第一章：面向对象程序设计概述

[1_1]什么是面向对象程序设计？

面向对象程序设计是一种新型的程序设计范型。这种范型的主要特征是：

程序=对象+消息。

面向对象程序的基本元素是对象，面向对象程序的主要结构特点是：第一：程序一般由类的定义和类的使用两部分组成，在主程序中定义各对象并规定它们之间传递消息的规律。第二：程序中的一切操作都是通过向对象发送消息来实现的，对象接受到消息后，启动有关方法完成相应的操作。

面向对象程序设计方法模拟人类习惯的解题方法，代表了计算机程序设计新颖的思维方式。这种方法的提出是软件开发方法的一场革命，是目前解决软件开发面临困难的最有希望、最有前途的方法之一。

[1_2]什么是类？什么是对象？对象与类的关系是什么？

在面向对象程序设计中，对象是描述其属性的数据以及对这些数据施加的一组操作封装在一起构成的统一体。对象可以认为是：数据+操作

在面向对象程序设计中，类就是具有相同的数据和相同的操作的一组对象的集合，也就是说，类是对具有相同数据结构和相同操作的一类对象的描述。

类和对象之间的关系是抽象和具体的关系。类是多个对象进行综合抽象的结果，一个对象是类的一个实例。

在面向对象程序设计中，总是先声明类，再由类生成对象。类是建立对象的"模板"，按照这个模板所建立的一个个具体的对象，就是类的实际例子，通常称为实例。

[1_3]现实世界中的对象有哪些特征？请举例说明。

对象是现实世界中的一个实体，其具有以下一些特征：

（1）每一个对象必须有一个名字以区别于其他对象。

（2）需要用属性来描述它的某些特性。

（3）有一组操作，每一个操作决定了对象的一种行为。

（4）对象的操作可以分为两类：一类是自身所承受的操作，一类是施加于其他对象的操作。例如：雇员刘名是一个对象

对象名：刘名

对象的属性：

年龄：36　生日：1966.10.1　工资：2000　部门：人事部

对象的操作：吃饭　开车

[1_4]什么是消息？消息具有什么性质？

在面向对象程序设计中，一个对象向另一个对象发出的请求被称为"消息"。当对象接收到发向它的消息时，就调用有关的方法，执行相应的操作。消息是一个对象要求另一个对象执行某个操作的规格的说明，通过消息传递才能完成对象之间的相互请求或相互协作。消息具有以下 3 个性质：

（1）同一个对象可以接收不同形式的多个消息，做出不同的响应。

（2）相同形式的消息可以传递给不同的对象，所做出的响应可以是不同的。

（3）消息的发送可以不考虑具体的接收者，对象可以响应消息，也可以不响应。

[1_5]什么是方法？消息和方法的关系是什么？

在面向对象程序设计中，要求某一对象作某一操作时，就向该对象发送一个响应的消息，当对象接收到发向它的消息时，就调用有关的方法，执行响应的操作。方法就是对象所能执行的操作。方法包括界面和方法体两部分。方法的界面也就是消息的模式，它给出了方法的调用协议；方法体则是实现某种操作的一系列计算步骤，也就是一段程序。在 C++语言中方法是通过函数来实现的，称为成员函数。消息和方法的关系是：对象根据接收到的消息，调用相应的方法；反过来，

[1_6]什么是封装和抽象？请举例说明。

在现实世界中，所谓封装就是把某个事物包围起来，使外界不知道该事物的具体内容。在面向对象程序设计中，封装是指把数据和实现操作的代码集中起来放在对象内部，并尽可能隐蔽对象的内部细节。对象好象是一个不透明的黑盒子，表示对象属性的数据和实现各个操作的代码都被封装在黑盒子里，从外面是看不见的，更不能从外面直接访问或修改这些数据及代码。使用一个对象的时候，只需要知道它向外界提供的接口形式而无需知道它的数据结构细节和实现操作的算法。封装机制可以将对象的使用者与设计者分开，使用者不必知道对象行为实现的细节，只需要使用设计者提供的接口让对象去做。

抽象是人类认识问题的最基本的手段之一。它忽略了一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。抽象是对复杂世界的简单表示，抽象强调感兴趣的信息，忽略了不重要的信息。例如，设计一个学籍管理程序的过程中，考察某个学生对象时，只关心他的姓名、学好、成绩等，而对他的身高、体重等信息就可以忽略。以一般观点而言，抽象是通过特定的实例（对象）抽象共同性质以后形成概念的过程。抽象是对系统的简化描述或规范说明，它强调了系统中的一部分细节和特性，而忽略了其他部分。抽象包括两个方面：数据抽象和代码抽象（或称为行为抽象）。前者描述某类对象的属性或状况，也就是此类对象区别于彼类对象的特征物理量；后者描述了某类对象的共同行为特征或具有的共同操作。

在面向对象程序设计方法中，对一个具体问题的抽象分析的结果，是通过类来描述和实现的。现在以学生管理程序为例，通过对学生进行归纳、分析，抽取出其中的共性，可以得到如下的抽象描述：

共同的属性：姓名、学号、成绩等，他们组成了学生数据抽象部分。用C++语言的数据成员来表示，可以是：

共同的行为：数据录入、数据修改和数据输出等，这构成了学生的行为抽象部分，用 C++语言的成员函数表示，可以是：input();modify();print(); 如果我们开发一个学生健康档案程序，所关心的特征就有所不同了。可见，即使对同一个研究对象，由于所研究问题的侧重点不同，就可能产生不同的抽象结果。

[1_7]什么是继承？请举例说明。

继承所表达的是对象类之间的相关关系，这种关系使得某类对象可以继承另一类对象的特征和能力。现实生活中，继承是很普遍和容易理解的。例如我们继承了父母的一些特征，如种族、血型、眼睛的颜色等，父母是我们所具有的属性的基础。继承所表达的是对象之间相关的关系，这种关系使得某一类可以继承另一个类的特征和能力。

[1_8]若类之间具有继承关系，则它们之间具有什么特征？

（1）类间具有共享特征（包括数据和操作代码的共享）

（2）类间具有差别或新增部分（包括非共享的数据和代码操作）

（3）类间具有层次结构

假设有两个类 A 和 B，若类 B 继承类 A，则类 B 包含了类 A 的特征（包括数据和操作），同时也可以加入自己所特有的新特性。这时，我们称被继承类 A 为基类或父类或超类；而称继承类 B 为 A 类的派生类或子类。同时，我们还可以说，类 B 是从类 A 中派生出来的。

[1_9]什么是单继承、多继承？请举例说明。

从继承源上分，继承分为单继承和多继承。单继承是指每个派生类只直接继承了一个基类的特征。多继承是指多个基类派生出一个派生类的继承关系。多继承的派生类直接继承了不止一个基类的特征。例如：小孩的玩具车继承了车的一些特性，还继承了玩具的一些特征。

[1_10]什么是多态性？举例说明。

多态性也是面向对象程序设计的重要特性，它是指不同的对象收到相同的消息时产生不同的行为方式。例如我们同样双击 windows 系统桌面上的图标时，有的是打开多媒体播放器，有的是打开资源管理器。利用多态性，用户只需发送一般形式的消息，而将所有的实现留给接收消息的对象。对象根据所收到的消息做出相应的动作。

[1_11]什么是函数重载和运算符重载？为什么要使用重载？

重载一般包括函数重载和运算符重载。函数重载是指一个表示符可同时用于为多个函数命名，而运算符重载是指一个运算符可同时用于多种运算。也就是说，相同名字的函数或运算符在不同的场合可以表现出不同的行为。

使用重载的目的是为了更好地表达行为共享，这种行为共享就象将相似的操作划分在一起。使用重载可以使程序员在只知道操作的一般含义，而不知道操作的具体细节的情况下能正确地对某个对象使用一个操作。另外，使用重载的直接益处是减少了程序员记忆操作的名字的负担。

第二章：C++基础

[2_1]简述 C++的主要特点。

（1）C++保持与 C 的兼容，用 C 编写的软件可以用到 C++中。

（2）用 C++编写的程序可读性好，代码结构更合理，可直接地在程序中映射问题空间的结构。

（3）生成代码的质量高。

（4）软件的可重用性、可扩充性、可维护性和可靠性有了明显的提高，从而节省了开发费用和时间。

（5）支持面向对象的机制，可方便地构造出模拟现实问题的实体和操作。

[2_2]下面是一个 C 程序，改写它，使它采用 C++风格的 i/o 语句

改写如下：

```
#include <iostream.h>

main()

{  int a,b,d,min;

cout<<"enter two numbers: ";

cin>>a;

cin>>b;

min=a>b?b:a;

for(d=2;d<min;d++)

if((a%b)==0)&&((b%d)==0)) break;

if(d==min)

{  cout<<"no common denominators\n";

return 0;

   }

   cout<<"the lowest common denominator is "<<endl<<d;

   return 0;

}
```

[2_3]测试下面的注释是否有效？

此注释有效，单行注释中可以嵌套/*.....*/方式的注释。

[2_4]以下简单的C++程序不可能编译通过，为什么？

原因是：在程序中，当一个函数的定义在后，而对它的调用在前时，必须将该函数的原型写在调用语句之前，而在本程序中缺少函数原型语句。在语句：#include <iostream.h>后加上语句sum(int a,int b);就可以通过了。

[2_5]（1）答：这两个函数原形是等价的，因为函数原型中的参数名可以缺省。

（2）答：这两个函数的第一行是不等价的，函数的第一行中必须包含参数名。

（3）答：这两个函数原型是等价的，因为在函数原型中未注明参数，C++认为该函数的参数表为空（void）

[2_6]答：输出结果为：10 20 因为 f 函数的参数是引用，所以修改 k 的值有效。函数调用后，主函数中 k 的值变为 10。由于 m 是对函数的引用，当 m 被赋值为 20 时，k 的值也变为 20。

[2_7] 举例说明可以使用 const 替代#define 以消除#define 的不安全性

答：例如：#include <iostream.h>

```
#define A 2+4
#define B A*3
void main()
{ cout<<B<<endl; }
```

上面程序的运行结果是 14 而不是 18，但很容易被认为是 18。用 const 替代#define 就能得到正确结果，从而消除了#define 的不安全性。

```
#include <iostream.h>
const A=2+4;
const B=A*3;
void main()
{ cout<<B<<endl; }
```

运行结果为 18。

[2_8]答：使用内联函数的优点主要有两个：一是能加快代码的执行，减少调用开销；二是能消除宏定义的不安全性。

[2_9] 用动态分配空间的方法计算 Fibonacci 数列的前 20 项并存储到动态分配的空间中。

答：#include <iostream.h>

```
void main()
{  int I,*p=new int[20];//动态分配 20 个整型空间
   *p=1;
*(p+1)=1;//前面两个空间赋值 1
cout<<*p<<"\t"<<*(p+1)<<"\t";
p=p+2;//p 指向第三个空间
for(i=3;i<=20;i++)
{  *p=*(p-1)+*(p-2);
   cout<<*p<<"\t";
   if(i%5==0) cout<<endl;
   p++;//指向下一个空间
}
}
```

结果：1　　1　　2　　3　　5

8　　13　　21　　34　　55

89　　144　　233　　377　　610

987　　1597　　2584　　4181　　6765

[2_10] 建立一个被称为 sroot()的函数，返回其参数的二次方根。重载 sroot()三次，让它返回整数、长整数与双精度数的二次方根（计算二次方根时，可以使用标准库函数 sqrt()）

```
#include <iostream.h>
#include <math.h>
int sroot(int );long sroot(long);double sroot(double);
double sqrt();//声明开方函数 sqrt()
void main()
```

```
{ int i,x;long l,y;double d,z;
  cin>>i;  cin>>l;  cin>>d;
  x=sroot(i);  y=sroot(l);  z=sroot(d);
  cout<<x<<"\t"<<y<<"\t"<<z<<endl;
}
int sroot(int i)
{ return sqrt(i); } //i 是整数
long sroot(long l)
{ return sqrt(l); } //l 是长整型
double sroot(double d)
{ return sqrt(d); } //d 是双精度
//敲进 9 16 25
//输出 3 4 5
```

习题[2_11] 编写C++风格的程序，解决百钱问题，将一元人民币兑换成1、2、5分的硬币，有多少种换法？

```
#include <iostream.h>
void main()
{  int i,j,sum=0;
  for(i=0;i<=20;i++)
    for(j=0;j<=50;j++)
      if(100-5*i-2*j>=0)
        {
sum++;
          cout<<100-5*i-2*j<<"\t"<<j<<"\t"<<i<<endl;
}
```

```
cout<<"sum is "<<sum<<endl;
}
```

习题[2_12] 编写 C++风格的程序，用二分法求解 f(x)=0 的根

```
#include <iostream.h>
#include <math.h>
inline float f(float x)
{  return 2*x*x*x-4*x*x+3*x-6;  }
void main()
{  float left,right,middle,ym,yl,yr;
   cout<<"pleass two number:"<<endl;//接受输入，确定第一组数据区域
   cin>>left>>right;
   yl=f(left);
   yr=f(right);
do
{  middle=(right+left)/2;
   ym=f(middle);
   if(yr*ym>0)
   {  right=middle;
      Yr=ym;
   }
else
{  left=middle;
   yl=ym;
    }
}while(fabs(ym)>=1e-6);
```

cout<<"\nRoot is:"<<middle<<endl;

}

本例使用了内联函数 f(x),因为在主函数中多次调用它，这样可以加快代码执行的速度。敲进两个数：-10 10 结果：Root is 2

[2_13]答：运行结果是：2 4 6 12 10 说明：本例使用的是返回引用的值，index(3)=12;语句的执行实际将 a[3]赋值为 12。

[2_14]答：运行结果为：101 说明：在语句::i=i+1;中赋值号左边::i 的 i 单元是全局变量，赋值号右边的 i 单元是局部变量 i。所以执行该语句的结果是

将局部变量 i 的值+1（101）赋值给全局变量 i

[2_15]答：结果是：10 10 说明：函数 f(int &a,b)中的第一个参数是引用，引用参数是一种按地址传递参数的方法，对其的调用是传地址调用；而第二个参数是变量参数，对它的调用是通常的传值调用。所以运行后，a 的值被改为 10，b 的值不变，仍为 10

[2_16]答：D

说明：int *p=new int(10);表示分配 1 个整型空间，初值为 10

int *p=new int[10];表示分配 10 个整型空间

int *p=new int;表示分配 1 个整型空间

int *p=new int[10](0)想给一个数组分配内存空间时，对整个数组进行初始化，这是不允许的。

[2_17]答：D 说明：name 被定义为指向常量的常指针，所以它所指的内容和本身的内容都不能修改，而 name[3]='a';修改了 name 所指的

常量，name='lin';和 name=new char[5];修改了常指针，只有

D 输出一个字符是正确的。

[2_18]答：A 说明：name 被定义指向常量的指针，这是一个不能移动的固定指针，它所指的内容不能改变，但指针所指的数据可以改变，而 name[3]='q';修改

了 name 所指的内容，都是错误的；name="lin"; name=new char[5]; name=new char('q');以不同的方法修改了常指针，都是错误的。

[2_19]答：A 说明：name 被定义指向常量的指针，不允许改变指针所指的常量，但指针本身的内容可以修改，而 name[3]='q';修改了 name 所指的内容，是错误的。name=="lin" name=new char[5];和 name=new char('q')以不同的方法修改了常指针，都是正确的。

[2_20]答：D 说明：C++中不能建立引用数组和指向引用的指针，也不能建立引用的引用。所以 A、B、C 是错误的，D 是正确的。

第三章：类和对象（一）

[3_1]答：类声明的一般格式如下：

```
class 类名
{  public:
     公有数据成员；
     公有成员函数；
   protected:
     保护数据成员；
     保护成员函数；
   private:
     私有数据成员；
     私有成员函数；
};
```

其中：class 是声明类的关键字；类名是要声明的类的名字；后面的花括号表示出类声明的范围；最后的分号表示类声明结束。

[3_2]答：构造函数是一种特殊的成员函数，它主要用于为对象分配空间，进行初始化。构造函数具有一些特殊的性质：

（1）构造函数的名字必须与类名相同。

（2）构造函数可以有任意类型的参数，但不能指定返回类型。它有隐含的返回值，该值在系统内部使用。

（3）构造函数是特殊的成员函数，函数体可写在类体内，也可写在类体外。

（4）构造函数可以重载，即一个类中可以定义多个参数个数或参数类型不同的构造函数。

（5）构造函数被声明为公有函数，但它不能象其它成员函数那样被显示地调用，它是在定义对象的同时被调用的。

析构函数也是一种特殊的成员函数。它执行与构造函数相反的操作，通常用于撤消对象时的一些清理任务，如释放分配给对象的内存空间等。析构函数有以下一些特点：

（1）析构函数与构造函数名字相同，但它前面必须加一个波浪号（~）

（2）析构函数没有参数，不能指定返回类型，而且不能重载。因此在一个类中只能有一个析构函数。

（3）当撤消对象时，编译系统会自动地调用析构函数。

[3_3]答：B　说明：C++中对构造函数有一些规定：不能带返回值；可以不带参数；也可以缺省定义；但构造函数的名字与类名必须完全相同。

[3_4]答：C　说明：C++中没有限定 private、public、protected 的书写次序。但是，不能在类的声明中给数据成员赋初值，数据成员的数据类型也不能是 register（寄存器类型），没有用 private、public、protected 定义的数据成员是私有成员。

[3_5]答：C　说明：C++中对析构函数也有一些规定：没有参数；不能重载；析构函数的名字是在类名前加"~"；析构函数不能指定返回类型。

[3_6]答：B　说明：构造函数的工作是在创建对象时执行的。

[3_7]答：语句"p1.age=30;"出现错误。因为 age 是私有数据成员，不能直接访问。

[3_8]答：第 1 个错误：printStu()、setStu()两个成员函数没有用 public 定义，则不允许外部函数对对象进行操作。

第 2 个错误：成员函数在类外定义，应加上类名"Student::"。

第 3 个错误：setAge 应在类中说明，并且在类外定义时，应加上类名"Student::"。

[3_9]答：语句"Point epoint;"是错误的，它试图用私有的构造函数 Point 访问公有数据成员 x 和 y，这是不对的。

[3_10]答：语句 Stack stt;"应该带参数，因为当类中没有定义构造函数时，编译器会自动生成一个缺省的不带参数的构造函数。但是，如果类中有自己定义的构造函数后，编译器将不再自动生成一个缺省的构造函数，例如：将上述语句改成"Stack stt(10);"就正确了。

[3_11]:下面是一个计数器的定义，请完成该类成员函数的实现

```
#include <iostream.h>
class counter
{  public:
    counter(int number);//构造函数
    void increment();  //给原值加 1
    void decrement();  ///给原值减 1
    int getvalue();    //取得计数器值
    int print();       //显示计数
  private:
    int value;
};
counter::counter(int number)//构造函数定义
{  value=number;  }
void counter::increment()//给原值加 1
```

```
{  value++;  }
void counter::decrement()//给原值减 1
{  value--;  }
int counter::getvalue()//取得计数器值
{  return value;  }
int counter::print()//显示计数
{  cout<<"value is "<<value<<endl;
   return 0;
}
main()
{  int i;
   cin>>i;
   counter a(0);
   for(int j=0;j<i;j++)
   {  a.increment();
      a.getvalue();
      a.print();
   }
   counter b(10);
   for(int k=1;k<i;k++)
   {  b.decrement();
      b.getvalue();
      b.print();
   }
   return 0;
```

}

习题：[3_12]根据注释语句的提示，实现类 Date 的成员函数

```
#include <iostream.h>
class Date
{  public:
     void printDate();  //显示日期
       void setDay(int d);//设置日期值
       void setMonth(int m);//设置月的值
       void setYear(int y);//设置年的值
private:
      int day,month,year;
};
void main()
{  Date testDay;
   testDay.setDay(5);
   testDay.setMonth(10);
   testDay.setYear(2003);
   testDay.printDate();
}
void Date::printDate()
{  cout<<"\nDate is "<<year<<".";
   cout<<month<<"."<<day<<endl;
}
void Date::setDay(int d)
{  day=d;  }
```

```
void Date::setMonth(int m)
{  month=m;  }
void Date::setYear(int y)
{  year=y;  }
```

习题：[3_13]下面定义了一个类 date,根据主程序的提示，实现重载构造函数 date ()

```
#include <iostream.h>
#include <stdio.h>
class date
{  public:
     date(int d,int m,int y);
       date::date();
       void show();
private:
     int day,month,year;
};
void date::show()
{  cout<<day<<'/'<<month<<'/';
  cout<<year<<"\n";
}
main()
{  date idate(28,10,1949);//构造函数的参数为 3 个整数
  idate.show();
  date indate; //构造函数没有参数，数据通过键盘直接输入
  indate.show();
```

```
    return 0;
}
//解：重载构造函数的实现如下：
date::date(int d,int m,int y)
{   day=d;
    month=m;
    year=y;
}
date::date()
{   cout<<"Enter month_day_year:\n";
    cin>>day;
    cin>>month;
    cin>>year;
}//注意：敲数据时要如：8 回车 9 回车 2005 回车
```

习题：[3_14]建立类 cylinder，cylinder 的构造函数被传递了两个 double 值，分别表示圆柱体的半径和高度，用类 cylinder 计算圆柱体的体积，并存储在一个 double 变量中。在类 cylinder 中包含一个成员函数 vol()，用来显示每个 cylinder 对象的体积。

```
#include <iostream.h>
class cylinder
{   public:
     cylinder(double a,double b);
       void vol();
private:
      double r,h;
```

```
    double volume;
};
cylinder::cylinder(double a,double b)
{  r=a; h=b;
   volume=3.141592*r*r*h;
}
void cylinder::vol()
{  cout<<"volume is: "<<volume<<"\n";  }
void main()
{
    cylinder x(2.2,8.09);
   x.vol();
}
```

习题：[3_15]建立一个 Stock 类，含有股票代码和股票现价两个数据成员。用 new 自动为 Stock 类的对象分配内存，并将股票代码"600001"，现价 8.89 存入内存的相应域中。

```
#include <iostream.h>
#include <string.h>
class Stock
{ public:
   void set(char *c,float pr);
    void print();
private:
    char Stockcode[7];
    float price;
```

```
};
void Stock::set(char *c,float pr)
{  strcpy(Stockcode,c);
   price=pr;
}
void Stock::print()
{  cout<<Stockcode<<":  "<<price;
   cout<<"\n";
}
main()
{  Stock *p;
   p=new Stock;    //为对象分配空间
   if(!p)          //判断分配是否成功
   {  cout<<"Allocation error.";
      return 1;
   }
   p->set("600001",8.89);//为对象赋值
   p->print();     //显示对象
   delete p;
   return 0;
}
```

习题：[3_16]声明一个栈类，利用栈操作实现将输入字符串反向输出的功能

```
#include <iostream.h>
//#include <iomanip.h>
//#include <ctype.h>
```

```
#include <string.h>
const int SIZE=10;
class stack
{  public:
    stack()         //构造函数
      { tos=0; }
      void push(char ch);//将数据 ch 压入栈
      char pop();       //将栈顶数据弹出栈
      char stck[SIZE];  //数组，用于存放栈中数据 SIZE 上面赋值为 10
      int tos;          //栈顶位置（数组下标）
};
//stack::stack()    //构造函数，初始化栈
//{  tos=0;  }
void stack::push(char ch)//压入栈
{
    if(tos==SIZE)
    {
        cout<<"Stack is full";//栈是满的
        return;
    }
    stck[tos]=ch;
    tos++;
}
char stack::pop()//弹出栈
{
```

```
    if(tos==0)
    { cout<<"Stack is empty";//栈是空的
      return 0;
    }
    tos--;
    return stck[tos];
}
void main()
{  int i;
   char str[20];
   char re_str[20];
   cout<<"\nplease input a string: ";
   cin>>str;
   stack ss;
   for(i=0;i<strlen(str);i++)
   ss.push(str[i]);
   for(i=0;i<strlen(str);i++)
        re_str[i]=ss.pop();
   re_str[i]='\0';
   cout<<"\nreverse string: ";
   cout<<re_str<<endl;
}
```

附：用 C 写反序输出程序

步骤：打开 VC 系统，FileànewàFileàC++Source Fileà 改变路径 Location 为本章的路径 àFile 处写文件名 àokà 开始写 C 程序 à 之后编译运行

```
#include <stdio.h>
#include <string.h>
//#include <ctype.h>
main()
{  int inverse(char str[]);  //函数原型说明
   char str[100];
   printf("Input string: ");
   scanf("%s",str);
   inverse(str);
   printf("Inverse string: %s\n",str);
}
int inverse(char str[])   //函数定义
{   char t;
    int i,j;
    for(i=0,j=strlen(str);i<strlen(str)/2;i++,j--)
    {   t=str[i];
        str[i]=str[j-1];
          str[j-1]=t;
    }
    return 0;
}
```

第四章：类和对象（二）

[4_1]什么是对象数组

所谓对象数组是指每一个数组元素都是对象的数组，也就是说，若一个类有若干个对象，我们把这一系列的对象用一个数组来存放。对象数组的元素是对象，不仅具有数据成员，而且还有函数成员。

[4_2]什么是 this 指针？它的主要作用是什么？

C++为成员函数提供了一个名字为 this 的指针，这个指针称为自引用指针。每当创建一个对象时，系统就把 this 指针初始化为指向该对象。每当调用一个成员函数时，系统就自动把 this 指针作为一个隐含的参数传给该函数。不同的对象调用同一个成员函数时，C++编译器将根据成员函数的 this 指针所指向的对象来确定应该引用哪一个对象的数据成员。

[4_3]友元函数有什么作用？

友元函数不是当前类的成员函数，而是独立于当前类的外部函数，但它可以访问该类的所有对象的成员，包括私有成员和公有成员。通过友元函数可以在不放弃私有数据安全的情况下，使得类外部的函数能够访问类中的私有成员。

当一个函数需要访问多个类时，友元函数非常有用，普通的成员函数只能访问其所属的类，但是多个类的友元函数能够访问相应的所有类的数据。此外，在某些情况，例如运算符被重载时，需要用到友元函数。

[4_4]假设在程序中已经声明了类 point，并建立了其对象 p1 和 p2。请回答以下几个语句有什么区别？

(1)point p1,p2;　用带缺省参数的构造函数或不带参数的构造函数，定义了 point 类的 2 个对象 p1 和 p2。

(2)point p2=p1;　依据已存在的对象 p1,用赋值形式调用拷贝构造函数，创建对象 p2。

(3)point p2(p1);　依据已存在的对象 p1，用显示调用拷贝构造函数，创建对象 p2

(4)p2=p1;　对象赋值语句，将对象 p1 数据成员的值拷贝到对象 p2 中。

[4_5]在下面有关静态成员的描述中，正确的说法是（B）见书 133 页

说明：C++中规定在建立对象前，就可以为静态数据成员赋值。同时规定在静态成员函数中不能使用 this 指针。静态成员函数在类外定义时，不需要用 static 前缀。静态成员函数即可以在类内定义也可以在类外定义

[4_6]在下面有关友元函数的描述中，正确的说法是（A）（134）

说明：在 C++中友元函数是独立于当前类的外部函数，一个友元函数可以同时定义为两个类的友元函数。友元函数即可以在类的内部也可以在类的外部定义，而在外部定义友元函数时，不必加关键字 friend

[4_7]友元函数的作用之一是（A）（134）

说明：由于友元函数可以直接访问对象的私有成员，所以友元的作用是提高程序运行的效率。

[4_8]指出下面程序的错误，并说明原因：答案是将其中对应的 2 条改成：

```
cout<<Student::get_x()<<"Studet exist,y="<<stu1.get_Sno()<<"\n";
cout<<Student::get_x()<<"Studet exist,y="<<pstu->get_sno()<<"\n";
```

因为：非静态成员函数的调用方法与静态成员函数的调用方法不同。

[4_9]答：

```
#include <iostream.h>
#include <stdlib.h>
class CTest
{  public:
    const int y2;
    CTest(int i1,int i2):y1(i1),y2(i2)
    {  y1=10;  //错误，y1 是调用 const 定义的，不能修改
      x=y1;
    }
```

```
    int readme() const;
    // .....
  private:
    int x;
const int y1;
};
int CTest::readme() const
{  int i;
  i=x;
  x++;  //错误，函数定义用了 const，表示该函数不能修改对象
  return x;
}
void main()
{  CTest c(2,8);
  int i=c.y2;
  c.y2=i;  //错误，y2 是常量，不能修改
  i=c.y1;  //错误，y1 是私有变量，不能直接存取
}
```

[4_10]答：

```
#include <iostream.h>
#include <stdlib.h>
class CTest
{  public:
    CTest ()
    { x=20; }
```

```
    void use_friend();
  private:
    int x;
    friend void friend_f(CTest fri);
};
void friend_f(CTest fri)
{ fri.x=55; }
void CTest::use_friend()
{ CTest fri;
  this->friend_f(fri); //错误。友元函数不是成员函数，所以不能用 this->调用友/
/元函数
  ::friend_f(fri);
}
void main()
{ CTest fri,fri1;
  fri.friend_f(fri); //错误，友元函数不是成员函数，所以不能用对象.函数名调
//用友元函数
friend_f(fri1);
}
```

[4_11]答：

```
#include <iostream.h>
#include <stdlib.h>
class CTest
{ public:
    CTest()
```

```
    { x=20; }
    void use_this();
  private:
    int x;
};
void CTest::use_this()
{ CTest y,*pointer;
  this=&y; //错误，不能对 this 直接赋值
  *this.x=10; //错误，按优先级原句的含义是*(this.x)=10，显然不对，正确的写
//法是(*this).x=10;或 this->x=10;
pointer=this;
pointer=&y;
}
void main()
{ CTest y;
  this->x=235; //错误，this 的引用不能在外部函数中，只能在内部函数中。
}
```

[4_12]答：运行结果是：

10，20

30，48

50，68

70，80

90，16

11，120

[4_13]答：运行结果是：

Constructing

10

Destructing.

100

Destructing

[4_14]答：运行结果是：

3 objects in existence

4 objects in existence after allocation

3 objects in existence after deletion

说明：这个程序使用静态数据成员追踪记录创建对象的个数，完成这一工作的方法就是每创建一个对象就调用构造函数一次，每调用构造函数一次，静态数据成员 total 就增加 1，每撤消一个对象就调用析构函数一次。每调用析构函数一次，静态数据成员 total 就减少 1。

[4_15] 运行结果是：

Here's the program output.

Let's generate some stuff...

Counting at 0

Counting at 1

Counting at 2

Counting at 3

Counting at 4

Counting at 5

Counting at 6

Counting at 7

Counting at 8

Counting at 9

说明：在程序中 main()只包括了一个 return 语句，但竟然有内容输出！什么时候调用了构造函数？构造函数在对象被定义时调用。那么对象 anObject 是何时被调用的呢？是在 main()之前，语句"test anObject"处。因此，anObject 的构造函数是先于 main()被调用的。在 main()之前的所有全局变量都是在 main()开始之前就建立了的。应该尽可能避免使用全局变量，因为全局变量有可能引起名称冲突，使程序的执行结果和预想的不一样。

[4_16] [4_17]构建一个类 book，其中含有 2 个私有数据成员 qu 和 price，建立一个有 5 个元素的数组对象，将 qu 初始化为 1~5，将 price 初始化为 qu 的 10 倍。显示每个对象的 qu*price 答案见下：

```
#include <iostream.h>
class book
{  public:
    book(int a,int b)
      { qu=a;  price=b; }
      void show_money()
      { cout<<qu*price<<"\n"; }
private:
    int qu,price;
};
main()
{  book ob[5]={ book(1,10),book(2,20),
            book(3,30),book(4,40),book(5,50) };
  //16 题用下面语句
/*int i;
```

```
  for(i=0;i<5;i++)
   ob[i].show_money();
  return 0;*/
//17 题用下面的语句
  int i;
  book *p;
  p=&ob[4];
for(i=0;i<5;i++)
{  p->show_money();
  p--;
}
return 0;
}
```

[4_18]使用 C++的（见书 139 页题）

```
#include <iostream.h>
//#include <conio.h>
class toy
{  public:
  toy(){ }
  toy(int p,int c)
  {  price=p;
    count=c;
  }
  void input(int p,int c);
  void compute();
```

```
    void print();
private:
    int price;
    int count;
    long total;
};
void toy::input(int p,int c)
{  price=p;
   count=c;
}
void toy::compute()
{  total=(long)price*count;  }
void toy::print()
{ cout<<"price="<<price<<"  count="<<count<<"  total="<<total<<"\n";  }
void main()
{ toy te(2,100);//测试构造函数
  toy *ob;
  ob=new toy[6];
  ob[0].input(25,130);
  ob[0].input(25,130);
  ob[1].input(30,35);
  ob[2].input(15,20);
  ob[3].input(25,120);
  ob[4].input(45,10);
  ob[5].input(85,65);
```

```
  for(int i=0;i<6;i++)
        ob[i].compute();
// clrscr();
  for(i=0;i<6;i++)
        ob[i].print();
  delete ob;
}
```

[4_19]构建一个类 stock 见书 139 页 答案如下：

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
const int SIZE=80;
class stock
{  public:
    stock()
      {  strcpy(stockcode,"");  }
      stock(char code[],int q=1000,float p=8.98)
      {  strcpy(stockcode,code);
        quan=q;
        price=p;
      }
      void print(void)
      {  cout<<this->stockcode;
        cout<<" "<<this->quan<<" "<<this->price<<endl;
      }
```

```
private:
    char stockcode[SIZE];
    int quan;
    float price;
};
main()
{
    stock st1("600001",3000,8.89);
    st1.print();
    stock st2;
    char stockc[]="600002";
    st2=stockc;
    st2.print();
    return 0;
}
```

说明：执行以下语句，可在定义对象的同时，给数据成员设置初值。但构造函数的参数必须定义为缺省值：

```
stock st2;
char stockc[]="600002";
st2=stockc;
st2.print();
```

定义不含第 2 和第 3 个参数的对象 st2 时，quan 的值为 1000，price 的值为 8.98

[4_20]编写一个有关股票的程序，见书 139 页题

```
#include <iostream.h>
```

```
#include <string.h>
#include <iomanip.h>
class shen_stock;   //深圳类
class shang_stock   //上海类
{ public:
    shang_stock(int g,int s,int p);  //构造函数
      friend void shang_count(const shang_stock ss);//计算上海的
                                        //股票总数
      friend void count(const shang_stock ss,const shen_stock zs);
                    //计算上海和深圳的股票总数
private:
      int general; //普通股票个数
      int st;      //ST 股票个数
      int pt;      //PT 股票个数
};
shang_stock::shang_stock(int g,int s,int p)//构造函数
{  general=g;
   st=s;
   pt=p;
}
class shen_stock
{  int general;  //普通股票个数
   int st;       //ST 股票个数
   int pt;       //PT 股票个数
public:
```

```
    shen_stock(int g,int s,int p);//构造函数
    friend void shen_count(const shen_stock es);//计算深圳的股票总数
    friend void count(const shang_stock ss,const shen_stock zs);
                    //计算上海和深圳的股票总数
};
shen_stock::shen_stock(int g,int s,int p)//构造函数
{  general=g;
   st=s;
   pt=p;
}
main()
{  shang_stock shanghai(1600,20,10);//建立对象
    //表示上海有 1600 支股票，20 支 ST 股票，10 支 PT 股票
   shen_stock shenzhen(1500,15,8); //建立对象
    //表示深圳有 1500 支股票，15 支 ST 股票，8 支 PT 股票
   shang_count(shanghai);//计算上海的股票总数
   shen_count(shenzhen);//计算深圳的股票总数
   count(shanghai,shenzhen);//计算上海和深圳的股票总数
   return 0;
}
void shang_count(const shang_stock ss)//计算上海的股票总数
{ int s;
  cout<<"stocks of shanghai are "<<ss.general+ss.st+ss.pt<<endl;
}
void shen_count(const shen_stock es)//计算深圳的股票总数
```

```
{ cout<<"stocks of shanghai and shenzhen "<<es.general+es.pt<<endl; }
void count(const shang_stock ss,const shen_stock es)
                    //计算上海和深圳的股票总数
{ int s;
  s=es.general+es.st+es.pt+ss.general+ss.st+ss.pt;
  cout<<"stocks of shanghai and shenzhen "<<s<<endl;
}
```

第五章：继承与派生类

[5_1]答：类的继承方式有 public(公有继承)、protected(保护继承)和 private(私有继承)3 种，不同的继承方式导致原来具有不同访问属性的基类成员在派生类中的访问属性也有所不同。

（1）基类中的私有成员

无论哪种继承方式，基类中的私有成员不允许派生类继承，即在派生类中是不可直接访问的。

（2）基类中的公有成员

当类的继承方式为公有继承时，基类中的所有公有成员在派生类中仍以公有成员的身份出现；当类的继承方式为私有继承时，基类中的所有公有成员在派生类中都以私有成员的身份出现；当类的继承方式为保护继承时，基类中的所有公有成员在派生类中都是以保护成员的身份出现。

（3）基类中的保护成员

当类的继承方式为公有继承时，基类中的所有保护成员在派生类中仍以保护成员的身份出现；当类的继承方式为私有继承时，基类中的所有保护成员在派生类中都是以私有成员的身份出现；当类的继承方式为保护继承时，基类中的所有保护成员在派生类中仍以保护成员的身份出现。

[5_2]答：派生类不能直接访问基类的私有成员，但是可以通过基类提供的公有

成员函数间接地访问基类的私有成员。

[5_3]答：保护成员可以被派生类的成员函数访问，但是对于外界是隐藏起来的，外部函数不能访问它。因此，为了便于派生类的访问，可以将基类私有成员中需要提供给派生类访问的成员定义为保护成员。C++规定，派生类对于保护成员的继承与公有成员的继承很相似，也分为两种情况：若为公有派生，则基类中的保护成员在派生类中也为保护成员；若为私有派生，则基类中的保护成员在派生类中成为私有成员。

[5_4]答：通常情况下，当创建派生类对象时，首先执行基类的构造函数，随后再执行派生类的构造函数；当撤消派生类对象时，则先执行派生类的析构函数函数，随后再执行基类的析构函数。

[5_5]答：当基类的构造函数没有参数或没有显示定义构造函数时，派生类可以不向基类传递参数，甚至可以不定义构造函数。当基类含有带参数的构造函数时，派生类必须定义构造函数，以提供把参数传递给基类构造函数的途径。

派生类构造函数的一般格式如下：

派生类构造函数名（参数表）：基类构造函数名（参数表）

{　　}

其中基类构造函数的参数，通常来源于派生类构造函数的参数表，也可以用常数值。由于析构函数是不带参数的，在派生类中是否定义析构函数与它所属的基类无关，基类的析构函数不会因为派生类没有析构函数而得不到执行，它们各自是独立的。

[5_6]答：当一个派生类具有多个基类时，这种派生方法称为多继承。多继承构造函数的执行顺序与单继承构造函数的执行顺序相同，也是遵循先执行基类的构造函数，再执行对象成员的构造函数，最后执行派生类构造函数的原则。在多个基类之间则严格按照派生类声明时从左到右的顺序来排列先后。而析构函数的执行顺序刚好与构造函数的执行顺序相反。

[5_7]在类的派生中为什么引入虚基类？虚基类构造函数的调用顺序是如何规定的？

当引用派生类的成员时，首先在派生类自身的作用域中寻找这个成员，如果没有找到，则到它的基类中寻找。如果一个派生类是从多个基类派生来的，而这些基类又有一个共同的基类，则在这个派生类中访问这个共同的基类中的成员时，可能会产生二义性。为了解决这种二义性，C++引入了虚基类的概念。

虚基类构造函数的调用顺序是：

（1）若同一层次中包含多个虚基类，这些虚基类的构造函数按它们的说明的先后次序调用。

（2）若虚基类由非虚基类派生而来，则仍然先调用基类构造函数，再调用派生类的构造函数。

（3）若同一层次中同时包含虚基类和非虚基类，应先调用虚基类的构造函数，再调用非虚基类的构造函数，最后调用派生类构造函数。

[5_8]答：A

[5_9]答：C　说明：C++中继承方式有 3 种：私有继承、保护继承和公有继承。

[5_10]答：C

[5_11]答：C　说明：set_price()和 print_price()在派生类中是私有成员函数，不能被派生类对象直接访问，所以 A 和 D 是错误的。code 是私有数据成员，不能直接存取，所以 B 也是错误的。

[5_12]答：B 说明：code 是私有数据成员，不能直接存取，所以 B 是错误的。

[5_13]答：运行结果是：

```
Person Name=Angel
Student Name=Angel
Employee Name=Angel
SideLine Name=Angel
```

说明：（1）从结果看出，首含不的虚基类 Person 的构造函数只执行了一次。当类 SideLine 的构造函数调用了虚基类 Person 的构造函数之后，类 Student 和类 Employee 对类 SideLine 的构造函数的调用被忽略了，这是初始化虚基类和非虚基类不同的地方。（2）多重继承的构造函数执行顺序遵循先执行基类的构造函数，再执行对象成员的构造函数，最后执行派生类构造函数的原则。

[5_14]运行结果是：40 50 10 20 30 40 50 60

[5_15]修改后的程序如下：

```
#include <iostream.h>
#include <iomanip.h>
class table
{  protected:
     int i,j;
public:
       table(int p,int q)
       {  i=p;  j=q;  }
       void ascii(void);
};
void table::ascii(void)
{  int k=1;
   for(;i<j;i++)
   {  cout<<setw(4)<<i<<" "<<(char)i;
      if((k)%12==0)
                cout<<"\n";
        k++;
   }
```

```
    cout<<"\n";
}
class der_table:public table
{  protected:
     char *c;
public:
       der_table(int p,int q,char *m):table(p,q)
       {  c=m;  }
       void print(void);
};
void der_table::print(void)
{  cout<<c<<"\n";
   table::ascii();
}
void main()
{  der_table ob('a','z',"ASCII value---char");
   ob.print();
}
```

[5_16]程序编制如下：

```
#include <iostream.h>
#include <string.h>
class time
{  public:
     time(int h,int m,int s)
        {  hours=h;
```

```
            minutes=m;
            seconds=s;
        }
        virtual void display()
        {  cout<<hours<<":"<<minutes<<":"<<seconds<<endl; }
protected:
        int hours,minutes,seconds;
};
class date
{  public:
    date(int m,int d,int y)
        {  month=m;
           day=d;
           year=y;
        }
        virtual void display()
        {  cout<<month<<"/"<<day<<"/"<<year;  }
protected:
        int month,day,year;
};
class birthtime:public time,public date//加此类
{  public:
     birthtime(char *cn,int mm,int dd,int yy,int hh,int mint,int ss)
                :time(hh,mint,ss),date(mm,dd,yy)
        {  strcpy(childname,cn);  }
```

```
        void display()
        {  cout<<childname<<' ';
           date::display();
        cout<<endl;
              time::display();
        }
protected:
       char childname[20];
};
void main()
{  birthtime yx("yuanxiang ",10,27,1966,13,20,0);
   yx.display();
}
```

[5_17]程序如下

```
#include <iostream.h>
#include <string.h>
class animal
{  public:
     animal()
        {  name=NULL;  }
        animal(char *n);
        ~animal()
        {  delete name;  }
        virtual void whoami();
protected:
```

```
        char *name;
};
class cat:public animal
{  public:
     cat():animal()
        {     }
        cat(char *n):animal(n)
        {     }
        void whoami();
};
class tiger:public cat
{  public:
     tiger():cat()
        {     }
        tiger(char *n):cat(n)
        {     }
        void whoami();
};
animal::animal(char *n)
{  name=new char [strlen(n)+1];
   strcpy(name,n);
}
void animal::whoami()
{  cout<<"generic animal";  }
void cat::whoami()
```

```
{  cout<<"i am a cat name "<<name<<endl; }
void tiger::whoami()
{  cout<<"i am atiger name "<<name<<endl; }
void main()  //添加的主函数
{  cat cat("john");
   cat.whoami();
   tiger tiger("richard");
   tiger.whoami();
}
```

[5_18]实现本题功能的程序如下：

```
#include <iostream.h>
class building
{  public:
   building(int f,int r,double ft)
   {  floors=f;
      rooms=r;
         footage=ft;
   }
protected:
        int floors;//层数
        int rooms;  //房间数
        double footage;//平方数
};
class house:public building
{  public:
```

```
    house(int f,int r,double ft,int br,int bth)
nbsp;    :building(f,r,ft)
  {  bedrooms=br;  bathrooms=bth;  }
  void show()
  {  cout<<"floors:  "<<floors<<'\n';
    cout<<"rooms:  "<<rooms <<'\n';
    cout<<"square footage:  "<<footage<<'\n';
      cout<<"bedrooms:  "<<bedrooms<<'\n';
      cout<<"bathroorns:  "<<bathrooms<<'\n';
  }
private:
    int bedrooms;//卧室数
    int bathrooms;//浴室数
};
class office:public building
{  public:
   office(int f,int r,double ft,int p,int ext)
        :building(f,r,ft)
     {  phones=p;  extinguishers=ext;  }
     void show()
     {  cout<<"floors:  "<<floors<<'\n';
       cout<<"rooms:  "<<rooms<<'\n';
        cout<<"square footage:  "<<footage<<'\n';
        cout<<"telephones:  "<<phones<<'\n';
        cout<<"fire extinguishers:  ";
        cout<<extinguishers<<'\n';
     }
```

```
private:
    int phones;//电话数
    int extinguishers;//灭火器数
};
main()
{  house h_ob(2,12,5000,6,4);
   office o_ob(4,25,12000,30,8);
   cout<<"housee: \n";
   h_ob.show();
   cout<<"\noffice: \n";
   o_ob.show();
   return 0;
}
```

[5_19]实现本题功能的程序如下：

```
#include <iostream.h>
#include <iomanip.h>
const int l=80;
class person
{  public:
    void input()
      {  cout<<"\n input name: ";
         cin>>name;
          cout<<"\n certificate no: ";
          cin>>id;
      }
      void print()
      {  cout<<"\n certificate no: "<<id;  }
      void printname()
```

```
    { cout<<setw(8)<<id<<
private:
    char name[1],id[1];
};
class stud
{ public:
  void input()
    { cout<<" input address: ";
      cin>>addr;
      cout<<" input telphone no: ";
      cin>>tel;
    }
    void print()
    { cout<<"\n address: "<<"\t"<<addr;
      cout<<"\n telphone no: "<<"\t"<<tel;
    }
private:
    char addr[1];
    char tel[1];
};
class student:private person
{ public:
  void input()
    { person::input();
      cout<<" input years old: ";
      cin>>old;
      cout<<" input score no: ";
      cin>>sno;
```

```
    }
    void print()
    { person::print();
      cout<<"\n tears old: "<<"\t"<<old;
       cout<<"\n score no: "<<"\t"<<sno;
    }
    void printname()
    { person::printname(); }
private:
    int old;
    unsigned long sno;
};
class score:private student,private stud
{ public:
   void input()
    { student::input();
      stud::input();
       cout<<"input math score: ";
       cin>>math;
       cout<<"input english score: ";
       cin>>eng;
    }
    void print()
    { student::print();
      stud::print();
       cout<<"\n math score: "<<"\t"<<math;
       cout<<"\n english svore: "<<"\t"<<eng;
       cout<<"\n average score: "<<"\t"<<float(math+eng)/2;
```

```
    }
    void printname()
    { student::printname(); }
private:
    int math;
    int eng;
};
class teacher:public person
{ public:
    void input()
    { person::input();
      cout<<" input degree: ";
      cin>>degree;
      cout<<" input department: ";
      cin>>dep;
    }
    void print()
    { person::print();
      cout<<"\n degree: "<<"\t"<<degree;
      cout<<"\n department: "<<"\t"<<dep;
    }
    void printname()
    { person::printname(); }
private:
    char degree[1],dep[1];
};
void main()
{ score c1;
```

```
    teacher t1;
    cout<<"input data for score 1: ";
    c1.input();
    cout<<"input data for teacher 1: ";
    t1.input();
cout<<"\n data on student ";
    c1.printname();
    c1.print();
    cout<<"\n data on teacher ";
    t1.printname();
    t1.print();
}
```

[5_20] [5_21] （略

第六章：多态性与虚函数

[6_1]答：所谓联编就是把函数名与函数体的程序代码连接（联系）在一起的过程。静态联编就是在编译阶段完成的联编。编译时的多态是通过静态联编来实现的。静态联编时，系统用实参与形参进行匹配，对于同名的重载函数便根据参数上的差异进行区分，然后进行联编，从而实现了多态性。

[6_2]答：动态联编是运行阶段完成的联编。运行时的多态是用动态联编实现的。即当程序调用到某一函数名时，才去寻找和连接其程序代码，对面向对象程序设计而言，就是当对象接收到某一消息时，才去寻找和连接相应的方法。

[6_3]答：编译时多态性主要是通过函数重载和运算符重载实现的。运行时多态性主要是通过虚函数来实现的。

[6_4]什么是虚函数？虚函数与函数重载有哪些相同与不同点？

虚函数就是在基类中被关键字 virtual 声明，并在派生类中重定义的函数。虚函数同派生类的结合可使 C++支持运行时的多态性。

在一个派生类中重新定义基类的虚函数是函数重载的另一种形式，但它不同于一般的函数重载。普通的函数重载时，其函数的参数或参数类型必须有所不同，函数的返回类型也可以不同。但是，当重载一个虚函数时，贝就是说在派生类中重新定义虚函数时，要求函数名、返回类型、参数个数、参数的类型和顺序与基类中的虚函数原型完全相同。如果仅仅返回类型不同，其余均相同，系统会给出错误信息；若仅仅函数名相同，而参数的个数、类型或顺序不同，系统将它作为普通的函数重载，这时将丢失函数的特性。

[6_5]答：纯虚函数是一个在基类中说明的虚函数，它在该基类中没有定义，但要求在它的派生类中必须定义自己的版本，或重新说明为纯虚函数。

纯虚函数的定义形式如下：

virtual 函数类型 函数名（参数表）=0;

如果一个类至少有一个纯虚函数，那么就称该类为抽象类。

[6_6]答：A 说明：C++规定构造函数不能是虚函数。

[6_7] 答：D 说明：C++规定派生类中的虚函数的返回类型、参数个数、参数类型必须与虚函数的原型相同。

[6_8]答：C 说明：virtual show()=0;表示 show 是纯虚函数，但没指定是否带返回值，错误。Virtual show();未表明 show 是纯虚函数，错误。void show()=0 virtual;把 virtual 的位置写错了。正确答案是 virtual void show()=0;

[6_9]题：不正确

因为虚函数在派生类中重新定义时，其函数原型，包括返回类型、函数名、参数个数与参数类型的顺序，都必须与基类中的原型完全相同，而本程序段中虚函数在派生类中重定义时，参数个数与基类中的原型不相同。

[6_10]题：本程序的运行结果为：

Stock class

Stock class

Stock class 说明如下：

（1）指向 Stock 类的对象指针，也可以指向派生类对象。例如本程序中，指向 Stock 类型的指针 ptr 可以接受派生类对象 f1 的起始地址。

（2）语句 ptr->print()均调用 Der1_Stock::print()或 Der2_Stock::print(),这是因为基类和派生类定义的 print()函数的返回值类型、参数类型和个数均相同，基类中定义的 print()未声明成虚类型(virtual)，这时的函数调用是在编译时静态联编的。

[6_11]题：将上例基类 Stock 中的 print()函数前面加 virtual 就行了。

[6_12]题：程序实现如下：

```
#include <iostream.h>
class car
{  public:
    double distance;
     car(double f)
     {  distance = f;  }
     virtual void travel_time()
     {  cout<<"car:travel time at 120 kph: ";
        cout<<distance/120<<"\n";
     }
};
class truck:public car
{  public:
   truck(double f):car(f)
   {  }
   void travel_time()
   {  cout<<"truck: travel time at 80 kpt: ";
      cout<<distance/80<<"\n";
   }
};
main()
{  car *p,car_time(150.0);
```

```
    truck truck_time(150);
    p=&car_time;
    p->travel_time();
    p=&truck_time;
    p->travel_time();
    return 0;
}
```

[6_13]题：程序如下：

```
#include <iostream.h>
class currency
{  public:
     void set_j(float j)
       {  japan=j;  }
       void set_e(float e)
       {  europe=e;  }
       virtual void print_exchange()=0;
protected:
      float japan,europe;
};
class ja:public currency
{  public:
     void print_exchange()
       {  cout<<"10000 ja exchange $: "<<10000/japan;
          cout<<endl;
       }
};
class eur:public currency
{  public:
```

```
    void print_exchange()
      { cout<<"100 eur exchange $: "<<100*europe;
        cout<<endl;
      }
};
main()
{ currency *ptr[2];
  ja j1;
  eur e1;
  ptr[0]=&j1;
  ptr[0]->set_j(116.30);
  ptr[0]->print_exchange();
  ptr[1]=&e1;
  ptr[1]->set_e(1.1634);
  ptr[1]->print_exchange();
  return 0;
}
```

[6_14]题：程序如下：

```
#include <iostream.h>
class currency
{ public:
    void set_j(float j)
      { japan=j; }
      void set_e(float e)
      { europe=e; }
      virtual void print_exchange(float a)=0;//加浮点参数 a
   protected:
      float japan,europe;
```

```
};
class ja:public currency
{  public:
    void print_exchange(float a)  //加浮点参数 a
     {  cout<<"10000 ja exchange $: "<<a/japan;  //变 a/japan
        cout<<endl;
     }
};
class eur:public currency
{  public:
    void print_exchange(float a)//
     {  cout<<"100  eur exchange $: "<<a*europe;  //变 a*europr
        cout<<endl;
     }
};
main()
{  currency *ptr[2];
   ja j1;
   eur e1;
   ptr[0]=&j1;
   ptr[0]->set_j(116.30);
   ptr[0]->print_exchange(100000);//放任意数
   ptr[1]=&e1;
   ptr[1]->set_e(1.1634);
   ptr[1]->print_exchange(1000);//放任意数
   return 0;
```

第七章：运算符重载

习题：[7_1]简述运算符重载的原则。

（1）运算符重载是针对新类型数据的实际需要，对原有运算符进行适当的改造完成的。一般来讲，重载的功能应当与原有的功能相类似。

（2）C++语言中只能重载原先已有定义的运算符。

（3）不是所有的运算符都能重载，它们是：类属关系运算符“.”、成员指针运算符“*”、作用域分辨符“：：”、sizeof 运算符和三目运算符“？：”如(x>y)?x:y

（4）运算符重载不能改变运算符的操作个数

（5）运算符重载不能改变运算符原有的优先级

（6）运算符重载不能改变运算符原有的结合性

（7）运算符重载不能改变运算符对预定义类型数据的操作方式

[7_2]友元运算符函数和成员运算符函数有什么不同？

（1）运算符函数可以定义为它将要操作的类的成员，称为成员运算符函数；定义为类的友元函数，称为友元运算符函数。对双目而言，成员运算符函数带有一个参数，而友元运算符函数带有 2 个参数；对单目运算符而言，成员运算符函数不带参数，而友元运算符函数带一个参数。

（2）双目运算符一般可以被重载为友元运算符函数或成员运算符函数，但当双目运算符的左操作数是一个基本数据类型，而右操作数是一个类类型时必须使用友元运算符函数。

（3）成员运算符函数和友元运算符函数可以用习惯方式调用，也可以用它们专用的方式调用

[7_3]答：D 说明：C++语言允许在重载运算符时改变运算符的原来的功能。例如将++符号重载时，可以定义为--的功能。但是，不提倡这样做。重载的功能应当与原有的功能类似。

[7_4]答：A 说明：C++规定不能用友元函数重载“=”、“[]”和“->”

[7_5]答：运行结果为：7 6 说明：重载++运算符时使用的友元函数无法引用 this 指针，函数通过传值的方法传递参数，函数体内对数据成员的改动无法传到函数体外。所以，（++I）以后，I.xi 无任何改动。

[7_6]答：结果为 5 7 说明：本题是重载运算符（），用（）模拟数组运算符[]。

[7_7]答：++dl 的调用是[illegible]

[7_8]编一程序，用成员函数重载运算符"+"和"-"将 2 个二维数组相加和相减，要求第一个二维数组的值由构造函数设置，另一个二维数组的值由键盘输入。

```
#include <iostream.h>
#include <iomanip.h>
const int row=2;  const int col=3;
class array
{  public:
    array()
      {  int i,j,k=1;
      for(i=0;i<row;i++)
          for(j=0;j<col;j++)
          {  var[i][j]=k;
            k=k+1;
          }
      }
      void get_array()//由键盘输入数组的值
      {  int i,j;
        cout<<" please input 2*3 dimension data "<<endl;
        for(i=0;i<row;i++)
          for(j=0;j<col;j++)
                cin>>var[i][j];
      }
      void display()//显示数组的值
      {  int i,j;
        for(i=0;i<row;i++)
          {  for(j=0;j<col;j++)
              cout<<setw(5)<<var[i][j];
```

```
            cout<<endl;
        }
    }
    array operator+(array X)//将 2 个数组相加
    { array temp;
      int i,j;
      for(i=0;i<row;i++)
          for(j=0;j<col;j++)
              temp.var[i][j]=var[i][j]+X.var[i][j];
          return temp;
    }
    array operator-(array X)//将 2 个数组相减
    { array temp;
      int i,j;
      for(i=0;i<row;i++)
          for(j=0;j<col;j++)
              temp.var[i][j]=var[i][j]-X.var[i][j];
          return temp;
    }
private:
    int var[row][col];
};
void main()
{ array X,Y,Z;
  Y.get_array();
  cout<<"Display object X"<<endl;
  X.display();
  cout<<"Display object Y"<<endl;
```

```
  Y.display();
  Z=X+Y;
  cout<<" Display object Z=X+Y"<<endl;
  Z.display();
  Z=X-Y;
  cout<<" Display object Z=X-Y"<<endl;
  Z.display();
}
```

[7_9]修改上题，用友元函数重载运算符"+"和"-"将 2 个二维数组相加和相减

```
#include <iostream.h>
#include <iomanip.h>
const int row=2; const int col=3;
class array
{  public:
    array()
      {  int i,j,k=1;
      for(i=0;i<row;i++)
          for(j=0;j<col;j++)
          {  var[i][j]=k;
            k=k+1;
          }
      }
      void get_array()//由键盘输入数组的值
      {  int i,j;
        cout<<" please input 2*3 dimension data "<<endl;
        for(i=0;i<row;i++)
            for(j=0;j<col;j++)
                cin>>var[i][j];
```

```
    }
    void display()//显示数组的值
    {  int i,j;
       for(i=0;i<row;i++)
       {  for(j=0;j<col;j++)
          cout<<setw(5)<<var[i][j];
          cout<<endl;
       }
    }
    friend array operator+(array X,array Y);//将 2 个数组相加
  friend array operator-(array X,array Y);//将 2 个数组相减
private:
    int var[row][col];
};
array operator+(array X,array Y)//将 2 个数组相加
{  array temp;
   int i,j;
   for(i=0;i<row;i++)
   for(j=0;j<col;j++)
   temp.var[i][j]=Y.var[i][j]+X.var[i][j];
   return temp;
}
array operator-(array X,array Y)//将 2 个数组相减
{  array temp;
   int i,j;
   for(i=0;i<row;i++)
     for(j=0;j<col;j++)
       temp.var[i][j]=Y.var[i][j]-X.var[i][j];
```

```
    return temp;
}
void main()
{   array X,Y,Z;
    Y.get_array();
    cout<<"Display object X"<<endl;
    X.display();
    cout<<"Display object Y"<<endl;
    Y.display();
    Z=X+Y;
    cout<<" Display object Z=X+Y"<<endl;
    Z.display();
    Z=X-Y;
    cout<<" Display object Z=X-Y"<<endl;
    Z.display();
}
```

[7_10]为 Date 类重载“+”运算符，实现在某一日期上（月日年）加一个天数

```
#include <iostream.h>
class Date
{   public:
      Date()
        {    }
      Date(int m,int d,int y)
      {   month=m;
          day=d;
          year=y;
      }
      void print()
```

```
    {  cout<<year<<".";
    Date operator+(int);
private:
    int month,day,year;
};
static int days[2][12]={{31,28,31,30,31,30,31,31,30,31,30,31},\
{31,29,31,30,31,30,31,31,30,31,30,31}};
int isleap(int year)
{  if((year%4==0&&year%100!=0)||(year%400==0))
       return 1;
  else
       return 0;
}
Date Date::operator +(int n)
{  int leap,y;
  leap=0;
  leap=isleap(this->year);
  n+=this->day;
  while(n>days[leap][this->month-1])
  {  n-=days[leap][this->month-1];
    if(++(this->month)==13)
       {  this->month=1;
         (this->year)++;
          leap=isleap(this->year);
       }
  }
  this->day=n;
  return *this;
```

```
}
void main()
{  Date d1(2,15,2000),d2;
   d1.print();
   d2=d1+95;
   d2.print();
}
```

第八章：模板

[8_1]为什么使用模板？函数模板声明的一般形式是什么？

模板是实现代码重用机制的一种工具，它可以实现类型参数化，即把类型定义为参数，从而实现了真正的代码重用。使用模板可以大幅度地提高程序设计的效率。

函数模板的一般说明形式如下：

```
template <class 类型参数>
返回类型 函数名(模板形参表)
{
   函数体
}
```

其中，template 是一个声明模板的关键字，它表示声明一个模板。类型参数前需要加关键字 class(或 typename)。

[8_2]什么是模板实参和模板函数？

将函数模板中实例化的参数称为模板实参，用模板实参实例化的函数称为模板函数。

[8_3]为什么使用类模板，类模板和模板类之间的关系是什么？

类模板允许用户为类定义一种模式，使得类中的某些数据成员、某些成员函数的参数或返回值能取任意数据类型。

类模板不是代表一个具体的、实际的类，而是代表着一类类。模板类是类模板对某一特定类型的实例，表示某一具体的类。

[8_4]函数模板与同名的非模板函数重载时，调用的顺序是怎样的？

（1）寻找一个参数完全匹配的函数，如果找到了就调用它。

（2）寻找一个函数模板，将其实例化，产生一个匹配的模板函数，若找到了，就调用它。

（3）若（1）和（2）都失败了，再试一试低一级的对函数的重载方法，例如通过类型转换可产生参数匹配等，若找到了，就调用它。

若（1）（2）（3）均未找到匹配的函数，则是一个错误的调用。如果在第（1）步有多于一个的选择，那么这个调用是意义不明确的，是一个错误调用。

[8_5] D　说明：实例化的模板实参之间必须保持一致的类型

[8_6] A

[8_7] 运行结果：0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,i out of range

说明：Holder 类的大小是 20，所以输出下标为 21 的元素会输出 i out of range

[8_8] 运行结果为：10　0.25

X This is a test

说明：程序声明了两种类型的对象，ob1 的两个参数分别为整形与双精度型，ob2 的两个参数分别为字符型与字符串型。

[8_9] 解：主函数中语句"Stack aSt(5);"是错误的，因为首先要将模板实例化，才能由模板生成对象，应该将此语句改为"Stack<int>aSt(5);"

[8_10]解：

```
#include <iostream.h>
template <class Type>
Type min(Type a,Type b,Type c)
{  a=(a<b?a:b);
   return (a<c?a:c);
}
```

[8_11]解：执行语句：

```
cout<<max(f,i)<<endl;    cout<<max(i,f)<<endl;
```

时出现错误。出错的原因是：当函数中的参数类型与调用该函数真正被调用时才能决定。在调用时，编译器将按最先遇到的实参的类型隐含地生成一个模板函数，用它对所有模板进行一致性检查。例如对语句：max(f,i);

编译器将先按变量 f 将 Type 解释为 double 类型，此后出现的模板实参 i 不能解释为 double 类型时，便发生错误，在此没有隐含的类型转换的功能。

如果想让运行结果为：10.56　10.56　主程序应修改为：

```
void main()
{  int i=10;
double f=10.56;
cout<<endl;
cout<<max(f,double(i))<<endl;
cout<<max(double(i),f)<<endl;
}
```

[8_12]程序实现如下：

```
#include <iostream.h>
template <class Type>
Type max(Type *Array,int size)
{  int i,j=0;
   for(i=1;i<size-1;i++)
        if(Array[i]>Array[j])
        {  j=i;  }
        return Array[j];
}
void main()
{  int intArray[]={11,12,13,14,7,8,9};
   double dArray[]={11.2,12.3,13.2,14.5,14.8,8.7,9.3};
   cout<<max(intArray,7)<<endl;
   cout<<max(dArray,7)<<endl;
```

```
}
```

[8_13]程序实现如下：

```
#include <iostream.h>
template <class Type>
void sort(Type *Array,int size)
{  int i,j;
   for(i=0;i<size-1;i++)
        for(j=0;j<size-i-1;j++)
            if(Array[j]>Array[j+1])
            {  Type temp=Array[j];
              Array[j]=Array[j+1];
               Array[j+1]=temp;
            }
}
void main()
{  int intArray[]={11,12,13,14,7,8,9};
   int i;
   sort(intArray,7);
   cout<<endl;
   for(i=0;i<7;i++)
       cout<<intArray[i]<<endl;
}
```

[8_14]程序实现如下：

```
#include <iostream.h>
template <class X>
class input
{  public:
    input(char *s,X min,X max);
```

```
private:
   X data;
};
template <class X>
input<X>::input(char *s,X min,X max)
{
     do
     { cout<<s<<": ";
       cin>>data;
     }
     while(data<min||data>max);
}
main()
{  input<int> i("enter int",0,10);
   input<char> c("enter char",'A','Z');
   cout<<"---end---"<<endl;
   return 0;
}
```

[8_15]程序实现如下：

```
#include <iostream.h>
template <class data_t>
class list
{  public:
    list(data_t d);
      void add(list *node)
      {  node->next=this;
         next=0;
      }
```

```
    list *getnext()
    { return next; }
    data_t getdata()
    { return data; }
private:
    data_t data;
    list *next;
};
template <class data_t>
list<data_t>::list(data_t d)
{ data=d;
  next=0;
}
main()
{ list<char> start('a');
  list<char> *p,*last;
  int i;
  //建立链表
  last=&start;
  for(i=1;i<26;i++)
  { p=new list<char>('a'+i);
    p->add(last);
      last=p;
  }
  //显示链表
  p=&start;
  while(p)
  { cout<<p->getdata();
```

```
    p=p->getnext();
  }
  cout<<endl;
  return 0;
}
```

说明：类模板的声明类似于函数模板的声明。list 类对象被定义的时候才确定模板参数的实际类型。特别注意：list <char> start('a');声明。注意所要求的数据类型是如何通过放在角括号内传递的。该程序建立了包含字母表字符的链表，并显示它们。但是，在建立对象的时候，只通过改变指定的数据类型，就可以改变表所保存的数据类型。例如，可以用声明：list<int> int_start(1);建立保存整数的其它对象。

[8_16]程序实现如下：

```
#include <iostream.h>
#include <process.h>
template <class Type>
class node
{  public:
    Type data;
     node<Type> *next;
};
template<class Type>
class Stacklink
{  public:
    Stacklink();//构造函数
     ~Stacklink();//析构函数
     void push(Type value);//进栈
     Type pop();//出栈，并返回栈顶元素
     int isnull();//判栈空
```

```
private:
    node<Type> *top;
};
template<class Type>
Stacklink<Type>::Stacklink()
{  node<Type> *T;
   T=new node<Type>;//申请一个空间
   T->next=NULL;//指针项清 0
   top=T;//栈顶指针指向新申请的空间
   //本例使用的是带头结点的单链表，第一个结点不会存储栈元素
}
template<class Type>
Stacklink<Type>::~Stacklink()//析构函数，删除单链表中的所有结点
{  node<Type> *N,*Temp;
   for(N=top;N!=NULL;)//从表头开始删除
   {  Temp=N;
      N=N->next;
      delete Temp;
   }
}</D< p>
, , V>
template<class Type>
void Stacklink<Type>::push(Type value)//进栈操作
{  node<Type> *T;
   T=new node<Type>;//申请一个空间
   if(T!=NULL)//判断申请是否成功
   {  T->next=top;//新申请的空间的指针域指向原栈顶元素
```

```
    T->data=value;//新申请结点数据赋值
      top=T;//栈指针指向新申请的空间
  }
  else
  {  cout<<"assigned failure!";
    exit(1);
  }
}
template<class Type>
int Stacklink<Type>::isnull()//判栈空
{  if(top->next==NULL)
  return 1;//如果链表中只剩一个结点（表头结点），栈为空返回 1
  else
      return 0;
}
template<class Type>
Type Stacklink<Type>::pop()//退栈
{  Type value;
  node<Type> *Temp;
  if(top->next==NULL)//判栈空
  {  cout<<"\n the stack is Null!\n";
    return(0);
  }
  value=top->data;//取栈顶元素
  Temp=top;//Temp 指向栈顶元素
```

```
    top=top->next;//Top 指向下一个结点
    delete Temp;//删除 Temp 指向的结点
    return(value);//返回原栈顶元素
}
void main()
{  int l=1,octal=0;
   long decimal,j;
   cout<<"\nplease input integer: ";
   cin>>decimal;//输入一个十进制数
   j=decimal;
   Stacklink<int> ss;//定义一个整数栈
   while(j!=0)
   {  ss.push(j%8*l);//将八进制的每一位数进栈
      j=j/8;
         l=l*10;
   }
   cout<<endl;
   while(!ss.isnull())//栈不空，循环
   {  j=ss.pop();//取栈顶元素
      octal=j+octal;//将栈顶元素累加
   }
   cout<<octal<<endl;
}
```

说明：本例使用的是带头结点的单链表，第一个结点不会存储栈元素。也就是说，空栈时，链表有一个结点，栈指针 top 指向该结点。

Stacklink 类中有构造函数、进栈、出栈（并返回栈顶元素）和判栈空成员函数。在构造函数中需要申请一个空间，然后将该空间的指针域清 0，最后用栈顶指针指向新申请的空间。析构函数则负责删除单链表中的所有结点。进栈操作的步骤是申请一个空间将新申请的空间的指针域指向原栈顶元素，并将新申请空间的数据域用需要进栈的数据赋值，最后栈指针指向新申请的空间。判栈空是判断链表中是否只剩一个结点（表头结点）。若是，说明栈为空返回 1；否则返回 0。退栈（并返回栈顶元素）的操作是：先判栈空，栈不为空时，做下列操作：取栈顶元素、删除表头的结点、返回原栈顶元素。

主函数的功能是将一个十进制数转换成八进制数。利用栈将八进制的每一位数存放到栈中，退栈时将该数累加。例如：对十进制数 100，是将 4、40、100 进栈，退栈时是 100+40+4=144. 由于进栈的是整数，所以使用模板定义一个整数栈。

[8_17]程序实现如下：

```
#include <iostream.h>
#include <process.h>
template <class Type>
class node
{  public:
    Type data;
    node<Type> *next;
};
template<class Type>
class Stacklink
{  public:
    Stacklink();//构造函数
     ~Stacklink();//析构函数
```

```
    void push(Type value);//进栈
    void pop();//退栈
    Type topvalue();//取栈顶元素
    int isnull();//判栈空
private:
    node<Type> *top;
};
template<class Type>
Stacklink<Type>::Stacklink()
{  node<Type> *T;
   T=new node<Type>;//申请一个空间
   T->next=NULL;//指针项清0
   top=T;//栈顶指针指向新申请的空间
   //本例使用的是带头结点的单链表，第一个结点不会存储栈元素
}
template<class Type>
Stacklink<Type>::~Stacklink()//析构函数，删除单链表中的所有结点
{  node<Type> *N,*Temp;
   for(N=top;N!=NULL;)//从表头开始删除
   {  Temp=N;
      N=N->next;
      delete Temp;
   }
}
template<class Type>
```

```
void Stacklink<Type>::push(Type value)//入栈
{  node<Type> *T;
   T=new node<Type>;//申请一个空间
   if(T!=NULL)//判断申请是否成功
   {  T->next=top;//新申请的空间的指针域指向原栈顶元素
      T->data=value;//新申请的空间的数据赋值
         top=T;//栈指针指向新申请的空间
   }
   else
   {  cout<<"assigned failure!";
      exit(1);
   }
}
template<class Type>
int Stacklink<Type>::isnull()//判栈空
{  if(top->next==NULL)
   return 1;//如果链表中只剩一个结点（表头结点），栈为空返回 1
   else
      return 0;
}
template<class Type>
void Stacklink<Type>::pop()//退栈
{
   node<Type> *Temp;
   if(top->next==NULL)//判栈空
```

```
    {  cout<<"\n the stack is NULL!\n";
    //value=top->data;//取栈顶元素
    Temp=top;//Temp 指向栈顶元素
    top=top->next;//Top 指向栈顶元素的下一个元素
    delete  Temp;//删除 Temp 指向的结点
    // return(value);//返回原栈顶元素
}
template<class  Type>
Type  Stacklink<Type>::topvalue()
{
      if(top->next==NULL)
      {  cout<<"\n the stack is null!\n";
         return(0);
      }
      return  top->data;
}
void  main()
{  int  i=0;
   char  a[]="(a*(b+c)-b*d";
   Stacklink<char>  ss;//定义一个字符栈
   while(a[i]!='\0')//字符串未结束时循环
   {
        if(a[i]=='(')  ss.push('(');//如果是左括号进栈
        else  if(a[i]==')')//否则，如果是右括号
        {
```

```
            if(ss.isnull()) break;//若栈空则退出循环
            else if(ss.topvalue()=='(') ss.pop();//否则如果栈顶元素是
                                                 //左括号退栈
        }
        i++;
    }
    if(ss.isnull()&&a[i]=='\0')//如果循环操作是结束于字符串终止符并且
                               //栈空,说明括号匹配
        cout<<"\nmatch";
    else
        cout<<"\nnot match"<<endl;
}
```

说明：此程序使用的也是带头结点的单链表，与上面的题稍有不同的是，退栈和取栈顶元素是 2 个成员函数。另外在主函数中进栈的是字符，所以使用模板定义一个字符栈。主函数的思路是循环扫描字符串，对扫描到的字符做运算：如果是左括号进栈，继续扫描；否则如果是右括号并且栈空，表明右括号多了，退出扫描循环；如果是右括号并且栈顶元素是左括号则退栈。如果循环操作是结束于字符串终止符并且栈空，说明括号匹配。

[8_18]建立并演示链式队列类模板

```
#include <iostream.h>
#include <ctype.h>
#include <iomanip.h>
template <class Type>
class node
{  public:
```

```
    Type data;
    node<Type> *next;
};
template<class Type>
class queuelink
{  public:
    queuelink();
    ~queuelink();
    void put(Type value);
    Type get();
    void clear();
    void showqueue();
private:
    node<Type> *head;
    node<Type> *rear;
};
template<class Type>
queuelink<Type>::queuelink()
{
node<Type> *T;
  T=new node<Type>;
  T->next=NULL;
  head=rear=T;
}
template<class Type>
```

```
queuelink<Type>::~queuelink()
{  node<Type> *N,*Temp;
   for(N=rear;N!=NULL;)
   {  Temp=N;
      N=N->next;
         delete Temp;
   }
}
template<class Type>
void queuelink<Type>::put(Type value)
{  node<Type> *T;
   T=new node<Type>;
   T->next=NULL;
   head->data=value;
   head->next=T;
   head=head->next;
   cout<<"you have put a data into the queue!\n";
}
template<class Type>
Type queuelink<Type>::get()
{  Type value;
   node<Type> *T;
   if(head==rear)
   {  cout<<"\n The queue has no data!\n";
      return(0);
   }
   value=rear->data;
   T=rear;
```

```
    rear=rear->next;
    delete T;
    cout<<"\n Get "<<value<<" from queue.\n";
    return(value);
}
template<class Type>
void queuelink<Type>::clear()
{  head=rear;
   cout<<"\n ***Queue is empty! ***\n";
}
template <class Type>
void queuelink<Type>::showqueue()
{  node<Type> *T;
   if(head==rear)
   {  cout<<"\n the queue has no data! ";
      return;
   }
   cout<<"\n the content of queue:  \n";
   for(T=rear;T!=head;T=T->next)
   cout<<setw(5)<<T->data;
   cout<<"\n\n";
}
main()
{  cout<<"<p>------put data to queue\n";
   cout<<"<g>------get data from queue\n";
   cout<<"<l>------clear queue\n";
   cout<<"<s>------show the content of queue\n";
   cout<<"<q>------quit.....\n";
```

```
queuelink<char> ss;
char value;
char ch;
while(1)
{  cout<<"\n please select an item: ";
   cin>>ch;
     ch=toupper(ch);
     switch(ch)
     {  case 'p':cout<<"\n enter the value that";
              cout<<" you want to put: ";
                   cin>>value;
                   ss.put(value);
                   break;
        case 'g':value=ss.get();
                   break;
        case 'l':ss.clear();
                   break;
        case 's':ss.showqueue();
                   break;
        case 'q':return(0);
     default;
cout<<"\nyou have inputted awrong item! please try agiain!\n";
          continue;
}
```

第九章：C++的输入输出

[9_1]答：C++除了完全支持 C 语言的输入输出系统外，还定义了一套面向对象的输入输出系统。为什么 C++还要建立自己的输入输出系统呢？这是因为在 C++中需要定义众多的用户自定义类型，面向对象方法的数据封装性就是通过用户

所定义的类类型来体现的，而输入和输出操作是通过对用户定义的类对象的操作来体现的。但 C 语言的输入输出系统不支持用户自定义的类型。

[9_2]答：C++中包含几个预定义的流对象，它们是标准输入流（对象）cin（与标准输入设备相关联）、标准输出流（对象）cout（与标准输出设备相关联）、非缓冲型的标准出错流（对象）cerr（与标准错误输出设备相关联（非缓冲方式））、和缓冲型的标准出错流（对象）clog（与标准错误输出设备相关联（缓冲方式））。

在缺省情况下，指定的标准输出设备是屏幕，标准输入设备是键盘。在任何情况下，指定的标准错误输出设备总是屏幕。

[9_3]答：cerr 和 clog 之间的区别是，cerr 没有被缓冲，因而发送给它的任何内容都立即输出；相反，clog 被缓冲，只有当缓冲区满时才进行输出，也可以通过刷新流的方式强迫刷新缓冲区。

[9_4] 答：C++提供了两种进行格式控制的方法：一种是使用 ios 类中有关格式控制的成员函数进行格式控制；另一种是使用称为操作符的特殊类型的函数进行格式控制。

[9_5]答：C++中进行文件输入输出的基本过程是：必须首先创建一个流，然后将这个流与文件相关联，再打开文件，此时才能进行输入输出操作，操作完后再关闭这个文件。

[9_6]以前介绍的文件操作都是按一定顺序进行读写的，因此称为顺序文件。对于顺序文件而言，只能按实际排列的顺序，一个一个地访问文件中的各个元素。为了增加对文件访问的灵活性，C++在类 istream 及类 ostream 中定义了几个与随机移动文件指针相关的成员函数，使得可以在输入输出流内随机移动文件指针，从而可以对文件的数据进行随机读写。

[9_7]答：B

[9_8]答：C

[9_9]答：C　说明：dir 的取值有 3 种：ios::beg　ios::cur　ios::end

[9_10]答：A　说明：用 ios::app 打开的文件只能用于输出。

[9_11]答：第一个 width 是 5，第二个 width 是 8

[9_12]答：friend istream &operator>>(istream &in,Stock &st);

friend ostream &operator<<(ostream &out,Stock &st);

[9_13]答：

```
#include <iostream.h>
#include <iomanip.h>
double fact(int n);
main()
{  for(int n=5;n<11;n++)
   cout<<setw(2)<<n<<"!="<<fact(n)<<endl;
   return 0;
}
double fact(int n)
{  double factor=1;
   for(int i=n;i>=1;i--)
        factor*=i;
   return factor;
}
```

[9_14]答：

```
#include <iostream.h>
#include <math.h>
main()
{  double x;
   cout.precision(5);
   cout<<" x     ln e     log x  \n\n";
   for(x=2.0;x<=10.0;x++)
   {  cout.width(2);
     cout<<x<<"  ";
```

```
        cout.width(10);
        cout<<log(x)<<"  ";
        cout.width(10);
        cout<<log10(x)<<"\n";
    }
    return 0;
}
```

[9_15]答：

```
#include <iostream.h>
#include <fstream.h>
#include <ctype.h>
int main()
{   int flag=0, flag2=0;//flag==0 表示前面的字符是空格
                //flag==1 表示前面的字符是'i'
    int sum=0;          //sum 记录"is"的个数
    char ch;
    fstream in("file1.txt",ios::in);
    if(!in)
    {   cerr<<"Error open file.";
        return 0;
    }
    while((ch=in.get())!=EOF)
        if(ch==' ')
        {   if(flag!=1) flag=1;   }
        else
        {   if(flag==1&&ch=='i')
            flag2=1;
            if(flag2==1&&ch=='s')
```

```
            { sum++;
              flag2=0;
            }
&nb p;          flag=0;
        }
        cout<<sum;//输出"is"的个数
        cout<<endl;
        return 1;
}
```

[9_16]答：

```
#include <iostream.h>
#include <ctype.h>
#include <fstream.h>
main()
{  fstream in("file1.txt",ios::in);
   if(!in)
   {  cerr<<"Error open file.";
      return(0);
   }
   fstream out("file2.tet",ios::out);
   if(!out)
   {  cerr<<"Error open file.";
      return(0);
   }
   char ch;
   while((ch=in.get())!=EOF)
        out<<char(toupper(ch));
   return 0;
```

```
}
```

[9_17]答:

```
#include <fstream.h>
#include <ctype.h>
main()
{  fstream in;
   in.open("file1.txt",ios::in);
   if(!in)
   {  cerr<<"Error open fllec. ";
      return(0);
   }
   fstream out;
   out.open("file2.txt",ios::app);
   if(!out)
   {  cerr<<"Error open fllc. ";
      return(0);
   }
   char ch;
   while((ch=in.get())!=EOF)
         out<<char(toupper(ch));
   return 0;
}
```

[9_18]答:

```
#include <iostream.h>
#include <fstream.h>
main()
{  ofstream pout("stock.txt");
   if(!pout)
```

```
{  cout<<"cannot open phone file\n";
   return 1;
}
pout<<"shen fa zhan 000001\n";
pout<<"shang hai qi che 600104\n";
pout<<"guang ju neng yuan 000096\n";
pout.close();
return 0;
}
```