
```

```
void Date::showDateTime(const Time &xTime)
{
    cout<<"Date:";
    cout<<mm<<"/"<<dd<<"/"<<yy<<endl;
    cout<<"Time:";
    cout<<xTime.hrs<<":"<<xTime.mins<<":"<<xTime.secs<<endl;
}

void main()
{
    Time aTime(10,20,30);
    Date aDate(11,11,99);
    cout<<"Date and Time:"<<endl;
    aDate.showDateTime(aTime);
}
```

8.6.3 友元类

C++中还允许将某个类定义为另一个类的友元，即友元类。

例，把类B声明为类A的友元类的格式是：

在类A的声明中需要声明语句：

```
friend  class B;
```

这就意味着友元类B中的所有成员函数可以访问类A的私有成员。

注意：

B是A的友元类，并不隐含A是B的友元类，友元关系不具有传递性。

例8.25 友元类的例子。 //对应8.24程序

```
#include <iostream>
using namespace std;
class  A
{
      friend  class  friendclass;  //声明friendclass为A的友元类
      char *name;
      int  age;
public:
     A(char *str,int  i){name=str; age=i;}
};
class  friendclass
{
public:
  void  display(A x)   // void  display(const A &x)
   { cout<<"The  man  "<<x.name<<" is "<<x.age<<" years old. "<<endl;  }
};
void  main()
{
      A  demo1 ("Liu",30);
      friendclass  demo2;
      demo2.display(demo1);
}
```

运行结果为：The man Liu is 30 years old.

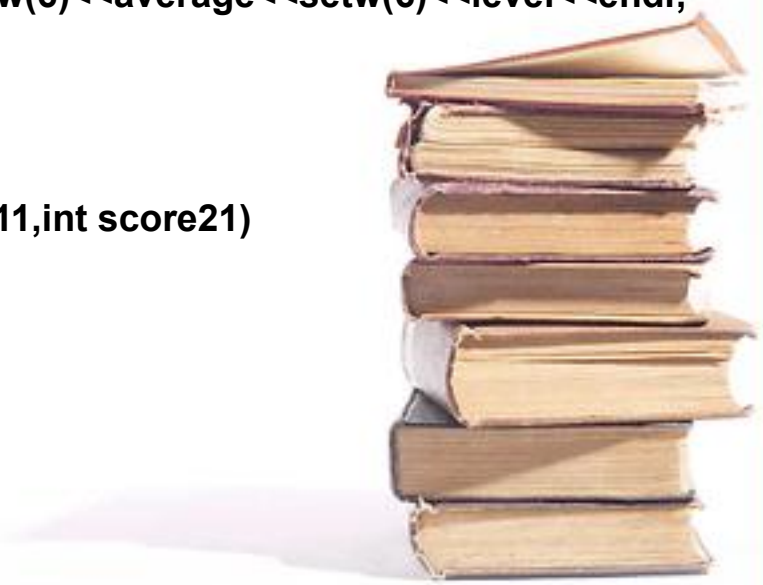
例8.26 有一个学生类Student，有成员学号（num），姓名（name），英语成绩（score1），计算机成绩（score2），平均成绩（average），编写必要的成员函数，求出平均分最高及最低的同学，并输出成绩及对应的等级：大于等于90为优；80~89为良；70~79为中；60~69为及格；小于60不及格。

分析：要把各对象之间的私有数据的平均分进行比较，就必须有一个函数能访问各对象之间的私有数据，这个函数必须声明为类的友元函数。同样识别等级的函数也应该是类的友元函数。

程序设计如下。

```
class Student
{
private:
      char  num[6];//学号
      char  name[10];//姓名
      int  score1;//英语成绩
      int  score2;//计算机成绩
      int  average;  //平均成绩
      char level[8]; //等级
public:
      Student(char num1[],char name1[],int score11,int score21);
      void display() {
        cout<<setw(10)<<num<<setw(10)<<name<<setw(6)<<average<<setw(6)<<level<<endl;
      }
      friend int compare(Student &s1,Student &s2);
      friend void Level(Student &s);
};
Student::Student(char num1[],char name1[],int score11,int score21)
{      strcpy(num,num1);
       strcpy(name,name1);
       score1=score11;
       score2=score21;
       average=(score1+score2)/2;
}
int compare(Student &s1,Student &s2) //一般函数
{              if(s1.average>s2.average)  return  1;
               else if(s1.average==s2.average)  return 0;
               else  return -1;
}
```

```
void Level(Student &s) //一般函数
{          if(s.average>=90)
     strcpy(s.level,"优");
   else if(s.average>=80)
     strcpy(s.level,"良");
   else if(s.average>=70)
     strcpy(s.level,"中");
   else if(s.average>=60)
     strcpy(s.level,"及格");
   else
     strcpy(s.level,"不及格");
}
```



见8.25

```
void main()          //测试函数
{
    Student  stu[]={ Student("3001","AABBBB",80,90),
    Student(“3002”,“BBCCCC”,70,88), Student(“3003”,“CCDDDD”,86,78),
    Student(“3004”,“DDEEEE”,60,78), Student(“3005”,“FFBBBB”,88,92) };  //对象数组
    int i,min=0,max=0;
    for(i=0;i<5;i++)
    {
        if(compare(stu[max],stu[i])==-1)
            max=i;
        else if(compare(stu[min],stu[i])==1)
            min=i;
        Level(stu[i]);
    }
    cout<<"最高分者："<<endl;
    stu[max].display();
    cout<<"最低分者："<<endl;
    stu[min].display();
    cout<<endl;
    cout<<endl;
    for(i=0;i<5;i++)
        stu[i].display();
}
```

8.7 常成员函数　constant

在C++程序设计中，数据隐藏保证了数据的安全性，但由于静态数据成员、友元等数据共享，又不同程序地破坏了数据的安全性，而const在程序运行期间可以有效保护数据，本节主要介绍常对象、常成员函数与常引用。

8.7.1 常对象

类的实例是个对象，当对象被const修饰时，此对象就称为常对象，**常对象的值在整个生存期内不能被改变**。常对象的定义为：

const　类名　对象名;

例如：类定义：

```
class date
{
private:
   int year, month, day;
public:
   date( int y, int m, int d );
   void changeValue() {month=10;day=1;}
   void  print( );
};
```

可以使用以下方法定义类的常对象：

```
const date AA(2012,9,1);
```

注意：

（1）常对象**只能初始化而不能赋值**。

（2）常对象只能调用常成员函数，而不能调用一般的成员函数。

由于对象的值只能通过赋值与成员函数调用二种途径改变。常对象不能赋值，为了不被一般的成员函数改变，规定只能调用常成员函数。

如有定义：

```
const   date    AA(2012,9,1);
date    BB(2012,9,1);
```

对象AA与BB的性质并不相同，简单来说，AA的值不能被改变，BB的值在程序中可以改变。

显然，BB.changeValue( );调用没有什么问题，而AA.changeValue( ) ；调用时编译器就发出错误信息。因为一个成员函数没有把传递给它的this指针指定为const，编译器就不允许const对象调用它。那么如何指定this指针为const？以下学习常成员函数。

error C2662: “date::changeValue”: 不能将“this”指针从“const date”转换为“date &”

8.7.2 常成员函数

使用const关键字修饰的成员函数为常成员函数，常成员函数的声明格式为：

类型说明符 函数明（参数表）const ;

例如，对date类：

```
class date
{private:
   int year, month, day;
   const int a;  //常数据成员
public:
   date( int y, int m, int d,int x ):a(x)   //常数据成员只能通过初始化来获得初值，稍后将学习
        {year=y;month=m;day=d;}
   void changeValue( )
        { month=10;day=1; }
   void  print( ) const
       {cout<<year<<"年"<<month<<"月"<<day<<"日"<<a<<endl;}
};
```

在函数的括号后面加上const，表示这个函数不会改变类的数据成员。常对象因此也就可以放心大胆地调用它了。如果不小心在常成员函数的函数体里修改数据成员时，编译器会有一个错误的报告。

注意：

常成员函数不能更新目的对象的数据成员；

const关键字可以用于对重载函数的区分。例如：

void print( ) const与void print( )完全是两个函数。

下列两种写法含义不同：

类型说明符 函数名（参数表） const ；表示目标对象值不能改变。

const 类型说明符 函数名（参数表）；告诉编译器这个函数的返回值不允许改变。

8.7.3 常数据成员

使用const关键字修饰的数据成员为常数据成员，常数据成员在类中的声明格式为：

const 类型说明符　变量名；

常数据成员只能通过初始化来获得初值，格式为：

构造函数（参数表）：常数据成员（实参）　见p76

{

.....

}

常数据成员表明任何函数都不能对该成员赋值。

例8.26 程序中有常数据成员(const int a;)及常成员函数（void print( )const），通过程序调试，分析常对象AA与一般对象BB在调用成员函数与常成员函数的区别。

```
#include<iostream>
using namespace std;
class date
{
private:
        int year, month, day;
        const int a; //常数据成员
public:
        date( int y, int m, int d,int x ):a(x)  //a获得初始值，初始化列表
            {year=y;month=m;day=d;}   //不能用a=x;
        void changeValue( )
            {month=10;day=1; }
        void  print( ) const  // 常成员函数
            {cout<<year<<"年"<<month<<"月"<<day<<"日"<<a<<endl;}
};

int main()
{
        date BB(2012,8,1,70);
        BB.print( );
        BB.changeValue( );  //可以调用常成员函数
        BB.print( );
        const date AA(2012,10,8,100);  //常对象
        AA.print( ); //常对象只能调用常成员函数
        return 0;
}
```

分析：由于AA为常对象，常对象的值不能改变，因而语句调用AA.changeValue( );有语法错误，而BB.changeValue( );是正确的，对象AA、BB都可以调用常成员函数，因而BB.print( ), AA.print( )都是正确的;从是否是常成员函数来说，语句改为void changeValue( ) const也是错误的。//常对象只能初始化而不能赋值

```
void changeValue( )  const  ?
{
        month=10;
        day=1;
}
// 常成员函数不能改变类的数据成员值
```

程序的运行结果为：

```
2012年8月1日70
2012年10月1日70
2012年10月8日100
Press any key to continue
```

8.7.4 常引用

如果在声明引用时用const修饰，被声明的引用就是常引用，常引用所引用的对象不能被改变，常引用的声明形式为：

```
const 类型说明符 &引用名;
```

常引用可以与常对象相关联，例如**通过常引用访问对象时，此对象就成为常对象**。这表明对于类类型的常引用，不能修改对象的数据成员，也不能调用它的非const成员函数。

例8.27 设类A代表野兔类，类B代表鱼类，它们都有私有数据质量w与价格p，请定义类A与类B，用于比较对象野兔、鱼总价值的友元函数，质量w与价格从测试函数main中输入。

```
class B;
class A {
private:
        double w,p;
public:
        A(double ww, double pp){w=ww;p=pp;}
        friend int com(const A &a,const B &b);  //互为友元函数
};
class B{
private:
        double w,p;
public:
        B(double ww, double pp){w=ww;p=pp;}
        friend int com(const A &a,const B &b); //互为友元函数
};
int com(const A &a,const B &b)   //一般函数
{           if(a.p *a.w > b.p *b.w )
                    return  1;
            else if(a.p *a.w == b.p *b.w )
                    return  0;
            else
                    return  -1;
}
```

```
int main()
{
        double x,y;
         cin>>x>>y;
         A aa(x,y);
         cin>>x>>y;
         B bb(x,y);
         int i=com(aa,bb);
         if(i>0)
                cout<<"A 价值高"<<endl;
         else if(i==0)
                cout<<"A, B价值一样高"<<endl;
         else
                cout<<"B 价值高"<<endl;
         return 0;
}
```

8.8 容器类

当某个类将另一个类的对象作为其成员时，称它为容器类。

例如，建立了一个日期类Date，描述出生日期，此类中有私有成员：Year、Month、Day。当在描述一个人的基本状况时，再定义一个类：Person，它的私有数据有姓名Name、出身年月Date、身高h、体重w。

当在定义Person类时，类中有日期类的对象作为它的成员，因而把Person类叫作容器类。此时可以定义日期类如下。

8-28A:

```
#include<iostream>
#include<string.h>
using namespace std;
class  Date
{
private:
      int  Year;
      int  Month;
      int  Day;
public:
      Date(int y,int m,int d){ Year=y; Month=m; Day=d;}
      void show()
      {
            cout<<"出生年月是"<<Year<<"年"<<Month<<
            "月"<<Day<<“日"<<endl;
       }
};
```

Person类定义如下：

```
class  Person          //Person为容器类
{
private:
      char  Name[8];
      Date   date;    //定义了Date类的对象date
      int h;
      int w;
public:
      Person(char  *n1,int y1,int m1,int d1,int h1,int w1 );
      void  print();
};
```

容器类的构造函数不仅需给它自己的成员赋值，还需给另一个类的对象赋值，它是通过调用对象名称来实现的，其形式为：

容器类构造函数名（参数表）：类中对象（变量表）

```
{
        容器类构造函数体；
}
```

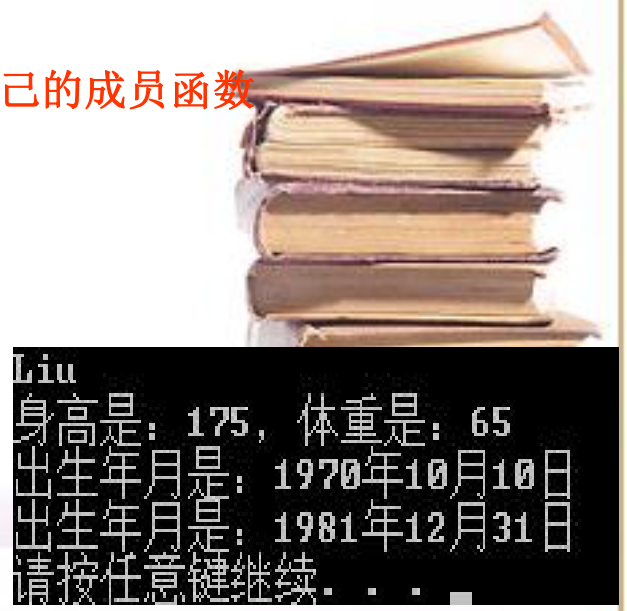
非Date类

Person 类的构造函数如下。

```
Person::Person(char *n1,int y1,int m1,int d1,int h1,int w1):date(y1,m1,d1)
//调用对象名称实现赋值，不能以赋值的形式完成
{
      strcpy(Name,n1);
      h=h1;
      w=w1;
}
void  Person::print()
{
      cout<<Name<<endl;
      cout<<"身高是"<<h<<" 体重是 "<<w<<endl;
      date.show( );     //通过Date的对象date调用自己的成员函数
}
```

用下列的main函数来测试容器类。

```
int  main()
{
          Person man("Liu", 1970,10,10,175,65);
          Date d(1981,12,31);
          man.print();
          d.show();
          return 0;
}
```



8.9　类与结构体*

C++中的结构体可以像“类”一样具有成员函数。

在缺省情况下，结构内的数据和函数是公有的，而类中的数据和函数是私有的。

例8.29 结构体类型中成员函数的使用。

```
#include <iostream>
#include <string.h>
using namespace std;
struct  person
{
   void init() { age = 30; strcpy(name,“Liu”); }  //未用构造函数。
   void display() {  cout << name <<" is "<< age <<" years old."<<endl;}
private :
   int age;
   char name[10];
};

void main()
{
   person  demo;
   demo.init();
   demo.display();
}
```

例8.30 结构体类型中成员函数的使用。 对应8.29程序

```
#include<iostream>
using namespace std;
struct  Location
{
private:
      int x,y;
public:
      void init( int x0,int y0);  //未用构造函数。
      int  Getx();
      int  Gety();
} ;
void  Location::init(int x0,int y0)
{
  x=x0;
  y=y0;
}
int Location::Getx(){ return x ;}
int Location::Gety(){ return y ;}
void main()
{
      Location  A;
      A.init(10,5);
      cout<<A.Getx() <<endl<<A.Gety() <<endl;
}
```

例8.31结构体类型中构造函数与析构函数的使用。 对应8.30程序

```
#include<iostream.h>
#define size 100
struct stack
{
  int stack1[size],tos;
public:
stack();                    //构造函数原型
~stack();                   //析构函数原型
 void push(int i);
 int pop();
};
stack::stack()              //构造函数实现定义
{tos=0;cout<<"stack init\n";}
stack::~stack()             //析构函数实现定义
{cout<<"stack destroyed \n";}

void stack::push(int i)
{if (tos==size) {
    cout<<"stack is full";return;}
stack1[tos]=i; tos++;
}
int stack::pop()
{if (tos==0)
{
   cout<<"stack underflow";
   return 0;
}
   tos--;
   return stack1[tos];
}
```

```
void main()
{  stack a,b;
   a.push(1);b.push(2);
cout<<a.pop()<<" ";
   cout<<b.pop()<<"\n";
   cout<<"exiting main\n";
}
```

程序运行结果为：

```
stack init
stack init
1 2
exiting main
stack destroyed
stack destroyed
```

8.10　对象数组与对象指针

对象数组的元素都是对象，不仅具有数据成员，而且还有成员函数。前面已经简单使用过对象数组。本节将重点介绍它的特殊之处。

定义一个一维对象数组的格式：

类名 数组名[下标表达式];

在使用时，要引用数组元素，也就是一个对象，通过这个对象，也可以访问它的公有成员，一般形式是：

数组名[下标].成员名;

例8.32 一个对象数组的例子。 见8.31程序

```
class  User
{
private:
        char  UserName[20];
        char  Pass[10];
public:
        void init(char u[],char p[])
            {strcpy(UserName,u);strcpy(Pass,p);}
        char *GetUserName()
            {return UserName; }
        char *GetPass()
            { return  Pass;  }
};
void main()
{
        User ob[5];  // 对象数组
        char Un[20];
        char ps[10];
        for(int i=0;i<5;i++)
        {
            cin>>Un>>ps;
            ob[i].init(Un,ps);
        }
        for(int j=0;j<5;j++)
            cout<<ob[j].GetUserName() <<" "<<ob[j].GetPass()<<endl;
}
```

8.10.2 指向类对象的指针

指向类对象的指针的定义形式为：

类名　*指针名；

其指针的赋值形式为：

指针名=&类对象；

例8.33 以下程序段所定义的对象指针p是指向对象obj的。见8.32

```
#include<iostream>
using namespace std;
class  myclass
{
        int i;
  public:
        myclass(int x) {i=x;}
        void  show();
};
void myclass::show()
{  cout<<i<<"\n";  }

void main()
{
        myclass  obj(10),*p;
        p=&obj;
        p->show();   // obj.show();
}
```

程序输出结果为10。

例8.34 指向类对象指针的应用举例。

```
class test
{
private:
        int num;
        float f1;
public:
        test();
        test(int n,float f);  //构造函数重载
        ~test();
        void show();
};

test::test()
{       num=0;
        f1=0.0;
        cout<<"Default constructor for object"<<endl;
        cout<<"num="<<num<<" "<<"f1="<<f1<<endl;}

test::test(int n,float f)
{       num=n;
        f1=f;
        cout<<"Constructor for object"<<endl;
        cout<<"num="<<num<<" "<<"f1="<<f1<<endl;}
```

```
test::~test()
{
        cout<<"Destructor is active and num="<<num<<endl;
}
void  test::show(void)
{
        cout<<"num="<<num<<" "<<"f1="<<f1<<endl;
}

void main()
{
        test *ptr1=new test;
        test *ptr2=new test(1,1.5);
        ptr1->show();
        ptr2->show();
        delete ptr2;
        delete ptr1;
}
```

程序运行的结果为：

```
Default constructor for object
num=0 f1=0
Constructor for object
num=1 f1=1.5
num=0 f1=0
num=1 f1=1.5
Destructor is active and num=1
Destructor is active and num=0
```

8.10.3 指向类成员的指针

指向类成员的指针既包括指向类成员变量的指针，又包含指向成员函数的指针。

(1) 指向类成员的指针

指向类成员的指针的定义形式为：

类成员的类型　类名:: *指针名；

其指针的赋值形式为：

指针名=&类名:: 类成员变量；

例8.35 指向类成员的指针应用举例。 见8.34程序

```
class myclass
{public:
        int i;     //公有变量
        float j;
        myclass(int x, float y) {i=x;j=y;}
        void show() {cout<<i<<" "<<j;}
        void func(int x, float y);
};
void myclass::func(int x, float y)
{  i=x+5;
   j=y-1;
   if ((i-j)>0)
      cout<<"ok \n";
   cout<<"error \n";
}
void  main()
{           int  myclass::*p1;      //定义指向类myclass的整型成员变量的指针p1
            float  myclass::*p2 ;   //定义指向类myclass的实型成员变量的指针p2
            p1=&myclass::i;         //指针的赋值
            p2=&myclass::j;
            myclass xx(1,2.5);
            myclass *px=&xx;     //定义一个指向类的指针px
            px->*p1=5;   //公有变量方可这样
            px->*p2=1.32;
            cout<<"i="<<xx.i<< "   j=" <<xx.j<<endl;
}
```

程序运行结果为：

i=5 j=1.32

例8.36　指向静态成员的指针的使用。　见8.35程序

程序运行结果为：

```
10
10
10
10
10
```

```
class myclass
{public :
      static int num;  //公有变量
      //定义静态变量，须在类外全局域中定义一下，也就是初始化
};
int myclass::num;  // 缺省时初始化为0。
void main()
{
      int *p=&myclass::num;  //定义并赋初值
      *p=10;
      cout<< myclass::num<<endl;
      cout<<*p<<endl;
      myclass x,y;
      cout<<x.num<<endl;  //只有num为公有变量才能如此调用
      cout<<y.num<<endl;
      cout<<myclass::num<<endl;
}
```

(2). 指向成员函数的指针

指向成员函数的指针的定义形式为：

函数返回值数据类型（类名:: *指向成员函数的指针名）（参数列表类型）

赋值方式为：

指针名=&类名:: 成员函数名；//注意同一般函数指针的区别

调用方式为：

（对象名或指向对象的指针名.*指向成员函数的指针名）（参数表）；

（指向对象的指针名—>*指向成员函数的指针名）（参数表）；

例8.37 指向成员函数的指针举例。 见8.36程序

```
class myclass
{
        int i;
        float j;
public:
        myclass(int x,float y) {i=x;j=y;}
        void show() {cout<<i<<" "<<j<<endl;}
        void func(int x,float y);
};
void myclass::func(int x,float y)
{       i=x+5;
        j=y-1;
        if((i-j)>0)cout<<"ok "<<endl;
        cout<<"error "<<endl;
}
void  main()
{   void (myclass::*p1)();                  //定义指向成员函数show()的指针
    void (myclass::*p2) (int,float );       //定义指向成员函数func()的指针
    p1=&myclass::show;                      //指针的赋值
    p2=&myclass::func;
    myclass obj(5,1.25);
     (obj.*p1)();                           //指向成员函数指针的调用
     (obj.*p2)(3,2.5);
}
```

程序的执行结果为：

```
5   1.25
    ok
    error
```

8.10.4 this指针

C++提供了一个特殊的对象指针—this指针。它是由C++编译器自动产生且较常用的一个隐含指针，隐含于每个类的成员函数中。类成员函数使用该指针来处理对象。

当某一对象调用一个成员函数时，指向产生这个调用的对象的指针将作为一个变元自动传给该函数，这个指针就是this指针。

例8.38 隐含this指针的例子。

```
#include <iostream>
using namespace std;
class myclass
{
        int i;
        float j;
public:
        void show()
        {
           cout<<i<<" "<<j<<endl;     //与cout<<this->i<<" "<<this->j<<endl;一样
        }
        void func(int x, float y);
};
void myclass::func(int x, float y)
{
        i=x;   //与this->i=x;一样
        j=y;   //与this->j=y;一样
}
void main()
{
        myclass obj;
        obj.func(3,2.5);
        obj.show();
}
```

this指针的理解：

当定义类myclass的对象obj后，调用成员函数：

obj. func(3,2.5);

那么C++编译器如何确定func（）的操作对象是obj，而不是程序中的其他对象？

实际上C++编译器在调用func（）函数时转换成如下格式：

func（&obj，3，2.5）；

此时，C++编译器把类的对象的地址作为参数传递给函数。尽管在书写函数的原型时没有这个形式参数，但程序编译时被C++转换成如下格式：

func（myclass *this，int x，float y）；

编译器所解释的成员函数func（int x，float y）的形式为：

```
void func(myclass *this，int x，float y)
{
        this->i=x;
        this->j=y;
}
```

因而隐式指针this保证了成员函数func（）的操作对象确实是类的对象obj。只是在一般情况下，并不直接写出this，而采取缺省设置。类的对象对其他成员函数的调用也一样。

初始化列表：

初始化类的成员有两种方式，一是使用初始化列表，二是在构造函数体内进行赋值操作。初始化列表以冒号开头，后跟一系列以逗号分隔的初始化字段。

主要是性能问题，对于内置类型，如int, float等，使用初始化类表和在构造函数体内初始化差别不是很大，但是对于类类型来说，最好使用初始化列表，为什么呢？使用初始化列表少了一次调用默认构造函数的过程，这对于数据密集型的类来说，是非常高效的。

```
class Animal
{public:
    Animal(int weight,int height): m_weight(weight), m_height(height) {  }
    //初始化列表
//  Animal(int weight,int height) { m_weight = weight;m_height = height; }
    //函数体内初始化
private:
    int m_weight;
    int m_height;
};
```

初始化列表：

```
class date
{
private:
        int year, month, day;
        const int a; //常数据成员
public:
        date( int y, int m, int d,int x ):a(x)  //a获得初始值
          {year=y;month=m;day=d;}   //另一种形式
        void changeValue( )
          {month=10;day=1; }
        void  print( )  const  // 常成员函数
          {cout<<year<<"年"<<month<<"月"<<day<<"日"<<a<<endl;}
};
```

见class 8\exer_1

总结

- 构造函数（含拷贝）、析构函数的特点与功能。
- 类的静态成员（静态数据成员和静态成员函数）
- 类的友元（友元函数、友元成员、友元类）
- 常成员函数（常对象、常成员函数、常数据成员、常引用）
- 容器类或组合类
- 类与结构
- 对象数组与对象指针（指向类对象的指针、指向类成员的指针）
- this指针

上机安排：实验九

实验九讲解安排

任务序号	讲解人
11-3	陈志豪
11-5	丁海洋
11-6	李俊儒

考核办法：

- 作业讲解占20%，目的：倡导学生独立完成作业，提升个人编写程序与调试程度的能力，要求完成讲解PPT，介绍要解决的问题并进行分析，编写源程序、调试并执行程序。
- 上机考试成绩占30%，第15周进行。上机分4道题，必须成功通过调试1题或以上，才能进入大作业答辩环节。
- 大作业及答辩占50%，要求：

◆ 程序总行数400行以上（达不到400行计0分）；

◆ 程序总共有五个以上菜单功能模块可选择，每个菜单解决一个问题；

◆ 每个菜单功能模块含有不同类、构造函数定义及使用、拷贝构造函数；

◆ 至少有一个菜单功能模块含有友元函数、友元类；

◆ 多重继承、虚基类、虚函数或抽象类的应用；

◆ 运算符重载，至少有一个菜单功能模块含有异常处理；

◆ 程序构成一个整体，且具有实际应用背景。。