高等院校信息管理与信息系统专业系列教材

C++程序开发例题与习题

张基温 张 伟 编著

清华大学出版社
http://www.tup.tsinghua.edu.cn

内容简介

本书是一本通过编程实践引导学生掌握C++程序开发能力的教材。它可以与《C++程序开发教程》配合使用，也可以独立使用。全书共分7章：第1章主要培养学生解题时首先考虑如何定义类的习惯；第2章、第3章分别训练如何设计方法函数和数据结构；第4章进行重载与模板的有关编程训练；第5章培养学生如何通过类的派生来解决复杂问题；第6章是关于I/O流类库使用的练习；第7章是关于API和MFC编程的基本训练。

本书既有编程知识介绍，也有精选的例题，还有大量自测习题及其参考答案，可以作为教授C++程序设计课程的习题课教材，也可供读者自学。

- 计算机组成原理教程（第2版）
- 计算机组成原理教程题解与实验指导
- 计算机网络工程教程
- 计算机网络工程教程题解与实验指导
- 操作系统原理教程
- 数据库系统原理教程
- 运筹学模型与方法教程
- 运筹学模型与方法教程例题分析与题解
- 信息系统的开发与管理教程
- 信息系统开发方法教程
- 信息经济学教程
- 数据结构及应用算法教程
- 离散数学（第二版）
- 离散数学题解
- 计算机原理与系统结构（第二版）
- 信息管理学教程
- C++程序开发教程
- C++程序开发例题与习题
- 电子商务基础教程
- Java程序开发教程
- Java程序开发例题与习题
- 管理信息系统教程

ISBN 7-302-06714-7

9 787302 067146 >

定价：26.00元

高等院校信息管理与信息系统专业系列教材

C++ 程序开发例题与习题

张基温　张　伟　编著

清华大学出版社

北　京

内容简介

本书是一本通过编程实践引导学生掌握C++程序开发能力的教材。它可以与《C++程序开发教程》配合使用，也可以独立使用。全书共分7章：第1章主要培养学生解题时首先考虑如何定义类的习惯；第2章、第3章分别训练如何设计方法函数和数据结构；第4章进行重载与模板的有关编程训练；第5章培养学生如何通过类的派生来解决复杂问题；第6章是关于I/O流类库使用的练习；第7章是关于API和MFC编程的基本训练。

本书既有编程知识介绍，也有精选的例题，还有大量自测习题及其参考答案。可以作为教授C++程序设计课程的习题课教材，也可供读者自学。

版权所有，翻印必究。
本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

图书在版编目(CIP)数据

C++程序开发例题与习题/张基温、张伟编著. 北京：清华大学出版社，2003
(高等院校信息管理与信息系统专业系列教材)
ISBN 7-302-06714-7

Ⅰ. C… Ⅱ. ①张… ②张… Ⅲ. C语言—程序设计—高等学校—习题 Ⅳ. TP312-44

中国版本图书馆CIP数据核字(2003)第045590号

出 版 者：清华大学出版社　　地　址：北京清华大学学研大厦
http://www.tup.com.cn　　邮　编：100084
社 总 机：010-62770175　　客户服务：010-62776969
组稿编辑：范素珍
文稿编辑：范素珍
印 刷 者：北京嘉实印刷有限公司
发 行 者：新华书店总店北京发行所
开　本：787×1092 1/16　印张：21　字数：481千字
版　次：2003年7月第1版　2003年7月第1次印刷
书　号：ISBN 7-302-06714-7/TP·5007
印　数：1～5000
定　价：26.00元

出版说明

20世纪三四十年代，一直摸索着前进的计算技术与刚走向成熟的电子技术结缘。这一结合，不仅孕育了新一代计算工具——电子计算机，还产生了当时谁也没有料到的巨大效应：电子计算机——这种当初为计算而开发出来的工具，很快就超出计算的范畴，成为"信息处理机"的代名词；人类开始能够高效率地开发并利用信息；信息对人类社会的作用得以有效地发挥，并逐步超过材料和能源成为人类社会的重要支柱；信息产业急剧增长，信息经济高度发展，社会生产力达到了新的高度；人们的信息化意识不断加强，人类在信息资源方面开始更加激烈地竞争，社会发展走上信息化轨道。

文化是时代的精髓，是特定的人群在一定的历史时期、一定的地域范围对其生产和生活模式、思维和行为方式的觉悟和理性化，它伴随着人类创造和使用工具能力的提高而不断发展。文者，经天纬地也；化者，变化、改变、造化、习俗、风气也。也可以说，文化作为社会的人们在生产和生活中思维和行为方式的理性化，是文治和教化的结果。因此，文化具有区域性、群体性和时代性。在信息时代的帷幕刚刚拉开、新时代的气息开始弥漫社会各个角落的20世纪70年代，先觉们就已开始创办以加速信息化的进程为宗旨、以培养信息资源开发人才为目标的信息管理与信息系统专业。

从与信息有关的学科纵向来看，信息管理与信息系统专业处于信息学、信息技术、信息管理、信息经济、信息社会学这个层次结构的中间，它下以信息学和信息技术为基础，上与信息经济和信息社会学相联系。从其涉及的学科横向来看，它处在管理学、信息科学与技术和有关专业领域的交叉点上。它对技术有极高的要求，又要求对组织有深刻的理解，对行为有合理的组织，反映了科学与人本的融合的特点。这种交叉与融合正是信息管理与信息系统专业的最重要的特征，是别的学科或专业难以取代和涵盖的。

我国的信息管理与信息系统专业创建于20世纪70年代末。在近20年的时间里，已发展到151个点，成为培养信息化人才的重要领域。其发展速度之快、影响之深远已令世人和学术界刮目相看。然而作为一个新的、特别是与各行各业关系极为密切的专业，其课程体系、教学内容以及教学方法、手段，都要经历一个逐步完善、逐步成熟的过程，其教材体系的建设更需要较长期的实践和探索。没有这样一个过程，具有专业特点、符合中国实际的教材体系是不会被建立的。近20年来，大家一直在课程体系的完善和建设有自己专业特点的教材方面不断进行探讨。1991年全国10所财经类院校的经济信息管理专业的负责人在太原召开第一次研讨会。以后，1993年在大连、1995年在武汉、1997年在烟台，又有更多的院校参加到了这一研讨之中。这些研讨活动得到了国家教委有关部门的赞许和支持。通过研讨，大家在建设具有专业特点的教材体系、改变简单照搬其他专业教材上取得了共识。在武汉会议之后，即着手进行系列教材的编写工作。经协商，由张基温教授担任主编，由魏晴宇教授、陈禹教授担任顾问。

这套教材是我国信息管理与信息系统专业的第一套教材。尽管编写者为它付出了巨大的辛劳，但在实践中我们也深深地感到了时代的鞭策和工作的难度。一方面，席卷全球的信

息化大潮已经使信息、信息管理、信息系统成为全社会关注的热点，人们对其期望和要求越来越高；另一方面，在世纪之交的今天，作为现代社会先导技术的信息技术和相关学科的更新速度在不断加速，多种社会因素相互渗透、相互影响，前所未有的新情况、新问题给专业的建设带来很大的困难。当然，这些对我们专业的发展和建设也是一种动力和机遇。为此，在这套教材问世之际，我们再一次表示一个心愿：希望与全国的同行共勉，在教材和专业建设上齐心协力，做出更大贡献。也由于如上种种，这套教材不会是完整的，也不会是完美的，一定存在这样那样的不足或错误，我们将会不断补充，不断修改，不断完善。对于它的任何建设性意见，都是我们非常期盼的。为此，这一套教材将具有充分的开放性：每一本教材都是一个原型，每一位有志者对它的建设性意见都将会被采纳，并享有自己的知识产权，以使它们逐步成为精品。

全国高等院校计算机基础教育研究会
财经信息管理专业委员会
信息管理与信息系统专业系列教材编委会
1997年8月

前　言

计算机应用总是与程序设计联系在一起的。但是，随着信息时代的到来，程序设计不仅是培养学生计算机应用开发能力的技术课程，更重要的是它应当作为训练信息时代基本素养——计算思维能力的重要公共基础课程。

程序设计是一门实践性极强的课程。要学好程序设计课程，提高程序设计的能力，主要要进行两方面的训练：熟练地掌握一种计算机语言，正确地运用语言机制，充分地利用语言提供的环境和资源；掌握基本的程序设计方法，不断积累程序设计的经验，使自己的程序设计能力不断提高。简单地说，就是在正确的程序设计方法指导下多练。本书就是为这样的读者群编写的一本教材：既可以是教授 C++ 程序开发课程的习题课的教材，也可以是读者自学的教材。

为了便于读者自学，本书每一章的内容都分为 3 部分：

- x.1　编程知识
- x.2　例题示范
- x.3　自我测试

  x.3.1　自测习题

  x.3.2　参考答案

本书内容的编排顺序与我编著的《C++ 程序开发教程》(清华大学出版社出版)相同：

第 1 章主要培养学生解题时首先考虑如何定义类的习惯；

第 2 章、第 3 章分别训练如何设计较复杂的函数和数据结构；

第 4 章是有关重载与模板的编程训练；

第 5 章培养学生如何通过类的派生来解决复杂问题；

第 6 章是关于 I/O 流类库使用的练习；

第 7 章是关于 API 和基于 MFC 编程的基本训练。

全书主要由我和张伟编写。杨开岌也参加了第 4、5、6 章例题的部分程序调试工作。

张基温

2003 年 5 月

目　录

第 1 章　C++ 程序开发初步 …… 1

1.1　编程知识 …… 1

1.1.1　对象、类与类界面 …… 1

1.1.2　C++ 中的函数及其类型 …… 2

1.1.3　函数结构与函数定义 …… 3

1.1.4　语句与表达式 …… 4

1.1.5　数据表示 …… 6

1.1.6　函数原型与函数声明 …… 10

1.1.7　函数调用的作用与特点 …… 11

1.1.8　引用类型 …… 11

1.1.9　友元 …… 12

1.2　例题示范 …… 12

1.3　自我测试 …… 16

1.3.1　自测习题 …… 16

1.3.2　参考答案 …… 20

第 2 章　算法设计与程序结构 …… 30

2.1　编程知识 …… 30

2.1.1　关系运算与选择结构 …… 30

2.1.2　循环结构 …… 32

2.1.3　变量的存储属性 …… 33

2.1.4　多文件程序结构与程序项目 …… 35

2.1.5　文件包含与条件编译 …… 36

2.2　例题示范 …… 37

2.2.1　控制结构基础 …… 37

2.2.2　穷举 …… 41

2.2.3　递推 …… 47

2.2.4　模拟 …… 51

2.2.5　递归 …… 56

2.2.6　对象(变量)的存储属性 …… 58

2.3　自我测试 …… 63

2.3.1　自测习题 …… 63

2.3.2　参考答案 …… 66

第 3 章　数据结构 …… 85

3.1　编程知识 …… 85

3.1.1　数组 …… 85

3.1.2　指针 …… 86

3.1.3　数组与指针 …… 87

3.1.4　字符串 …… 89

3.1.5　枚举 …… 89

3.1.6　共用体 …… 90

3.2　例题示范 …… 91

3.3　自我测试 …… 105

3.3.1　自测习题 …… 105

3.3.2　参考答案 …… 110

第 4 章　重载与模板 …… 135

4.1　编程知识 …… 135

4.1.1　函数名重载 …… 135

4.1.2　模板 …… 136

4.2　例题示范 …… 137

4.3　自我测试 …… 149

4.3.1　自测习题 …… 149

4.3.2　参考答案 …… 157

第 5 章　程序的类层次结构 …… 176

5.1　编程知识 …… 176

5.1.1　类的继承与派生 …… 176

5.1.2　虚函数 …… 177

5.2　例题示范 …… 178

5.3　自我测试 …… 186

5.3.1　自测习题 …… 186

5.3.2　参考答案 …… 196

第 6 章　I/O 流类库 …… 231

6.1　编程知识 …… 231

6.1.1　流类库及其结构 …… 231

6.1.2　流的格式化控制 …… 234

6.1.3　流的出错处理 …… 236

6.1.4　文件流操作 …… 236

6.2　例题示范 …… 240

6.3 自我测试 …… 250
6.3.1 自测习题 …… 250
6.3.2 参考答案 …… 251

第7章 Visual C++ 程序设计 …… 270

7.1 编程知识 …… 270
7.1.1 Windows 应用程序设计 …… 270
7.1.2 基于 MFC 的 Visual C++ 编程模式 …… 274
7.2 例题示范 …… 279
7.3 自我测试 …… 297
7.3.1 自测习题 …… 297
7.3.2 参考答案 …… 298

第1章 C++程序开发初步

1.1 编程知识

1.1.1 对象、类与类界面

面向对象的模型认为世界是由一些对象(objects)组成的，每一个对象包括属性和方法(C++中称为成员函数)两部分；属性是描述对象特征的一些数据或数据结构，方法表明对象的变化或一个对象对其他对象的作用，对象间通过消息传递进行通信；一个问题从原始状态到目标状态，是通过对象间相互传递消息时进行的。

但是对象是形形色色的，处理起来非常繁琐。为此，基于分类学的原理，将对象抽象为类(class)。不过，程序处理时，还是要针对具体的对象。这时，就要由类生成一个具体的对象。面向对象的程序设计的关键是如何找到问题中包含哪些类，然后定义这些类的界面。

类的定义包含两部分：

(1) 数据成员：描述某类对象的属性；

(2) 成员函数：描述某类对象的行为和变化等。

类成员分为公开的(public)与私有的(private)两类。私有成员不允许外界直接访问，只能由本类的成员函数访问。外界与对象间的信息传送只能通过公开成员进行。或者说，公开成员是对象的外部接口。一般说来，成员函数是外界可以访问的部分，由关键词 public 说明为公开的；数据成员是对象隐藏的部分，由关键词 private 说明为私有的。默认的成员都是私有的，即当私有成员放在类界面的开头部分时，关键词 private 可以默认。

下面是C++中类的一般结构：

```
class 类名 {
    private:
        私有成员声明
    public:
        公开成员声明
};
```

只含有成员声明语句的类界面，也称为类的声明。

在程序中使用的类有两种：

(1) 程序员自己定义的类；

(2) 系统事先定义好存放在类库中的类。

类库中的类的声明通常按照类别集中放在一些头文件中。在程序中使用系统类库中的某个类之前，要先把含有它的声明的头文件用文件包含命令#include包含进来。

1.1.2 C++ 中的函数及其类型

函数是 C++ 程序中组织过程的机制。在 C++ 中，函数主要有 3 大类：

(1) 成员函数；

(2) 主函数；

(3) 其他具有特别功能的函数。

1. 成员函数

成员函数定义在具体的类中，用于表现类的一个动态属性。对象之间的通信——消息传递，主要通过类的成员函数之间的调用和参数传递进行。

在成员函数中，有两种特殊的成员函数：构造函数和释放函数。

(1) 构造函数用于对象的创建——生成：

① 给对象一个名字(像变量名)；

② 给对象开辟一个存储空间；

③ 给对象中的成员变量初始化。

构造函数具有如下特征：

① 与类同名；

② 不标以返回类型(它的返回值是隐含的，是指向类本身的值)；

③ 构造函数允许参数缺省调用。C++ 编译器总要为它自动建立一个无参构造函数——缺省构造函数。

(2) 释放函数(也称析构函数)用于对象的撤销，在对象撤销时被自动调用，用于执行一些清理任务，如释放由构造函数分配的内存等。释放函数有如下一些特点：

① 与类同名，之前冠以波浪号，以区别于构造函数；

② 不指定返回类型；

③ 不能指定参数。

当有释放函数的显式定义时，应当在类定义中显式地声明(也可以是内联的)；当没有显式地给出释放函数的定义时，它是类的隐含的成员。隐含的释放函数是一个“空”的函数体，它只执行清理任务。

2. 主函数

主函数是 C++ 中起总控与总调度角色的函数。从宏观上说，一个 C++ 程序中的对象间主要消息传递都是在主函数中进行的，或者由主函数控制进行的。每一个 C++ 程序都有并且只有一个主函数(Windows 程序中的主函数为 WinMain())。主函数的基本框架如下：

```
int main()
{
    ...
}
```

任何一个C++程序都是执行主函数，即从main()的前花括号起开始执行，一直到main()的最后一个花括号为止，里面是一些与对象的生成、撤销以及对象间消息传递有关的语句。

3. 其他具有特别功能的函数

其他具有特别功能的函数主要用于主函数或成员函数中，用于实现一些特定的功能。将这些功能的实现语句用一个函数调用语句代替，可以使程序简洁，具有较好的可阅读性。

这种函数，可以由程序员设计，也可以从系统函数库中调用合适的库函数，也可以使用其他类的成员函数（包括类库中的类的成员函数）。

1.1.3 函数结构与函数定义

一个C++函数的定义由函数头与函数体两部分组成，形式如下：

```
函数首部
{
        函数体
}
```

1. 函数首部

函数首部是函数的接口部分，其组成形式如下：

类型名　函数名(形式参数表列)

说明：

(1) 函数类型规定为函数返回值的类型。void类型的函数无返回值，并且函数类型定义为void才不返回任何值。因为非void的函数体中无返回表达式的return;语句也要返回一个相应类型的值，只是这个返回值是随机的，它只执行将流程返回以及撤销函数中的参数变量的存储空间的操作。

(2) 函数名是函数的标识，它应是一个有效的C++标识符。

(3) 形式参数表列是括在圆括号中的0个或多个以逗号分隔的形式参数。它定义了函数将从调用函数中接收的数据个数和每个数据的类型。

(4) main()函数是一个特殊的函数。

① 它的名字是固定的；

② 它是由操作系统调用的一个函数，返回值可以由操作系统接收。其返回值可以是void型或int型（可以默认）。当其返回值为int型时，用0或非0，告诉操作系统程序是否正常结束。

2. 函数体

一个函数体是一个语句块，是用一对花括号封装的语句序列。它描述了函数实现一个功能的过程。它最后执行一个函数返回，返回的作用是：

(1) 将流程从当前函数返回其上级(调用函数);
(2) 撤销函数调用时为各参数及变量分配的内存空间;
(3) 向调用函数返回顶多一个值。
return 语句的一般形式是:

return 表达式; 或 return (表达式);

一个函数体中可以有多个 return 语句,但只能通过一个 return 语句执行返回操作。这时每个 return 语句的返回值的类型都应与函数原型一致。如果这个表达式的类型与函数定义或声明时的类型不一致,则将被强制转换成函数定义或声明时的类型。

3. 类的实现——成员函数的定义

类的实现,实际上是要进一步定义它的成员——成员函数。成员函数是类定义中用以描述类对象行为的成员。在 C++ 中称为成员函数(在面向对象的方法学中也称为方法)。成员函数的定义有如下格式:

类型 类名 :: 成员函数名(参数表)
{
函数体
}

"::"称为范围运算符。在本例中,用于说明后面的函数定义属于哪个类的范围。
除特殊指明外,成员函数操作的是同类某一对象中的数据成员。

4. 内联函数

在调用与返回过程中,需要付出一定的时间与空间代价用于处理现场。当函数较小,又反复使用时,处理现场的开销比重会急剧增大。内联函数可以大大提高运行速度,节省开销。

内联函数有两种定义方法:一种是将函数定义内嵌在类的界面中,一种是定义在类界面的外部。当定义在类界面外部时,要用"inline"修饰;内嵌在类界面中时,可以不加标识符"inline"。应当注意,由于编译器必须知道内联函数的函数体才能进行内联替换,因此,内联函数在程序中被调用之前,必须被编译器看见,即内联函数必须在调用之前定义或声明。

1.1.4 语句与表达式

1. C++ 语句

函数以及类的基本组成元素是语句。语句都以分号结尾。C++ 的语句可以分为 3 种类型:

(1) 声明语句,包括变量和对象的声明语句、类的声明语句、函数声明语句。
(2) 控制语句——用于改变程序的流程,如函数的调用、返回语句等。

(3) 表达式语句——实现有关数据的变换和运算。

2. 运算符与表达式

运算符是一种程序记号，它作用于操作数而触发某种操作。由运算符和操作数组成的符号序列称为表达式。学习运算符要注意 3 个问题：

(1) 运算符的含义；

(2) 运算符能作用于哪些数据；

(3) 当一个表达式中含有多个运算符时，以什么顺序运算。

在 C++ 中，运算符在表达式中的运算顺序由运算符的 3 个主要特性决定：

(1) 运算符的优先级别；

(2) 运算符与操作数的结合方向；

(3) 运算符的先后排列顺序。

表 1.1 中给出了 C++ 运算符及其优先级别和它们的结合方向。

表 1.1　C++ 运算符的优先级别

优先级类别	符　号	操作描述	结合方向
1. 最高级	( )	函数调用	→
	[ ]	数组下标	
	->	成员间接选择	
	::	作用域限定	
	.	成员直接选择	
2. 一元运算	!	逻辑非	←
	~	按位求补	
	+	正	
	-	负	
	++	+1	
	--	-1	
	&	求地址，引用	
	*	指针，间接访问	
	sizeof	求存储空间	
	new	申请动态空间	
	delete	撤销动态空间	
3. 成员访问	.*	间接引用	→
	->*	指针间接引用	
4. 乘除模	*	乘	
	/	除	
	%	模(求余)	
5. 加减	+	加	
	-	减	
6. 位移	<<	左移	
	>>	右移	
7. 关系	<	小于	→
	<=	小于等于	
	>	大于	
	>=	大于等于	
8. 关系	==	等于	
	!=	不等于	
9. 位与	&	按位与	
10. 位异或	^	按位异或	
11. 位或	|	按位或	
12. 与	&&	逻辑与	
13. 或	||	逻辑或	
14. 条件	?:	则，否则	←
15. 赋值	=	赋值	
	*=	赋值乘	
	/=	赋值除	
	+=	赋值加	
	-=	赋值减	
	&=	赋值与	
	^=	赋值异或	
	|=	赋值或	
	<<=	赋值左移	
	>>=	赋值右移	
16. 逗号	,	顺序求值	→

1.1.5 数据表示

1. 数据类型

C++ 程序中的数据都属于一定的类型。使用类型的意义在于对数据在如下几个方面进行规范化：

- 为数据分配相应大小的存储空间；
- 确定数据值的范围；
- 确定数据的表示精度；
- 确定数据可以进行的运算种类。

(1) 基本数据类型

C++ 中数据类型非常丰富，表 1.2 为 32 位系统中几种最常用的数据类型的比较。

表 1.2 32 位系统中几种最常用的数据类型的比较

数据类型关键字	意 义	存储空间/位	表数精度/位(有效数字)	表数范围	运算类型
char	字符	8			+、-、%
short	短整数	16		-32 768～32 767	+、-、*、/、%
int	整数	32		-2 147 483 648～2 147 483 647	
float	单精度浮点	32	7	$\pm(3.4\times10^{-38}\sim3.4\times10^{38})$	+、-、*、/、%
double	双精度浮点	64	16	$\pm(2.23\times10^{-308}\sim1.79\times10^{308})$	
long double	长双精度	80	18	$\pm(3.37\times10^{-4932}\sim1.18\times10^{4932})$	

(2) 字符类型

在计算机中，字母、标点符号等统称为字符，它们在机器内部也要用 0、1 进行编码表示，为此要采用某种规定的编码规则。目前应用最为广泛的编码规则是 ASCII 码。表 1.3 为 ASCII 码字符表。

表 1.3 ASCII 码(7 位码)字符表

项目	列	0	1	2	3	4	5	6	7
行	$b_6\,b_5\,b_4$ / $b_3\,b_2\,b_1\,b_0$	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u

续表

项目	列	0	1	2	3	4	5	6	7
行	$b_6 b_5 b_4$ / $b_3 b_2 b_1 b_0$	000	001	010	011	100	101	110	111
6	0110	ACK	SYM	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1101	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESU	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1101	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

(3) 转义字符序列(见表 1.4)

表 1.4 转义字符序列

序列	值	字符	功能
\a	0X07	BEL	警告响铃
\b	0X08	BS	退格
\f	0X0C	FF	走纸
\n	0X0A	LF	换行
\r	0X0D	CR	回车
\t	0X09	HT	水平制表
\v	0X0B	VT	垂直制表
\\	0X5c	\	反斜杠
\'	0X27	'	单引号
\"	0X22	"	双引号
\?	0X3F	?	问号
\o	整数	任意	0:最多为 3 位的八进制数字串
\xH	整数	任意	H:十六进制数字串

(4) 字符串

字符串常量也称字符串文字或字符串,它们是括在一对双撇号内的字符序列。

字符串常量与字符型常量在内部存储上是不同的。一个字符串的存储形式是给定的字

符序列后加一个空字符('\0'),而字符型常量不加.

(5) sizeof 运算符

sizeof 是一个单目运算符,用以计算操作数在内存中占用的字节数。

2. C++ 程序中数据的两种存在形式——变量与常量

(1) 变量的意义

① 变量指明了程序中使用的一个被命名的存储区域(称程序实体);

② 属于一个特定的类型;

③ 具有可由程序修改的值　　变量值。

(2) 变量名的命名规则

① 字符的使用:

- 大小写字母　　可用于变量名的任何位置;
- 下划线　　可用于变量名的任何位置;
- 数字　——不能用于变量名的首部。

② 长度:C++ 变量名的长度不受限制,但 Visual C++ 视前 247 个字符为有效,别的版本有别的规定。

③ 不可使用 C++ 保留字。保留字在书写程序时有特别的含义。表 1.5 为 C++ 保留字。

表 1.5　C++ 保留字

关键字	用途	说　明
char	数据类型	一个字节长的字符值
short		短整数
int		整数
unsigned		无符号类型,最高位不作符号位
long		长整数
float		单精度实数
double		双精度实数
struct		用于定义结构体
union		用于定义共用体
void		空类型,用它定义的对象不具有任何值
enum		定义枚举类型
signed		有符号类型,最高位作符号位
const		表明这个量在程序执行过程中不可变
volatile		表明这个量在程序执行过程中可被隐含地改变
typedef		用于定义同义数据类型
auto	存储类别	自动变量
register		寄存器类型
static		静态变量
extern		外部变量声明

续表

关键字	用途	说　明
break	流程控制	退出一层循环或 switch 语句
case		switch 语句中的情况选择
continue		跳到下一轮循环
default		switch 语句中其余情况标号
do		在 do …while 循环中的循环起始标记
else		if 语句中的另一种选择
for		带有初值、测试和增量的一种循环
goto		转移到标号指定的地方
if		语句的条件执行
return		返回到调用函数
switch		从所有列出的动作中做出选择
while		在 while 和 do…while 循环中语句的条件执行
class	类相关	定义类标识
public		公开成员说明
private		私有成员说明
protected		保护成员说明
virtual		指明虚基类或虚函数
friend		友元声明符
template		定义模板
operator		运算符函数名
try	异常处理	异常处理
throw		异常抛出
catch		捕捉异常类型
new	动态分配	申请动态内存块
delete		释放动态内存块
inline	其他	内联函数
asm		汇编声明
sizeof		计算表达式和类型的字节数

④ 清晰有序。

当程序中变量很多时，毫无规则地命名变量名，会造成程序的混乱。通常采用"见名知义"的原则。此外，Windows 程序员还广泛采用匈牙利人 Charles Simonyi 提出的将类型作为变量名前缀的命名方法——通常称为匈牙利命名法。表 1.6 为部分常用匈牙利前缀。

(3) 字面常量与符号常量

① 用 const 定义的符号常量。

② 用宏定义符号常量，形式为：

#define　宏名　宏体

表 1.6 部分常用匈牙利前缀

匈牙利前缀	数据类型	变量名举例
a 或 ar	数组	arAge
b	BULL(布尔值)	bDone、b
by	BYTE(无符号字符)	byCount、by
c 或 ch	char	c、ch
d 或 dbl	double	d、dbl、dCost、dblCost
dw 或 w	无符号整数	wNumber
f 或 fl	float	f、fl、fCost、flCost
fn	函数	fnFun1
h	句柄	hWnd
L 或 l	long	L、l、lCount
m	类的数据成员	
n 或 i	int	n、i、nCount、iCount
p	指针	pInt、pWnd
s	字符串	sName
sz	"0"结束的字符串	sz、szMystring
x、y	无符号整型或坐标	

1.1.6 函数原型与函数声明

1. 函数原型

函数原型也称函数模型，是由函数定义中抽取出来的能代表函数应用特征的部分：

(1) 函数类型；

(2) 函数名；

(3) 参数个数及其类型。

函数原型的形式为：

类型 函数名(参数类型 1,参数类型 2,…,参数类型 n)

2. 函数提前声明

C++ 要求在某函数被调用之前，应当让编译器知道该函数原型，以便让编译器利用函数原型提供的信息去检查调用的合法性，强制参数为适当类型，以保证参数的正确传递。提前声明来自：

(1) 从函数定义中抽取函数原型——定义在前，调用在后；

(2) 程序员在调用之前用函数原型对函数进行声明——定义在后，调用在前；

(3) 从头文件中 库函数、类库中的类的成员函数等。

1.1.7 函数调用的作用与特点

1. 函数调用的作用

(1) 用实参数(向形式参数传递消息)一 数据；

(2) 中断现行(调用)函数，把流程转向(被调用)函数的入口处，开始执行被调函数。

函数调用表达式的形式为：

函数名(实参表列)

2. C++ 函数调用的特点——传值调用

传值调用(值传递)即调用函数向被调用函数传送的是参数值。

1.1.8 引用类型

1. 引用类型的概念

引用(reference)是 C++ 为变量建立的一种别名机制。声明一个引用可以采用下面的格式：

类型 & 引用名 = 变量名;

引用具有如下性质：

(1) 对引用的存取就是对变量的存取；

(2) 引用与被引用的变量具有相同的实体；

(3) 创建常量的引用时，系统将建立一个临时变量；

(4) 不能建立引用的引用，不可使其作为另一个变量的别名。

2. 引用参数

引用参数传送的不是变量(参数)的值，而是变量(参数)的名字。

3. 复制构造函数

用一个对象初始化为另一个对象 这时，需要一种特殊的构造函数 复制构造函数。其特殊之处在于它只有一个参数，并且是其所在类的对象的引用。其原型为

X :: X(const X&)

其中，X 为复制构造函数所在类的类名。

复制构造函数用于既有对象初始化一个新对象时，它在以下 3 种时刻被调用：

(1) 声明并初始化对象变量时；

(2) 对象作为函数参数时；

(3) 对象作为函数值被返回时。

对于任何一个类X，如果程序员没有显式地为其定义一个复制构造函数，C++编译器都会自动生成一个形如X::X(const X &)的默认构造函数。

对象作为参数的函数传值调用，实际上是用实参初始化形参，为此要调用复制构造函数。这个对象形参将在函数返回时被撤销，为此要调用释放函数。当函数返回一个对象时，要先在函数内部创建一个隐含的对象，为此要调用复制构造函数。该隐含对象的生存期结束时，也要调用释放函数。

对象作参数或使用返回对象的函数，都要正确地使用复制构造函数，否则会出现错误。

1.1.9 友元

1. 友元函数

在某个类定义中用friend声明了一个外部函数(也许是其他类的一个成员)后，就好像宣布了一种"亲友"关系，这个外部函数便可以例外地访问该类的任何私有成员。

友元函数常常是另一个类的一个成员函数，这样通过友元函数可以使一个类对象直接访问另一个类对象的私有成员。这种成员函数不仅可以访问自己所在本类对象中的私有成员，还可以访问friend声明所在类的对象中的私有与公开成员。

当友元函数是另一个类的成员函数时，还应当在另一个类中声明它的友元关系。声明语句的格式为

friend 函数类型　类名::函数名(参数表列);

2. 友元类

当一个类要和另一个类协同工作时，使一个类成为另一个类的友元类是很有用的。这时友元类中的每一个成员函数都成为对方的友元函数。

3. 友元关系的特点

(1) 单向性——不具有交换性；

(2) 局部性——只在两个类之间有效；

(3) 无传递性。

1.2 例题示范

例1.2.1 设计一个电子日历类Date的界面。

解：设计一个类时，一般先分析对象的应用环境，分析对象应具有的静态属性和动态属性，由此抽象出类数据成员和方法成员，从而得到类的界面。

1. 电子日历类Date的静态属性

(1) Year(表示年)；

(2) Month(表示月)；

(3) Day(表示日)。

2. 电子日历类 Date 的动态属性

使用一个电子日历时，进行的操作与对应的方法有：

(1) 创建一个电子日历对象　　构造方法，它与类同名；

(2) 设定电子日历的日期　　方法 getDate()；

(3) 电子日历给出日期　　方法 getDate()；

(4) 让电子日历进行相关计算：

① 计算与当前日期相距多少天　——方法 getDays(Date date1)；

② 求出与当前日期相距多少天的日期　　方法 getOtherDate(int d)。

3. 电子日历类 Date 中各成员的访问权限的确定

正像对电子日历不能让用户拆开外壳直接去拨动显示装置中的元件，而只能让用户使用电子日历上的有关按键进行操作一样，数据成员(Year、Month 和 Day)是不允许外部操作的，因此访问权限应为 private；而方法好像是电子日历上的有关按键，专门供用户使用的，因此访问权限应为 public。Year、Month 和 Day 只能由方法成员进行操作。

4. 方法的类型的确定

单个的 Year、Month 和 Day 都是些整数，所以它们都是 int 类型。

在将要设计的类 Date 中，方法给出的结果有如下 4 种情形。

(1) 给出天数，应为 int 类型，如计算与当前日期相距多少天的方法 getDays(Date date1)。

(2) 给出日期，应为 Date 类型，如：

① 计算与当前日期相距多少天的日期的方法 getOtherDate(int d)；

② 给出日期的方法 getDate()，

(3) 只进行一些操作，没有输出的方法，应为 void 类型，如设定电子日历日期的方法 getDate()。

(4) 不能有任何修饰的方法　　构造方法。

5. 方法参数的确定

一个方法需要外界(调用者)向它传递的数据，称为参数。对于类 Date 来说，它的方法按照参数可以分为 4 种情形：

(1) 只要传递一个整数——天数的方法：求出与当前日期相距多少天的日期方法 getOtherDate(int d)；

(2) 只要传递一个日期的方法：计算与当前日期相距多少天的方法 getDays(Date date1)；

(3) 不需要传递数据，好像只要按键就可以的方法：

① 设定电子日历日期的方法 getDate()；

② 给出日期的方法 getDate()。

(4) 要具体给出 Year、Month 和 Day 的值的方法　　构造方法。好像制造好一个电子日历时要先给它一个具体的日期值一样。

根据上述分析，可以得到 Date 类界面如下：

```
class Date                                  // 定义 Date 类
{
    //以下是 Date 类的数据成员
  private:
    int Year;                               // 年
    private int Month;                      // 月
    private int Day;                        // 日

  // 以下是 Date 类的方法成员
  public:
    Date(int year,int month,int day);       // 带参数的构造方法
    void setDate();                         // 设定日期的方法
    Date getDate();                         // 给出日期的方法
    int getDays(Date date1);                // 计算与当前日期相距多少天
    Date getOtherDate(int d);               // 计算与当前日期相距多少天的日期
}
```

当然，上面定义的仅仅是类 Date 的界面，还不是一个完整的类。因为每一个方法如何实现还没有给出进一步的定义。

例 1.2.2　设计一个长方体类 Cuboid。

解：

1. 类界面的设计

设计一个类时，一般先归纳抽象出类包含的共同特征(属性和方法)，然后设计类所拥有方法的实现代码。Cuboid 类包括的属性有 Length(表示长度)、Width(表示宽度)和 High(表示高度)。它包括的方法有构造函数 Cuboid()(初始化属性的值)、getLength()(获取属性长度的值)、getWidth()(获取属性宽度的值)及 getHigh()(获取属性高度的值)。设计的 Cuboid 类的界面如下：

```
class Cuboid
{
    private:
        int Length;                         // 长度
        int Width;                          // 宽度
        int High;                           // 高度
    public:
        Cuboid (int year,int month,int day);  // 构造方法
        int getLength();                    // 获取属性长度的值
```

```
        int getWidth();                    // 获取属性宽度的值
        int getHigh();                     // 获取属性高度的值
};
```

2. 类的成员函数的实现

```
// 构造函数的实现
Cuboid::Cuboid(int length,int width,int high)
{
    Length=length;                         // 置 Length 初值为 length
    Width=width;                           // 置 Width 初值为 width
    High=high;                             // 置 High 初值为 high
}

// 成员函数 getLength 的实现
int Cuboid::getLength()
{
    return (Length);                       // 获取属性长度的值
}

// 成员函数 getWidth 的实现
int Cuboid::getWidth()
{
    return (Width)                         // 获取属性宽度的值
}

// 成员函数 getWidth 的实现
int Cuboid::getWidth()
{
    return (High);                         // 获取属性高度的值
}
```

3. 测试函数的设计

测试时先创建一个 Cuboid 类的对象，然后通过对象名来调用类的公有函数（发送消息）。

main 函数设计如下：

```
#include "iostream.h"
main()
{
    Cuboid cub(20,15,80);          // 创建一个 Cuboid 对象 cub,
                                   // 同时调用构造函数初始化私有成员
```

```
    cout << "长方体的长度:"<< cub.getLength() << "米" << endl;
    cout << "长方体的宽度:"<< cub.getWidth() << "米" << endl;
    cout << "长方体的高度:"<< cub.getHigh() << "米" << endl;
}
```

本程序运行结果如下：

```
长方体的长度：20 米
长方体的宽度：15 米
长方体的高度：80 米
```

1.3 自我测试

1.3.1 自测习题

题 1.3.1 考察一些熟悉的对象，如学生、课程、教师、教室等，假如某个教学管理软件要用到这些对象，请设想一下这些对象的使用环境，据此抽取这些对象的属性和行为，并设计出相应的类的界面。

题 1.3.2 定义一个时间类 Time 的界面。它能提供由时、分、秒组成的时间，并可以进行时间的设置。

题 1.3.3 定义并测试一个名为 MyRectangle 的矩形类，它有长、宽两个属性，有计算矩形面积的方法。

题 1.3.4 定义一个盒子类 Box，它能提供盒子的体积和表面积。

题 1.3.5 下列哪些是合法的 C++ 常数？哪些是非法的 C++ 常数？

2002　.123　(−368)　−479　25.　+ −256　− −863　0x12.5

03.14　0187　0xABQD　ABF　0.618e+8　3.68E−(3+5)　0x89e5

题 1.3.6 下列哪些是合法的 C++ 变量名？哪些是非法的 C++ 变量名？

ABC　iNT　int　π　αβγ　sCity　nMax　a.b　2abc

swip$　[]123　_123　a[]b　a_b　fun()　XYZ　maina_b

题 1.3.7 执行下面的语句

```
char c = d;
int a = 5, b = 3;
float x = 2.6, y = 3.7;
```

后，下列各表达式的值是什么？

(1) (++a)+(++a)+(++a)　　(2) ++a+a*b+--b

(3) (++a)+(++a)*(++a)　　(4) a+++c/20*++b

(5) x*a++ --y/2+a%b　　(6) c−a/a+y/int(x)

(7) b.a*x+c.a*int(x++ −y)　　(8) a.b*x+y+b.a

(9) (c += a+b+int(x)%int(y))　　(10) (c+=a+=b--y*x)+c

题 1.3.8　对于语句

int a = 2, b = 2, c = 2;

判断下列表达式是否正确，并计算正确的表达式的值。

(1) a += b += c++　　(2) a -= b -- c

(3) a -= ++b -++c　　(4) a += b +++c

(5) a += b ++-+c　　(6) a += b +++++c

(7) a -= +b +++c　　(8) a -= -b +++c

(9) a -= b += 3 + (c += 5)　　(10) a += (a + b)·(b - c)

题 1.3.9　编写一个 C++ 程序，使其输出如下两行字符：

```
Programming is fun
Hello,"C++ !"
```

题 1.3.10　写一个 C++ 程序，输入 3 个数，输出绝对值最大的数。

提示：

(1) 在 math.h 头文件中，计算绝对值函数原型为

int abs(int x)　或　double fabs(double x)

(2) 在 minmax.h 头文件中，求两个数最大值的宏定义为

#define max(a,b)　(((a) > (b)) ? (a) : (b))

题 1.3.11　编写函数，把华氏温度转换为摄氏温度，公式为 $C=(F-32)\times 5/9$；在 main 函数中提示用户输入一个华氏温度，转换后输出相应的摄氏温度。

题 1.3.12　编写一个计算年利率为 i，现存入 P 元钱 N 年后所得到钱数 F 的函数，计算公式为 $F = P(1+i)^N$；在 main 函数中提示用户输入年利率、存入的钱数和年数，计算后输出存入的钱数和利息。

提示：在 math.h 头文件中，计算 x^y 函数原型为

double pow(double x, double y)

题 1.3.13　基于习题 1.12，试定义一个计算年利率为 i，存入 P 元钱 N 年后所得到利息的类，然后通过 main 函数测试这个类。

题 1.3.14　修改下面这个程序中的错误(注：10 个错误)，改正后指出运行结果。

```
#include <iostream.h>
void main()
{
  int i ( 10 ) ,j ,k ;
  j ( 20 ) ;            /* 给 j 赋值 */
  k = i * 2 - - j ;
  cout<<' k+ = 1:'<<k + = 1<<end ;
  return 0 ;
}
```

题 1.3.15 阅读下面的程序，指出运行结果，并分析该程序所实现的功能。

```
#include <iostream.h>
main()
{
   int i(2), y(3);
   x = x - y;
   y = x + y;
   x = y - x;
   cout << "x= " << x << ",y= " << endl;
}
```

题 1.3.16 阅读下面的程序，并指出运行结果。

```
#include <iostream.h>
main()
{
   int x(0);
   x++;
   cout << x++ <<endl<< x++ << endl << x++ << endl;
   cout << x << endl;
}
```

题 1.3.17 阅读下面的程序，并指出运行结果。

```
#include <iostream.h>
main()
{
   int a=321;
   char c=a;
   int b=c;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   cout << "c = " << c << endl;
   return 0;
}
```

题 1.3.18 上机运行下面的程序，根据程序的运行结果指出标识符 dec、oct 和 hex 所起的作用。

```
#include <iostream.h>
int main()
{
   unsigned int i = 1000, j = 512, k = 4096;
```

```
    cout << "i =" << dec << i << endl;
    cout << "j =" << oct << j << endl;
    cout << "k =" << hex << k << endl;
    return 0;
}
```

题 1.3.19 指出并改正下列类声明中的错误。

```
class Rectangle{
    int length=10, width=20 ;
public
    int Rectangle(int len, int wid) ;
    int area() ;
}

int Rectangle(int len , int wid)
{
    length=len ;
    width=wid ;
}

int area()
{
    return length * width ;
}
```

题 1.3.20 指出下列程序中的错误。

```
#include <iostream.h>
class Date{
    int day,month,year;
    void setDay(int d);
    void setMonth(int m);
    void setYear(int y);
    void printDate();
};

void setDay(int d)
{
    day=d;
}

void setMonth(int m)
```

```
{
    month=m;
}
void setYear(int y)
{
    year=y;
}
void printDate()
{
    cout<<"\nDate:"<< day << "/ "<< month << "/ "<< year;
}
int main()
{
    Date aDay;
    aDay setDay(1);
    aDay setMonth(10);
    aDay setYear(1949);
    aDay printDate();
}
```

1.3.2 参考答案

题 1.3.1 参考答案：

```
class School{
    char teacher[20];                                   //教师姓名属性
    char student[20];                                   //学生姓名属性
    char lesson[40];                                    //课程名称属性
    unsigned classroom;                                 //教室编号属性
                ⋮
public:
    School();                                           //初始化
    ~School();                                          //删除动态表
    void insert_data();                                 //向管理信息库文件中插入纪录
    void modify_data();                                 //修改信息库文件中的数据
    void delete_record();                               //删除信息库文件中的纪录
    void query_teacher(char * teacher_name);            //查询某教师行为
    void query_student(char * student_name);            //查询某学生行为
    void query_lesson(char * lesson_name);              //查询某课程行为
    void query_teacher_lesson(char * teacher_name);
                                                        //查询某教师所讲授课程的行为
    void query_student_lesson(char * student_name);     //查询某学生所选课程的行为
    void query_lesson_classroom(char * lesson_name);    //查询某课程所在教室
```

```
    void print_list();          //打印各类属性表
                ⋮
};
```

题 1.3.2 参考答案：

```
#include "iostream.h"
class Time{
    int hours,minutes,seconds;
public:
    Time(int,int,int);                          //构造函数进行对时间的设置
    void display(void);
};
Time::Time(int h,int m,int s)
{
    hours=h;
    minutes=m;
    seconds=s;
}
void Time::display()
{
    cout<<hours<<":"<<minutes<<":"<<seconds<<endl;
}
void main()
{
    Time time1(15,30,0),time2(10,10,30);
    time1.display();
    time2.display();
}
```

题 1.3.3 参考答案：

```
class MyRectangle{
    int length,wide;
public:
    MyRectangle(int,int);
    int area();
};
MyRectangle::MyRectangle(int l,int w)
{
    length=l;
    wide=w;
}
int MyRectangle::area()
```

```
{
    return length * wide;
}
void main()
{
    MyRectangle rect(30,6);
    cout<<rect.area()<<endl;
}
```

题 1.3.4 参考答案：

```
class Box{
    int length,wide,high;
public:
    Box(int,int,int);
    int volume();
    int area();
};

Box::Box(int l,int w,int h)
{
    length=l;
    wide=w;
    high=h;
}

int Box::volume()
{
    return length * wide * high;
}

int Box::area()
{
    return 2 * (length * wide+length * high+high * wide);
}

void main()
{
    Box box(2,4,6);
    cout<<box.area()<<endl;
    cout<<box.volume()<<endl;
}
```

题 1.3.5 参考答案：

合法常数：2002　.123　-479　25.　03.14　0.618e+8　0x89e3

非法常数：(−368)　+ −256　− −863　0x12.5　0187　0xABQD　ABF　3.68E−(3+5)

说明：

(1) 一个常数是表达式，反之不成立。如常数−368 是表达式，(−368)和− −256 是表达式但它们不是常数(提示：用 cin 验证)。

(2) 由于八进制数用 0 作为开始标记，所以十进制整数除零之外不能以零开头，但实型数除外。如 0187 是非法的常数，但 03.14 则是合法的常数，它等价于 3.14。

题 1.3.6 参考答案：

合法变量名：ABC　iNT　sCity　nMax　_123　a_b　maina_b

非法变量名：int　π　αβγ　a.b　swip$　−123　a−b　fun()　XYZ

说明：

(1) 合法的变量名是以字母(大小写不限)和下划线"_"打头的字母、下划线和数字组成的字符串。如 ABC 和_123 等是合法的变量名，但 π 和 swip$ 等则是非法的变量名，它们均含有构成变量名的非法字符。

(2) 保留字在结构上符合变量名定义，但它们有特定的语义，C++ 规定不能用作变量名。如 int 是保留字，则它是非法的变量名。

(3) C++ 中的大小写字母是有区别的。如 xyz 和 XYZ 是不同的两个变量名。int 是保留字，而 iNT 则是变量名。

题 1.3.7 参考答案：

(1) 22　(2) 26　(3) 72　(4) 25　(5) 13.15

(6) 82.85　(7) 120　(8) 6.3　(9) c　(10) 198

题 1.3.8 参考答案：

(1) a += b ++ c ++ 是错误的表达式，它等价于 a += (b ++)(c ++)，即两个子表达式(b ++)和(c ++)之间缺少运算符。

(2) a −= b −− − c 是错误的表达式，它等价于 a −=(b −−)c，即子表达式(b −−)和变量 c 之间缺少运算符。

(3) a += ++ b +++ c 是错误的表达式，它等价于 a +=(++(b ++))+ c，使++ 的运算对象是表达式(b ++)而不是变量(左值)。

(4) a += b +++ c 是正确的表达式，它等价于 a +=(b ++)+ c，表达式的值是 6。

(5) a += b ++++ c 是错误的表达式，它等价于 a +=((b ++)++)c，使++ 的运算对象是表达式而不是变量，且在错误的子表达式 b ++++ 和变量 c 之间还缺少运算符。

(6) a += b +++++ c 是错误的表达式，它等价于 a +=((b ++)++)+ c，使 ++ 的运算对象是表达式而不是变量。

(7) a −= −b +++ c 是正确的表达式，它等价于 a −= −(b ++)+ c，表达式的值是−2。

(8) a −= −b +−+ c 是正确的表达式，它等价于 a −= −(b ++)+ c，表达式的值是 2。

(9) a + - b += 3 + (c + - 5) 是正确的表达式，它等价于 a += (b += (3 + (c += 5))),表达式的值是 14。

(10) a += (a + b) / (b - c)表达式语法没有错误但运行时出现语义错误，由于 a = b = 2 使表达式等价于 a += (a + b) / 0，即除数是 0，产生溢出。

题 1.3.9 参考答案：

```
#include "iostream.h"
void main()
{
    cout<< "Programming is fun" << endl ;
    cout << "Hello,\" C++ ! \"\n" ;
}
```

说明：

(1) 由于双撇号"的直接使用，在 C++ 中用来作为字符串的起始和结束标记，所以，如果字符串本身含有双撇号，字符串中的双撇号应该使用转义字符序列\"或\042，避开直接使用双撇号而引发的语法错误。

(2) endl 和转义字符序列"\n"二者的作用相同，但要注意它们各自使用的语法位置的异同。相同之处是"\n"和 endl 均可独立地使用；而不同之处是"\n"可以在字符串中引用，但 endl 不允许。

题 1.3.10 参考答案：

```
#include "iostream.h"
#include "math.h"
#include "minmax.h"
void main()
{
   int x ,y ,z ,absmax ;
   cin >> x >> y >> z ;
   absmax = max( abs( x ) ,abs( y ) );
   absmax = max( abs( absmax ) ,abs( z ) );
   cout << "The absolute value of maximal number is: ";
   cout << absmax << endl;
}
```

题 1.3.11 参考答案：

```
#include "iostream.h"
double FtoC(double f)
{
   return ((f-32) * 5/9);
}
```

```
void main()
{
    double c,f;
        cout<<"Input Fahrenheit:";
        cin>>f;
        c=FtoC(f);
        cout<<"Centigrad:"<<c<<endl;
}
```

题 1.3.12 参考答案：

```
#include "iostream.h"
#include "math.h"
#define I 0.1
double rate(double p,double n)
{
    return p*pow(1+I,n);
}
void main()
{
    int p,n;
    cout<<"Input P,N:";
    cin>>p>>n;
    cout<<"F="<<rate(p,n)<<endl;
}
```

题 1.3.13 参考答案：

```
#include "iostream.h"
#include "math.h"

class Interest{
    private:
        int n;
        double p,i;
    public:
        input();
        output();
};
Interest::input()
{
        cout<<"Interest rate is:";
        cin>>i;
        cout<<"Money number is:";
```

```
        cin≫p;
        cout≪"Year number is:";
        cin≫n;
}

Interest::output()
{
        cout≪"The interest is:"≪p * pow(1+i,n)-p≪endl;
}

int main()
{
        Interest a;
        a.input();
        a.output();
        return 0;
}
```

题 1.3.14 参考答案：

(1) 错误的位置(用灰底标记)和序号。

```
#include <iostream.h>
void main()
①
{
    int i(10), j,k ;
    j ( 20 ) ; /* 给 j 赋值 /*
      ②  ③                  ④
    k = i * 2 - - j ;
                ⑤
    cout≪ ' k+=1: ' ≪    k+=1    ≪ end ;
        ⑥          ⑦  ⑧          ⑨   ⑩
    return 0 ;
}
```

(2) 改正后的程序及运行结果。

```
#include <iostream.h>
int main()
{
    int i ( 10 ) ,j ,k ;
    j = 20 ;                    // 给 j 赋值
    k = i * 2 - ( - j ) ;
    cout≪"k+=1:"≪(k+=1 )≪ endl ;
    return 0 ;
}
```

运行结果：

```
k+=1;41
```

题 1.3.15 参考答案：

运行结果：

```
x=3, y=2
```

分析：该程序实现的功能是交换变量 x 和 y 中的值。此程序利用变量在不同的时刻可以存取不同值的特性，用其中一个变量保留两个变量值的差，而另一个变量保持不变，仍然是原来的值。这一功能用语句"x = x - y ;"加以实现。执行这条语句后，变量 x 保留其原值同变量 y 的差－1，y 值保持不变。这时 x 的原值等于 y(原值)加上 x 的现在值。这一功能用语句"y = x + y ;"来实现，同时把 x 的原值赋给变量 y，使 y 得到 x 的原值 2。同理 y 的原值 3 由语句"x = y - x ;"中的表达式"y - x ;"求得，且把它赋给变量 x。这样就实现了两个变量值的交换。

说明：由于上述程序的可读性较差，而且如果变量是实型时可能产生计算误差，建议在实现两个变量值的交换时尽量少用本题程序中采用的计算方法，最好采用引用另一个临时变量的方式来实现。

题 1.3.16 参考答案：

运行结果：

```
3
2
1
4
```

说明：产生上述结果的原因是 C++ 在处理输出项的表达式计算时采用了下堆栈技术。

题 1.3.17 参考答案：

运行结果：

```
a=321
b=65
c=A
```

说明：注意系统给整型变量 a 分配的两个字节存放整型数 321 存储的顺序。

题 1.3.18 参考答案：

运行结果：

```
i=1000
j=1000
k=1000
```

标识符 dec、oct 和 hex 的作用是进行数制转换。dec 代表十进制转换，oct 代表八进制转换，hex 代表十六进制转换。

题 1.3.19 参考答案：

该类的声明中有两类错误，它们是：

(1) 语句"int length=10、width=20 ;"是错误的，因为在类的定义中不能给 private 成员初始化。

(2) 在类结构外声明成员函数时，函数名前必须用类名后接符号"::"作为标识。

改正后的类声明如下：

```
class Rectangle{
    int length , width ;
public
        int Rectangle(int len ,int wid) ;
    int area() ;
}

int Rectangle::Rectangle(int len , int wid)
{
        length=len ;
        width=wid ;
}
int Rectangle::area()
{
        return length * width ;
}
```

题 1.3.20 参考答案：

该程序中出现的错误是类结构外定义的成员函数名前没有使用"类名"后接符号"::"作为标识，修改后的代码如下：

```
#include <iostream.h>
class Date{
    int day,month,year;
    void setDay(int d);
    void setMonth(int m);
    void setYear(int y);
    void printDate();
};

void Date::setDay(int d)
{
    day=d;
}
```

```
void Date::setMonth(int m)
{
    month= m;
}
void Date::setYear(int y)
{
    year=y;
}
void Date::printDate()
{
    cout<<"\nDate:"<< day << "/ "<< month << "/ "<< year;
}
int main()
{
    Date aDay;
    aDay setDay(1);
    aDay setMonth(10);
    aDay setYear(1949);
    aDay printDate();
}
```

第2章　算法设计与程序结构

2.1　编程知识

2.1.1　关系运算与选择结构

1. 关系运算符与关系表达式

有比较才有鉴别。关系运算即比较运算，C++提供如下6个关系运算符：

<(小于)　　<=(小于等于)　　>(大于)　　>=(大于等于)

==(等于)　　!=(不等于)

关系运算符的特点：

(1) 用标量"非0"与"0"来代表"真"与"假"；

(2) 都是双目运算符，并且都是从左向右结合的；

(3) 关系运算要求两个同类型的操作数。

2. 逻辑运算符与逻辑表达式

C++提供的3个逻辑运算符为

&&(逻辑"与")　　||(逻辑"或")　　!(逻辑"非")

3. if … else 选择结构

if … else 结构的基本形式为

```
if(判断表达式)
        语句1
else
        语句2
if(判断表达式)
        语句
```

4. 条件运算符与条件表达式

这种表达式称为条件表达式。条件表达式的一般形式为

```
e1 ? e2 : e3
```

其含义为：若e1为"真"(非0)，则条件表达式取e2的值；否则取e3的值。表达式中的"?:"称为条件运算符，它是三目运算符，并且是从右向左结合的。它的优先级比赋值运算符

略高。

5. else if 结构

if … else 结构组合成的多分支结构：

```
if(判断表达式 1)
    语句 1
else if(表达式 2)
    语句 2
…
else if(表达式 n)
    语句 n
else
    语句 n+1
```

6. switch 结构

switch 结构也叫开关结构，其结构形式如下：

```
switch(开关表达式)
{
    case 常量 1:
        语句表列
    case 常量 2:
        语句表列
        …
    default:
        语句表列
}
```

它的控制机理为：当程序的流程到达 switch 结构时，首先对开关表达式进行计算，用其值从上到下在所有的 case 常量中寻找匹配（相等）者；流程开关从找到的第一个匹配的 case 后的第一个语句开始接通，即以此 case 作为 switch 结构的入口；当所有的 case 常量都不匹配时，便以 default 为入口；当没有 default 且所有的 case 常量都不匹配时，流程不进入该 switch 结构。

注意：

（1）switch 结构的测试表达式只能是整型表达式（包括数字表达式、字符型表达式、条件表达式、逻辑表达式的值，因为这些表达式的值也可以看作是特殊的整型表达式）；

（2）各 case 常量与开关表达式的类型要相容；

（3）在同一个 switch 结构中，不能有重复的 case 常量，否则将出现矛盾的结果；

(4) switch 结构以匹配的 case 作为入口，为了真正地进行分情况处理，当执行完一个 case 语句后，为使之不再执行后面的 case 语句内容，应使用 break 语句，跳出 switch 结构；

(5) switch 要求一对花括弧。

2.1.2 循环结构

1. for 循环结构

for 结构的基本形式为

```
for(表达式 1; 表达式 2; 表达式 3)
        循环体
```

说明：

(1) 表达式 1 为初始化表达式，表达式 2 为判断表达式，表达式 3 为修正表达式。

(2) 流程如图 2.1 所示。

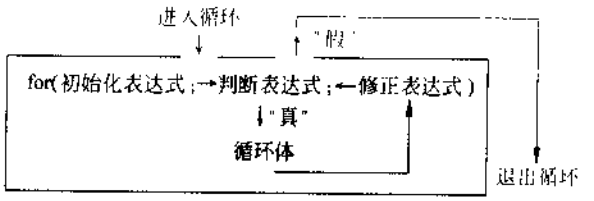

图 2.1 for 结构的流程

(3) C++ 对 for 结构中的 3 个表达式无太严格的要求，在圆括号中，任何一个表达式都可以默认，但所需的分号不可缺少。

2. while 循环结构

while 结构的一般形式如下：

```
while(判断表达式)
      循环体语句
```

说明：在进入循环之前，先测试一次判断表达式，如果判断表达式为真(值不为 0)，则进入循环，开始执行循环体语句；以后每执行完一次循环体语句，就再测试一次判断表达式，如果判断表达式还为"真"(值不为 0)，就继续执行循环体语句；如此重复，直到某一次循环体语句执行完，判断表达式变为"假"止。

3. do … while 循环结构

do … while 结构与 while 结构非常相似，一点微小的区别仅在于 while 结构是先判断后执行循环体，而 do … while 结构是先执行一次循环体再判断。也就是说，do … while 结构最少要执行循环体一次，而 while 结构有可能一次也不执行循环体。因此，同样的问题，

使用 while 结构和使用 do … while 结构，在设置循环条件时应有所差别。其形式为

```
do
    循环体语句
while(判断表达式);
```

应当注意，do … while 结构的最后要求一个分号。因此，从整体上来说 do … while 相当于一条语句。

4. 基于循环控制结构的跳转语句——continue 和 break

图 2.2 为循环结构中使用 continue 语句、break 语句与一般情况的执行流程的比较。

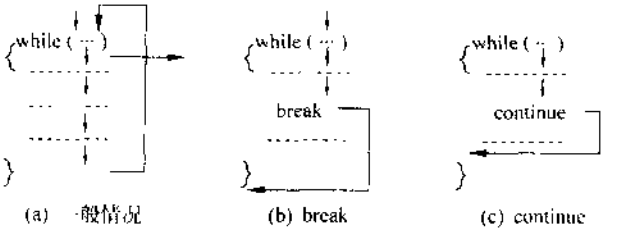

图 2.2 循环结构中的 break 和 continue

2.1.3 变量的存储属性

变量的作用域和生存期，统称变量的存储属性。

1. 变量的作用域

(1) 局部变量或内部变量：只能在类、函数或语句块中起作用的变量。

(2) 全局变量：可以在一个文件范围内或一个程序的所有文件范围内起作用的变量。

2. 变量的生存期

(1) 永久生存期——静态变量：存储在静态存储区中的变量，在编译时即被创建，在程序运行结束时才被撤销，相对于程序的生命而言，其生命是永久的。

(2) 动态生存期——动态变量：存储在动态存储区中的变量，在程序运行过程中被动态创建，在程序运行完所在域即被撤销。相对于程序而言，其生命是动态的。

3. C++ 的自动变量

自动变量被声明在"内部"(即类、函数、块、声明语句之中)，具有如下特征：

(1) 作用域是局部的——局部于定义语句所在的类、函数、块、声明语句之中；

(2) 生存期是动态的。

4. C++ 的外部变量

(1) 外部变量的特征

① 永久生存期 —与程序共存亡；

② 全程作用域；

③ 只能被初始化一次。

(2) 外部变量的声明

外部变量有两种声明形式：

① 定义性声明：为了建立实体；

② 引用性声明：为了建立标识符与实体之间的联系。

变量的定义性声明的形式为

extern 类型 变量名 = 初始化表达式；

通常，只要是定义在外部的，就认为是外部变量，可以把存储类声明符 extern 默认。

定义性声明的作用是指示编译器为变量分配内存，引用性声明的作用是告诉编译器标识符实体的特征。引用性声明与定义性声明在应用上的区别在于：

(1) 定义性声明一定是在外部，而引用性声明不限于在外部，只要在引用之前给出便可。主要用于引用出现在定义之前的情况。

(2) 定义性声明具有初始化的作用，而引用性声明没有。

(3) 定义性声明有两种形式：带有 extern、初始化表达式以及定义在外部并默认 extern。而引用性声明只有一种形式，带有 extern 但不带有初始化表达式。

(4) 定义性声明只能有一次，而引用性声明可以有多次。

5. 静态变量

(1) 静态变量的特征

① 生存期是永久的；

② 作用域可以是全程的，也可以是局部的——与声明语句的位置有关。

(2) 静态变量的声明

关键字：static。

(3) 静态成员

静态成员的特点：

① 在类定义之外定义静态成员函数时，不使用 static；

② 静态成员函数也可以定义成内嵌的；

③ 类中的任何成员函数都可以访问类对象的静态成员；

④ 由于静态成员函数不与特定的对象相联系，调用时应使用：

类名 :: 静态成员函数名()

⑤ 静态成员函数只能通过对象名(或指向对象的指针)访问该对象的非静态成员。

6. const 对象

一个 const 对象，只允许被声明为 const 的方法函数引用。要注意修饰符 const 的位置，如果写在最前面就不对了。

以大对象作为参数时，为避免对象拷贝所引起的开销，多将参数对象声明为引用类型。这时为避免对对象参数的偶然修改，如大多数字符串函数一样，把对象参数声明为 const 对象。对这个对象的操作只能使用 const 指定的成员函数。

2.1.4 多文件程序结构与程序项目

1. 多文件程序结构

一个程序规模很小时，可以把源代码编写在一个源文件中。而一个程序的规模较大时，为了进行编写和编译、调试的便利，常常要把它分成多个源文件。每一个源文件就成为该程序的一个编译单元——编译器总是以文件为单位工作的。在进行程序源代码的分割时，要注意几点：

(1) 在一个程序的多个源文件中，只能有一个文件具有主函数 main()；

(2) 每一个函数都应当完整地存放在一个文件中；

(3) 一个类内联成员函数，不可与其类定义(类接口)分开；

(4) 为了便于链接，要将同一个源程序的多个源文件定义成一个项目(project)文件。

2. 程序项目

在集成化的开发环境中，把一组相互关联的源文件、资源文件以及支撑这文件的类的集合称为一个工程项目(Project，后面简称为项目)。

项目是应用程序开发的基本单位。项目管理把组成一个项目的多个文件作为一个整体来处理。这个整体称为项目文件。应当注意：

(1) 项目文件含有环境信息。当环境(如盘号、路径、设置的参数等)改变时，应重新建立项目文件，否则可能出现莫名其妙的错误。

(2) 一个项目文件要涉及许多头文件，因此要注意它们之间的一致性。

(3) 一个项目文件必须且只能有一个主函数。

关于项目管理器的其他功能和用法请参阅有关手册。

3. 多文件程序中变量的连接属性

(1) 外部变量的连接属性

当一个程序的不同源文件中要使用某一个公共变量时，应当使用外部变量。

(2) 全程静态变量

全程静态变量的作用域仅全局于其所在的文件。使用局部静态变量的好处是各编译单元分别设计时，不需要考虑别的编译单元中是否使用了相同的全局变量名。

2.1.5 文件包含与条件编译

1. 文件包含

使用#include指令进行文件包含，其实质就是在预处理时将一个源文件嵌入到当前文件中的指令所在处。文件包含命令有两种格式。

格式1——标准方式：

```
#include <文件名>
```

C++预处理器遇到这条命令后，就到include子目录下搜索给出的文件，并把它嵌入到当前文件中。这种搜寻方式称为标准方式。这种方式适合于搜寻系统建立的文件。

格式2：

```
#include "文件名"
```

预处理器遇到这种格式的文件包含命令后，首先在当前文件所在目录中进行搜寻；如果搜寻不到，就再按标准方式进行搜寻。这种方式适合于用户自己建立的头文件。

用户显式地保证程序一致性的基本方法是保持声明的一致性。保持声明一致性的简单办法是提供一个头文件，让头文件中的信息作为各模块之间的接口信息，有利于提供可重用的模块。

2. 条件编译

条件编译是源文件内系统级的选择结构，其格式有：

```
#if 常量表达式 1
    语句或预处理指令表列 1
#elif 常量表达式 2
    语句或预处理指令表列 2
        ⋮
#elif 常量表达式 n
    语句或预处理指令表列 n
#else
    语句或预处理指令表列 n+1
#endif
```

或

```
#ifdef 标识符
    语句或预处理指令表列 1
#else
```

```
    语句或预处理指令表列 2
#endif
```

或

```
#ifndef 标识符
    语句或预处理指令表列 1
#else
    语句或预处理指令表列 2
#endif
```

在常量表达式中可以使用 C++ 的算术、逻辑或关系运算符，以及在它之前定义的宏名和常量，但不能使用变量，因为编译预处理时程序还没有开始执行。

条件编译的主要用途：

(1) 控制编译器有选择地对源程序中的某部分进行编译或不进行编译；

(2) 用于程序的跟踪调试。

2.2 例题示范

2.2.1 控制结构基础

例 2.2.1 从键盘输入 3 个数，找出最大的一个数。

解：

1. 类界面设计

```
class ThreeNumber{
        int m1,m2,m3;
    public:
        void Input3Integers();
        void GetMax();
};
```

2. 测试程序设计

```
int main()
{
    ThreeNumber a;
    a.Input3Integers();
    a.GetMax();
    return 0;
}
```

3. 成员函数的实现

```
include <iostream.h>

void ThreeNumber::Input3Integers()
{
        cout << "Input 3 Integers:\n";
        cin >> m1 >> m2 >> m3;
}

int ThreeNumber::GetMax()
{
        int max;

        if (m1 > m2)
                if (m1 > m3)
                    max = m1;
                else
                    max = m3;
        else
                if (m2 > m3)
                    max = m2;
                else
                    max = m3;
        cout << "The Max integer is:"<< max << endl;
}
```

例 2.2.2 从键盘上输入一个字符，判断它是控制字符，还是数字、大写字母、小写字母，还是其他字符。

解：

1. 类界面设计

```
class Character
{
        char mCh;
    public:
        void GetCharacter();
        void GetClass();
};
```

2. 测试程序设计

```
int main()
{
        Character c;
        c.GetCharacter();
        c.GetClass();
        return 0;
}
```

3. 成员函数 GetCharecter()的实现

```
# include <iostream.h>
void Character::GetCharacter()
{
      cout << "Enter a character:\n";
      cin >> mCh;
}
```

成员函数 GetClass()的实现较为复杂，为了得到它的实现，先进行下面的分析：

判断一个字符是控制字符，还是数字、大写字母、小写字母，还是其他字符，可以用下面的算法描述：

```
if(mCh 是控制字符)
        打印"是控制字符"
else if(mCh 是数字)
        打印"是数字"
else if(mCh 是大写字母)
        打印"是大写字母"
else if(mCh 是小写字母)
        打印"是小写字母"
else
        打印"是其他字符"
```

进一步描述上述各判断表达式。根据 ASCII 码表，可以看出：

① c 是控制字符：c < 31

② c 是数字：c >= '0' && c <= '9'

③ c 是大写字母：c >= 'A' && c <= 'Z'

④ c 是小写字母：c >= 'a' && c <= 'z'

由上面的分析可以得到下面的成员函数 GetClass()的实现：

(1) 用嵌套的 if-else 结构实现

```
void Character::GetClass()
{
    if(mCh < 31)
        cout << "\nIt's a control character.";
    else if(mCh >= '0' && c <= '9')
        cout << "\nIt's a digit character.";
    else if(mCh >= 'A' && c <= 'Z')
        cout << "\nIt's a capital letter.";
    else if(mCh >= 'a' && c <= 'z')
        cout << "\nIt's a lower letter.";
    else
        cout << "\nIt's an other character.";
}
```

(2) 用 switch 结构实现

```
void Character::GetClass()
{
        switch(mCh)
        {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                cout << "\nIt's a digit character.";
                break;
            case '':
            case '\n':
            case '\t':
                cout << "\nIt's a white character.";
                break;
            default:
                cout << "\nIt's a letter character.";
                break;
        }
}
```

例 2.2.3 高斯(Gauss)问题,即计算 $\sum i$。

解：

1. 类界面设计

```
class Gauss
{
        int mi;
    public:
        InputInteger();
        GetGaussSum();
};
```

2. 测试程序设计

```
int main()
{
        Gauss g1;
        g1.InputInteger();
        g1.GetGaussSum();
        return 0;
}

# include <iostream.h>

void Gauss::InputInteger()
{
        cout << "Enter a character:\n";
        cin >> mi;
}

void Gauss::GetGaussSum()
{
        int i, isum = 0;
        for(i = 1; i <= mi; i ++)
                isum += i;
        cout << "\nThe sum is:" << isum;
}
```

2.2.2 穷举

例 2.2.4 大奖赛评分程序。在唱歌等奖赛评分时，一般要有若干名评委，记分规则为：去掉一个最高分和一个最低分，再算平均分。设按百分制记分，试设计一个算分程序。

解：

1. 类界面设计

```
class Score
{
        int mNumber_Adjudicator;          // 评委人数
  public:
            Score(int iNum_Adj)
             {
                 mNumber_Adjudicator = iNum_Adj;
             }
        void GetResult();                 // 计算并给出结果
};
```

2. 测试程序设计

```
int main()
{
        Score m(15);                      // 假定有 15 个评委
        m.GetResult();
        return 0;
}
```

3. 成员函数 GetResult()的设计

计算结果的关键是要用穷举的成员函数找出最高分和最低分，然后将剩余分数平均。在程序中筛选最高分的办法是：由于评委打分的最小值不会小于 0，所以以 0 为最高分 max 的初值；以后，每输入一个评委打分，都与 max 比较一次，若比 max 大，就以其为新的 max 值，直到全比较完为止。这样才能找出真正的最高分。如果以最大整型数为 max 的初值，则由于没有数比它大，最后找出的 max 是最大整型数。筛选最低分的办法与之相仿。

```
# include <iostream.h>

void Score::GetResult()
{
      int          iScore,                    // 每个评委的给分
                   i,                         // 临时变量
                   max = 0,                   // 最高分
                   min = 100;                 // 最低分
        float      fSum = 0;                  // 均分
        for(i = 1; i <= mNumber_Adjudicator; i ++)
        {
            cout << "\n 输入第" << i << "个分";
            cin >> iScore;                    // 输入评委评分
```

```
            fSum += iScore;                          // 计算总分
            if(score>max) max = score;               // 筛选最高分
            if(score<min) min = score;               // 筛选最低分
        }
        cout << "\n 均分为" << (fSum-max-min)/(mNumber_Adjudicator-2);
}
```

例 2.2.5 打印 3 ～ N 中的素数。

解：

1. 类界面设计

```
class Primer
{
        int       mNumber;
    public:
                  Primer(int n)
                  {
                      mNumber = n;
                  }
        void      GetPrimers();
};
```

2. 测试程序设计

```
int main()
{
      Primer      n(100);            //设 N=100
      n.GetPrimers();
      return 0;
}
```

3. 设计成员函数 GetPrimers()

素数是除 1 和它本身之外不能被其他任何整数整除的正整数。找素数数列的一种算法称为筛法，它是一种穷举法。对本例来说可以写为：

```
for(int m = 3; m <= mNumber; m ++)  // 穷举测试 3～number 中的各数是否素数
{
      测试一个具体的 m 是否素数；
      不是，则取下一个数；是，则打印，取下一个数；
}
```

测试一个数 number 是否素数的基本思想是：用 2 到 number/2 之间的数依次去除

number，只要有一个数能将 number 整除，number 就不是素数。只有 2 到 number 之间的任何数都不能将 number 整除时，number 才是素数。

```
# include <iostream. h>

Void Primer::GetPrimers()
{
        cout << "\nThe primers from 3 to " << mNumber << "is:\n";
        for(int m = 3; m <= mNumber; m ++)   {
                int flag = 1;                    // 设置标志
                for(int n = 2; n <= m / 2; n ++)  {
                        if(m % n == 0)  {
                                flag = 0;  // 修正标志
                                break;
                        }
                }
                if(flag == 0)                    // 判断标志
                        continue;
                cout << m;
        }
}
```

例 2.2.6 36 块砖，36 人搬，男搬 4、女搬 3、两个小儿抬一砖，要求一次全搬完，问男、女、小儿各若干？

解：

1. 类界面设计

```
class RemoveBricks
{
        int iNumberOfPeople;
        int iNumberOfBrick;
  public;
                RemoveBrick(int iNumPeop, int iNumBrc) // 构造函数
                {
                        iNumberOfPeople = iNumPeop ;
                        iNumberOfBrick = iNumBrc ;
                }
        void Calculate();
};
```

2. 测试程序设计

```
int main()
```

```
{
    BrickProblem    rb(36,36);  // 用构造函数创建对象并初始化
    rb.Calculate();             // 调用对象的成员函数进行计算
    return 0;
}
```

3. 成员函数设计

这里，构造函数 RemoveBrick()与类名一致，作用是创建对象。它的参数用来初始化数据成员。因为，不允许在类定义中直接初始化数据成员。当主函数中，用类名声明一个对象(如 rb)时，就会自动调用它，并用声明语句中提供的实参来初始化对象的数据成员。如果程序员不定义这种构造函数，则在创建对象时，系统会调用一个默认的构造函数。这个默认的构造函数是无参数的，不能用来进行数据成员的初始化。

下面考虑成员函数 Calculate()的实现。

设男人、女人、小孩数各为 men、women、children，则可得如下模型：

```
4 * men + 3 * women + children / 2 = 36
men + women + children = 36
```

这是一个不定方程，求解还须增加其他约束条件。按常识，men、women 和 children 都应为正整数，且它们的取值范围分别应为：

```
men:0 ~ 8
women:0 ~ 11
children:0 ~ 36
```

解题就是在上述 3 个数的范围之内找满足上述两个方程的 men、women 和 children 的组合。

求解本题的一个自然想法，是依次对 men、women、children 取值范围内的各数一一进行试探，找满足前面两方程的组合：首先从 0 开始，列举 men 的各个可能值，在每个 men 值下找满足两个方程的一组解，算法如下：

```
men = 0;
while(men ++ < 8)
{
    s1:找满足两个方程的解的 women,children
    s2:输出一组解
}
```

在一个集合内对每个元素一一测试方法称为穷举法。穷举法对人来说常常是单调而又繁琐的工作，但对计算机来说，重复计算正好可以用简洁的程序发挥它运算速度高的优势。

下面进一步用穷举法来表现 s1：

```
women = 0;
while(women ++ < 11)
{
        s1.1 找满足方程的一个 children
        s1.2 输出一组解
}
```

由于对列举的每个 men 与每个 women 都可以按下式：

```
children = 36 - men - women
```

求出一个 children。因此，只要该 children 满足方程

```
4 * men + 3 * women + children / 2 = 36
```

便可以得到一组满足题意的 wen、women、children。故 s1.1 与 s1.2 可以改写为：

```
children = 36 - men - women;
if(4 * men + 3 * women + children / 2 == 36)
        输出一组解
```

经过剥葱头似的几步求精过程，已把模型基本上表现为 C++ 程序了。加入类型声明语句并调整输出格式，并考虑更一般的情形，便可以得到函数 Calculate()。

```
# include <iostream.h>

void BrickProblem::Calculate()
{
        cout << "men ***** women *** children\n");
        int men = 0;                                    // 赋初值
        while(men ++ < iNumberOfPeople/4) {             // 修订并判断
                int women = 0;                          // 赋初值
                while(women ++ < iNumberOfPeople/3) {   // 修订并判断
                        int children;
                        children = iNumberOfPeople - men - women;
                        if(4 * men + 3 * women + children / 2 == iNumberOfBrick)
                        {
                                cout.width (6);         // 设置输出域宽
                                cout << men << women << children;
                        }
                }
        }
}
```

4. 程序测试结果

```
men *** women * children
1       6       29                    (这是一组错误解)
3       3       30
```

5. 结果分析与程序改进

(1) 分析输出结果的两组解显见，第一组解是不正确的。因为按照题意，小儿数只能是偶数。问题出在什么地方呢？首先可以肯定用上述逐步求精的成员函数导出的程序的逻辑是正确的。经过仔细检查发现问题在声明语句中。我们声明 men、women、children 都是整型，而在条件语句的判断表达式中有运算：

```
children / 2
```

两个整型数相运算的结果仍是整型数。这样当 children 为奇数时，就会出现由于运算的误差而引出的错误结论。为避免这种误差而引起的计算错误，可采用两种办法：

① 将 children 声明为实型；

② 将表达式 children / 2 改为 children / 2.0。

以后凡遇到整型数相除都应注意这一点。

(2) 语句

```
cout. width (6);
```

用于设置数据输出的域宽，即后面每个字段占用的字符个数。本例中，每个数据占用 6 个字符宽度。由这个语句可以看出，"cout"是一个对象名，它(以及 cin)实际上是类 iostream 的对象。函数 width()是一个成员函数。由于 iostream 类定义在头文件 iostream. h 中，所以要使用对象 cout(以及 cin)时，应当用文件包含语句将 iostream. h 包含在程序中。

2.2.3 递推

例 2.2.7 求两个非负整数 m、n 的最大公因子。

解：

1. 类界面设计

```
class Gcd
{
        int       mM,nN;
    public:
        void      GetTwoPositiveInt()
        {
              cout << "\nEnter 2 positive integers:";
```

```
            cin ≫ mM ≫ mN;
        }
        Void    GetGcd();
};
```

2. 测试程序设计

```
int main()
{
        Gcd         gcd1;
        Gcd1.GetTwoPositiveInt();
        Gcd1.GetGcd();
        Return 0;
}
```

3. 成员函数 GetGcd()的设计

求两个非负整数 m、n 的最大公因子，可用辗转相除法。如果用 u 表示被除数，v 表示除数，r 表示余数，辗转相除算法可以描述为：

```
u = m;
v = n;
当 r = u % v 不为 0 时，
        u = v;
        v = r;
输出 v;
```

如 $m=36$，$n=21$，计算过程为：

```
u: 36    21    15    6
     ↙    ↙    ↙
v: 21    15    6     3 → 最大公因子
     ↙    ↙    ↙
r: 15    6     3     0
```

上述的算法模型，用 C++ 程序表现非常简单。

```
# include <iostream.h>

Void        Gcd::GetGcd()
{
        int u, v, r;
```

```
        u = mM;
        v = mN;

        if(u * v != 0){
                while(r = u % v) {
                        u = v;
                        v = r;
                }
                cout << "\nThe gcd is:" << v;
        }
        else
                cout << "\nDivision by zero!";
}
```

说明：递推是一个不断用变量的旧值递推其新值的过程。一般说来递推算法要由以下3个环节组成：

(1) 迭代初始值，如本例中的u的初始值为m，v的初始值为n。

(2) 迭代公式，如本例中的

```
u ← v;
v ← r;
r ← u % v;
```

(3) 迭代终止条件，本例中的u % v == 0，是用一个表达式的值来确定迭代是否终止。还有一种是用计数器来确定，使迭代了规定次数后终止。

例 2.2.8 一天一只小猴子摘下一堆桃子，当即吃去一半，还觉得不过瘾，又多吃了一个。第二天接着吃了前一天剩下的一半，馋不忍罢又多吃了一个。以后每天如此。到第十天小猴子去吃时，只剩下一个桃子了。问小猴子共摘了多少桃子。

解：

1. 类界面设计

```
class Peach
{
        const int  mDay = 10;
    public:
        void GetPeachNum();
};
```

2. 测试程序设计

```
int main()
{
```

```
        Peach Peach1;
        Peach1.GetPeachNum();
        Return 0;
}
```

3. 成员函数 GetPeachNum()的设计

首先分析一下用代数方法如何求解此题。设小猴子共摘了 x 个桃子，根据题意可有：

第 1 天吃掉 $x/2+1$ 个，剩下 $x/2-1=(x-2)/2$ 个；

第 2 天吃掉 $((x-2)/2)/2+1$ 个，剩下 $(x-2)/2-(((x-2)/2)/2+1)=(x-2)/2-(x-2+2^2)/2^2=(x-2-2^2)/2^2$ 个；

……

第 i 天剩下：$(x-2-2^2-\cdots-2^i)/2^i$ 个；

……

第 9 天剩下：$(x-2-2^2-\cdots-2^9)/2^9=1$ 个，即第 10 天看到的 1 个桃子。解此方程可得：

$$x=2^9+2^8+2^8+2^7+2^6+2^5+2^4+2^3+2^2+2=1534\ (个)$$

根据此式可以编写出解题程序，但程序将执行一个较复杂的计算。如果题目给的不是到第 10 天才剩一个，而是到第 100 天才剩一个，则有：$x=2^{99}+2^{99}+2^{98}+\cdots+2$，计算更为复杂。若用倒推法(倒退)计算就简单多了。其基本思想是，从最后一天起逐次推出前一天的桃子数，直到推出第 1 天吃桃子前的全部桃子数为止。具体成员函数如下：

根据题意，猴子某天吃桃子前的桃子数是前一天吃桃子之前的桃子数的一半再减 1，即

$$Peach_N=1/2\ Peach_{N-1}-1$$

或

$$Peach_{N-1}=(Peach_N+1)*2$$

这样就可以从第 n 天的桃子数倒推出第 $(n-1)$ 天的桃子数(都是指该天未吃桃子前的数)，并一直可递推求出最初的值。反复的递推过程可以用循环结构实现。

```
# include <iostream.h>

void Peach::GetPeachNum()
{
    int peach = 1, day = mDay;
    while(day > 1){
        peach = (peach + 1) * 2;
```

```
        day --;
    }
    cout << "The number of peach is:" << peach;
}
```

2.2.4 模拟

例 2.2.9 导弹追击飞机。导弹追击飞机时，由于飞机在运动，导弹要不断地调整自己的方向，以保持弹头始终指向飞机。问题是，导弹应以多大的时间间隔进行方向调整并以多大的速度追击较好。这一问题看起来简单，它的数学方程的求解却很困难。用计算机进行模拟常常可以获得较满意的结果。

解：

1. 导弹追击飞机的模型

为简化问题，权且假定飞机只沿 X 轴作水平飞行，导弹也在同一平面内飞行。图 2.3 给出了导弹追击飞机的示意图。图中，当飞机出现在坐标原点(0,0)时，导弹从(Xm_0,Ym_0)处开始追击飞机。

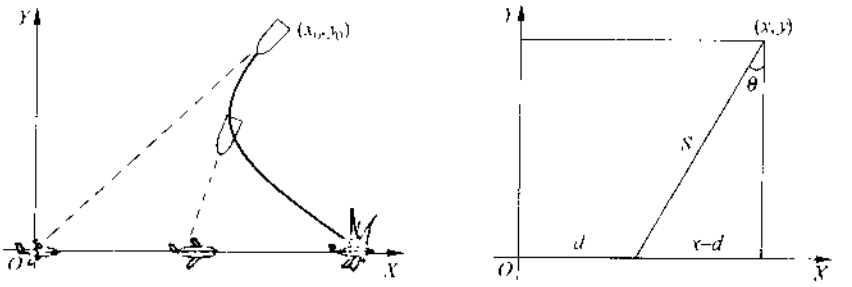

图 2.3 导弹追击飞机

设某一时刻导弹与飞机的坐标分别为：

(Xm,Ym)：某一瞬间导弹的坐标；

(Xa,0)：某一瞬间飞机的坐标。

由于导弹与飞机的飞行轨迹都是时间的函数，因此采用时间步长法模拟较为合理。设

dt：时间步长变量；

va：飞机的速度；

vm：导弹的速度；

s：某一瞬间导弹与飞机间的距离。

由图 2.3 所示导弹与飞机的位置关系，可以得出：

$$s=\sqrt{Ym^2+(Xm-Xa)^2}$$

再设导弹飞行的方向角为 θ，则有：

$\sin\theta = (Xm - Ma)/s$
$\cos\theta = Ym / s$

每经过一个时间步长 dt，导弹与飞机的位置将发生如下变化：

```
Xm = Xm - Vm * dt * (Xm - Xa)/s
Ym = Ym - Vm * dt * Ym/s
Xa = Xa + Va * dt
```

这就是按时间步长法得到的导弹追击飞机的模型。

2. 导弹和飞机的类界面

一般说来，对于导弹类和飞机类都有如下形式：

```
class XXXX
{
        坐标;
        速度;
    public:
        double 计算经过一个时间步长后的坐标(时间步长);
};
```

但是，从上述导弹追击飞机的模型中，导弹的坐标与飞机的坐标有关，即在计算导弹的成员函数中要访问飞机的数据成员。而为了使导弹类的成员函数能访问飞机类的数据成员，就必须将导弹类计算坐标的成员函数声名为飞机类的友元函数。下面是两个类的定义：

```
#include <math.h>                                   // 数学库头文件
class Air_raider;                                   // 须预先声明
class Missile
{
        float x,y;                                  // 某时刻导弹的坐标
        float v;                                    // 导弹速度
    public:
            Missile (float x0, float y0, float Vm)  // 构造函数
            {
                x = x0; y = y0; v = Vm;
            }
        float    nextXY (Air_raider &a);
};


class Air_raider
{
        float    x;                                 // 某时刻飞机的坐标(水平)
```

```
        float     v;                                    // 飞机的速度
    public:
                  Air_raider(float x0, float va)        // 构造函数
                  {
                      x = x0; v = va;
                  }
            void  nextX()
                  {
                      x = x + DT * v;
                  }
            friend float Missile :: nextXY (Air_raider & ); // 友元函数声明
};
```

3. 成员函数设计

```
float     Missile ::nextXY (Air_raider &a)
{
    float s;
    s = sqrt(x * y + (x - a.x) * (x - a.x));
    x = x - v * DT * (x - a.x)/s;
    y = y - v * DT * y/s;
    return s;
}
```

4. 测试程序

```
#include <iostream.h>
#define S0   5                          // 近似地认为已经追上的最大距离
#define DT   5                          // 时间步长

int main()
{
    Missile m(4000,100000,10000);       // 生成一个导弹对象
    Air_raider a(0,500);                // 生成一个飞机对象

    int steps = 0;

    do {
        steps ++;                       // 计算导弹方向调整步数
        a.nextX();
    } while(m.nextXY(a,s)>S0);
```

```
    cout << "\n" << steps << "," << s;
    return 0;
}
```

5. 测试结果

```
2
```

例 2.2.10 中子扩散问题。原子反应堆的壁是铅制的。中子从铅壁的内侧(为简化问题,设以垂直方向)进入,走一定距离(设此距离为铅原子的直径 d),与铅原子碰撞,之后改变方向(这个方向是随机的),又走一定距离(仍为 d),与另一个原子碰撞。如图 2.4 所示,如此经过多次碰撞后,中子可能穿透铅壁辐射到反应堆外,也可能将其能量耗尽被铅壁吸收,还可能被反射回反应堆内。显然,铅壁设计得越厚,穿透的概率就越小,反应堆就越安全。由于每次碰撞后弹出的角度是随机的,因此中子最后是穿透,还是被吸收或返回,也是随机的,是由大量中子运动的统计规律决定的。要导出铅壁厚和穿透率之间的关系,用解析成员函数是极为困难的,用计算机模拟会使问题的求解大大简化。

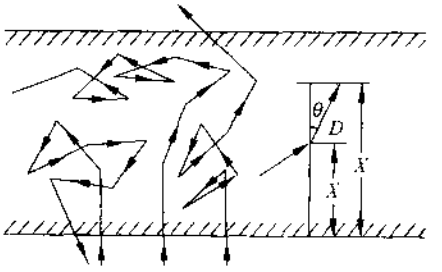

图 2.4 中子穿壁运动

1. 问题分析

为了建立概率模拟模型,首先分析一个中子在铅壁内的运动情况。

中子在壁内的运动与其每次与铅原子碰撞后的弹射角 θ 有关,由图 2.4 可知,这个角是随机的,设为

```
θ = 2 * π * random (10 001)/10 000
```

设中子在壁内与某一铅原子碰撞时距内壁距离为 x,则下一次碰撞前 X 产生的变化为

```
x = x + d * cos (2 * π * random (10 001)/10 000)
```

这里,表达式 random (10 001)/10 000 旨在产生小于 1 的随机数。显然,取 10 000 要比取 10 所产生的随机数要精确一些。若以铅原子直径为单位,可以写为

```
x += cos (2 * π * random (10 001)/10 000)
```

设反应堆的壁厚为 m * d，每个中子在铅壁内碰撞 n0 次后其能量就会被铅原子吸收。那么当碰撞次数 n ≥ n0 时，就可以由 x 的变化表明它是被吸收了（0 ≤ x ≤ m * d），还是返回到反应堆（x < 0），或是扩散到了反应堆外（X > m * d）。模拟一个中子的运动，得到一个结果，从对大量中子运动进行模拟的结果，可以统计出中子的穿透率 NP%、吸收率 NA% 和返回率 NR%。

2. 反应堆模拟类

```
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#define N0          10          // 中子最多碰撞数
#define PI          3.1415926
#define RAND_MAX    10000

class AnaReactor                // 反应堆为 reactor
{
    int  mWallThick;            // 反应堆壁厚(铅原子直径倍数)
    int  mNumNeutron;           // 用来模拟的中子数

  public:
      AnaReactor (int mWT,int nNumNeu)
      {
          mWallThick = mWT;
          mNumNeutron = nNumNeu;
      }

  void AnalogComputing()
  {
      int nr = 0, np = 0, na = 0 ;
      int n;                    // 计算中子在壁内碰撞次数的变量
      double x ;                // 计算中子进入壁深度的变量
      for(int i = 1;i <= mNumNeutron; i++){
          x = 0,n = 0;
          do{
              x += cos(PI * ran(10001) / RAND_MAX);
              n ++;
              if(x < 0)             {nr ++; break;}
              if(x > mWallThick)    {np ++; break;}
              if(n > N0)            {na ++; break;}
          } while(1);
```

```
        }
        cout << "\nnp =" << 100 * np/ mNumNeutron << "%,na ="
        << 100 * na/ mNumNeutron << "%,nr = " << 100 * nr/ mNumNeutron
        << "%";
    }
};
```

在本例中使用的 do … while 中，无显式的循环变量，并且判断表达式是一个固定的 1。这表明该结构是一个永循环结构。系统运行到永循环结构时，将一直运行下去。这时，应当有一个事件来结束永循环，否则将毫无意义。本例中，以计算的 x 或 n 的值，并通过 break 语句，来结束永循环。

3. 测试程序

用主程序对上述类进行测试。

```
int main()
{
    AnaReactor anReac(20,1000);
    anReac. AnalogComputing();
    return 0;
}
```

2.2.5 递归

例 2.2.11 汉诺塔(Tower of Hanoi)问题。古代印度布拉玛庙里僧侣玩一种游戏，游戏的装置是一块铜板，上面有 3 根杆，在 a 杆上自下而上、由大到小顺序地串有 64 个金盘(图 2.5)。游戏的目的是把 a 杆上的金盘全部移到 b 杆上。条件是，一次只能够动一个盘，可以借助 a 与 c 杆；移动时不允许大盘在小盘上面。容易推出，n 个盘从一根杆移到另一根杆需要 2^n-1 次，所以 64 个盘的移动次数为：$2^{64}-1=18\ 446\ 744\ 073\ 709\ 511\ 615$。这是一个天文数字，即使一台功能很强的现代计算机来解汉诺塔问题，每一微秒可能模拟(不输出)一次移动，那么也需要几乎 100 万年。而如果每秒移动一次，则需近 5 800 亿年，目前从能

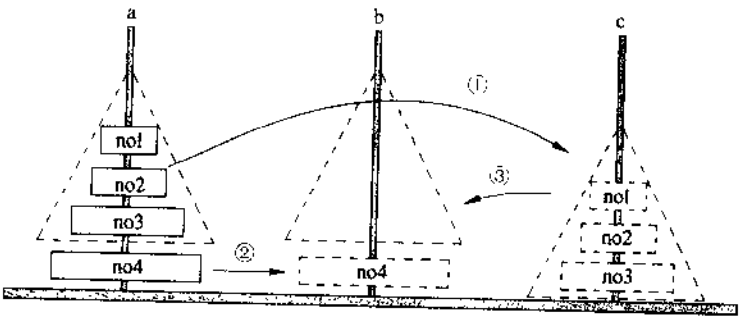

图 2.5 汉诺塔游戏

源角度推算，太阳系的寿命也只有150亿年。

解：

1. 类界面及其测试程序的轮廓

(1) 类界面

```
class Hanoi
{
        int        mDiscNumber;
    public:
                   Hanoi(int iDiscNum)
                   {
                       mDiscNumber = iDiscNum;
                   }
        void       MoveDisc(int n, char a,char b, char c);
};
```

(2) 测试函数的基本形式

```
int main()
{
    Hanoi     h(3);
    h.MoveDisc();
}
```

2. 成员函数 MoveDisc(int n, char a, char b, char c)的实现

对于移动盘子的函数 MoveDisc(n, a, b, c)，描述了借助 c 杆将 a 杆上的盘子移到 b 杆上。这个函数中的 4 个参数分别为：

- 第 1 个参数：盘子总数；
- 第 2 个参数：源柱名称；
- 第 3 个参数：目的柱名称；
- 第 4 个参数：辅助柱名称。

想起来，这是一个非常麻烦的过程，并且盘子越多，越麻烦。但是用下面的思路可以非常自然、又很简单地描写其实现过程：

第 1 步：先把 n-1 个盘子设法借助 b 杆放到 c 杆，如图 2.5 中的箭头①所示，记作：MoveDisc(n-1,a,c,b)；

第 2 步：把第 n 个盘子从 a 杆移到 b 杆，如图 2.5 中的箭头②所示；

第 3 步：把 c 杆上的 n-1 个盘子借助 a 杆移到 b 杆，如图 2.5 中的箭头③所示，记作：MoveDisc(n-1,c,b,a)。

于是便可以非常简单地用下面的程序段来描述函数 MoveDisc()。

```
void Hanoi::MoveDisc(int n,char a,char b,char c)          // 将 n 个盘子从 a 借 c 移到 b
{
    if(n > 0) /* 递归终止条件 */ {
        MoveDisc(n-1,a,c,b);                              // 第 1 步
        cout <<"Move disc no:"<< n <<"from pile"<< a <<"to"<< b << endl;
                                                          // 第 2 步
        MoveDisc(n-1,c,b,a);                              // 第 3 步
    }
}
```

3. 测试函数的进一步修正

```
#include <conio.h>                                        // 函数 clrscr()所在库的头文件
int main()
{
    Hanoi h(3);
    cout << "*********************************************\n";
    cout << "*        Program for simulating the solution    *\n";
    cout << "*           of the game of tower of Hanoi       *\n";
    cout << "*********************************************\n";
    cout << "\n";                                          // 输出一空行
    h.MoveDisc(3,'a','b','c');                            // 调用递归函数
}
```

4. 测试情形

```
*****************************************************
*        Program for Simulating the Solution        *
*           of the Game of Tower of Hanoi           *
*****************************************************
Move disc no:1 from pile a to b
Move disc no:2 from pile a to c
Move disc no:1 from pile b to c
Move disc no:3 from pile a to b
Move disc no:1 from pile c to a
Move disc no:2 from pile c to b
Move disc no:1 from pile a to b
```

2.2.6 对象(变量)的存储属性

例 2.2.12 一个时钟程序。

程序的功能：

- 时钟可以显示时(hour)、分(minute)和秒(second)；
- 可以对时钟进行初始化(用构造函数进行)；

• 可以对时钟进行设置。

```
#include <iostream.h>

class Clock                                                    // 定义时钟类
{
    private:
        int     mHour,mMinute,mSecond;
    public:                                                    // 外部接口
                Clock()
                ~Clock();
        void    SetTime(int NewH,int NewM,int NewS);           // 时间设置
        void    ShowTime();                                    // 显示时间
}

// 时钟类成员的实现
Clock::     Clock()
{
    mHour = 0;mMinute = 0;mSecond = 0;
}

Clock::
void    SetTime(int NewH,int NewM,int NewS)
{
    mHour = NewH;
    mMinute = NewM;
    mSecond = NewS;
}

Clock::
void    ShowTime()
{
    cout << mHour << ":" << mMinute << ":" << mSecond << endl;
}
// 声明一个外部对象
Clock globClock;

// 测试程序
Void main()
{
    cout << "Primal time: "
    globClock.ShowTime();

    globClock.SetTime(7,22,33);                                // 设置时间
```

```
    Clock myClock(globClock);                    // 用已有对象初始化新对象

    cout << "New time: "
    myClock.ShowTime();
}
```

测试结果：

```
Primal time: 0:0:0
New time: 7:22:33
```

例 2.2.13 王婆卖瓜问题。王婆卖瓜，每卖一个瓜要计出该瓜的重量，还要计出所卖出的总重量和总个数。王婆还允许退瓜。

用面向对象的程序设计设计王婆卖瓜的程序，首先要确定类的语义。一个自然的设想是以每个西瓜为对象，对西瓜类，应定义数据成员与成员函数。

根据题意，西瓜类的数据成员应当是：

```
weight          // 一个西瓜的重量
total_weight    // 累计重量
total_number    // 累计个数
```

这里，weight 是属于某个西瓜的，而 total_weight 与 total_number 是所有卖了的西瓜共有的。也就是说，不论某个西瓜对象是否还存在，这两个数据总是要保留的。为此，要把这两个数据定义为静态数据成员。即有

```
class watermelon
{
        float weight;
        static float total_weight;          // 静态成员
        static int total_number;            // 静态成员
            ...
};
```

静态数据成员有以下特点：

(1) 静态数据成员与静态变量一样，是在编译时创建并初始化的。程序开始执行时，它们已存在，一直到程序结束。因此任何对象都可以访问它们。使用静态数据成员可以减少对全局变量的需要。

(2) 对静态数据成员除要进行引用性声明外，还应在文件作用域内的某个地方进行定义性声明。如对上述两个静态数据成员可进行定义性声明如下：

```
...
float watermelon :: total_weight = 0;     // 定义性声明
int watermelon :: total_number = 0;       // 定义性声明
...
```

注意，初始化静态成员前要用类名指定作用域。

(3) 系统对每个类的每个静态成员只有一个实体，而不是像其他成员那样对每个对象都有它的一个实体。这样就能够保证共用数据的一致性

下面考虑成员函数的选取与定义。根据题意，类 watermelon 应有如下成员函数：

① 卖瓜——用构造函数模拟；

② 退瓜——用释放函数模拟；

③ 显示瓜重——用 display()成员函数模拟。

成员函数也可以定义为静态的。静态成员函数与静态数据成员一样，系统对每个类只建立一个实体。通常用不与特定对象相联系的静态成员函数来访问静态数据成员。这里为类 watermelon 定义一个模拟显示总重与总个数的静态成员函数。于是得到如下的 watermelon 类的定义。

```
class watermelon
  {
            float weight;
            static float total_weight;          // 静态数据成员
            static int total_number;            // 静态数据成员

      public:
            watermelon(float w)                 // 模拟售瓜
            {
                  weight=w;
                  total_weight+=weight;
                  total_number++;
            }

            ~ watermelon()                      // 模拟退瓜
            {
                  total_weight-=weight;
                  total_number--;
            }
            void display() ;                    // 显示瓜重
            static
            void total_disp() ;                 // 显示总重与总数的静态成员函数
  };

  float watermelon::total_weight=0;             // 静态数据成员的定义性声明
  int watermelon::total_number=0;               // 静态数据成员的定义性声明

   #include <iostream.h>
   #include "watermelon.hpp"

  void watermelon::display()                    // 显示瓜重
```

```
{
        cout<<"The watermelon weight is:";
        cout<<weight<<"\n";
}

void watermelon::total_disp()                    显示总重与总数的静态成员函数
{
        cout<<"Total weight is:";
        cout<<total_weight<<"\n";

        cout<<"Total number is:";
        cout<<total_number<<"\n";
}
```

下面是 watermelon 类的测试程序：

```
#include <iostream.h>
#include "watermelon.hpp"

void main()
{
    watermelon w1(3.5f);                    // 卖出 w1
                w2.display();
                watermelon::total_disp();

    watermelon w2(6.3f);                    // 卖出 w2
                w2.display();
                watermelon::total_disp();

    watermelon w3(5.6f);                    // 卖出 w3
                w3.display();
                watermelon::total_disp();
    w2.~watermelon();                       // 退回 w2
                watermelon::total_disp();
}
```

测试结果：

```
The watermelon weight is:3.5          (卖出 w1)
Total weight is:3.5
Total number is:1
The watermelon weight is:6.3          (卖出 w2)
Total weight is:9.8
Total number is:2
```

```
The watermelon weight is:5.6          (卖出 w3)
Total weight is:15.4
Total number is:3
Total weight is:9.1                   (退回 w2)
Total number is:2
```

2.3 自我测试

2.3.1 自测习题

题 2.3.1 下面哪一个函数原型声明是不合法的？为什么？

(1) func (int, int *);

(2) int& fsum (int,int);

(3) void fcoor(x, y);

(4) bool fdelete(int&,const int &);

题 2.3.2 指出下面程序的功能

```
#include <iostream.h>
main()
{
    int  i;
    int  sum = 0;
    cout << "Enter number\n";
    cin >> i;
    for ( ; ; )
    {
        if ( i < 0 )  break;
        sum += i;
        cin >> i;
    }
    cout << "sum=" << sum <<endl;
}
```

题 2.3.3 写出下面程序的执行结果。

```
#include <iostream.h>
main()
{
    int x, y = 1 , z ;
    if ( y != 0 ) x = 5;
    cout << "x=" << x << "\ t";
    if(y==0)
    x=3;
    else
    x=5;
```

```
    cout << "x=" << x << "\t";
    z=-1;
    if(z<0)
    if(y>0)
    x=3;
    else
    x=5;
    else
    x=7;
    cout << "x=" << x << "\t";
    cout << "z=" << z << "\t";
    if(x=y=z)
    x=3;
    cout << "x=" << x << "\t";
    cout << "z=" << z << "\t";
}
```

题 2.3.4 指出下列程序中存在的错误。

```
#include <iostream.h>
#define N 4
int main()
{
    for ( int  i = 0 ; i < N ; i++ )
        for ( int  j = 0 ,int  a[N][N]; j < N ; j++ )
          * ( * ( a + i ) + j ) = i + j ;
    for ( int i = 0 ; i < N ; i++ )
    {
        for ( int  j = 0 ; j < N ; j++ )
            cout << * ( * ( a + i ) + j ) << "  ";
        cout << endl;
    }
    return  0;
}
```

题 2.3.5 编写程序，输入一整数，判断它能否被 3、5、7 同时整除。

题 2.3.6 编写程序，求 100～999 中的"水仙花"数。若 3 个数其各位数字立方和等于该数本身，即为水仙花数，如 $153=1^3+5^3+3^3$，则 153 是一个"水仙花"数。

题 2.3.7 求 2～1 000 中的同构数。某数的平方，其低位与同该数本身相同，则称该数为同构数，如 $25^2=625$，25 是同构数。

题 2.3.8 输出 1～999 中能被 3 整除且至少有一位数字是 5 的所有整数。

题 2.3.9 给定一个数，并由键盘输入若干个数，找出与预先给定的数最接近的数，并指出它是由键盘输入的第几个数。

题 2.3.10 从键盘输入两个正整数 m 和 n，找出它们的最小公倍数。

题 2.3.11 求 Fibonacci 数列 1,1,2,3,5,8,13,…第 n 项的值。

题 2.3.12 编程验证：任何一个自然数 m 的立方均可以写成 m 个连续的奇数之和。例如：

$1^3=1$

$2^3=3+5$

$3^3=7+9+11$

$4^3=13+15+17+19$

题 2.3.13 换硬币：把一元人民币换成 5 分、2 分、1 分的硬币，有多少种换法？

题 2.3.14 百钱买百鸡问题：公元 5 世纪末，我国古代数学家张丘建在《算经》中提出了如下问题：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。凡百钱买百鸡，问鸡翁、母、雏各几何？使用 C++ 程序求解该题。

题 2.3.15 五家共井问题：我国古代数学巨著《九章算数》卷第八·第十三题为"五家共井，甲二绠(汲水用的井绳)不足，如(接上)乙一绠；乙三绠不足，如丙一绠；丙四绠不足，如丁一绠；丁五绠不足，如戊一绠；戊六绠不足，如甲一绠。如各得所不足一绠，皆逮(dia，及)。问井深，绠长各几何。答曰："井深七丈二尺一寸，甲绠长二丈六尺五寸，乙绠长一丈九尺一寸，丙绠长一丈四尺八寸，丁绠长一丈三尺九寸，戊绠长七尺六寸。"这是世界上最古老的不定方程问题。

题 2.3.16 奇妙的算式：有人用字母代替十进制数字写出下面的算式，请找出这些字母代表的数字：

```
    E G A L
×         L
-----------
  L A G E
```

题 2.3.17 牛的繁殖问题：有位科学家曾出了这样一道数学题：有一头母牛，它每年年初要生一头小母牛；每头小母牛从第四个年头起，每年年初也要生一头小母牛。按此规律，若无牛死亡，第 20 年头上共有多少头母牛？

题 2.3.18 把下列数列延长到第 50 项：

1,2,5,10,21,42,85,170,341,682,…

题 2.3.19 一个排球运动员一人练习托球，第 2 次只能托到前一次托起高度的 2/3 偏高 25cm。按此规律，他托到第 8 次时，只托起了 3.5m，问他第 1 次托起了多高？

题 2.3.20 假设银行一年整存整取的月息为 0.32%，某人存入了一笔钱，在 5 年内，每年年底都能取出 200 元，并到第 5 年年底刚好用完。请设计一个 C++ 程序，计算他当初共存了多少钱。

题 2.3.21 某日，王母娘娘送唐僧一批仙桃，唐僧命八戒去挑。八戒从娘娘宫挑上仙桃出发，边走边望着眼前罗筐中的仙桃咽口水，走到 128 里时，倍觉心烦腹饥口干不能再忍，于是找了个背静处开始吃起前头箩筐中的仙桃来，越吃越有兴头，不觉竟将一筐仙桃吃尽，才猛然觉得大事不好。正在无奈之时，发现身后还有一筐，便转悲为喜，将身后的一筐仙桃一分为二，重新上路。走着走着，又馋病复发，才走了 64 里路，便故伎重演，又在吃光一筐仙桃后，把另一筐一分为二，才肯上路。以后，每走前一段路的一半，便吃光一头箩筐中的仙

桃，才上路。如此这般，最后走了一里走完，正好遇上师傅。师傅一看，两个箩筐中每个只有一只仙桃，于是大怒，要八戒交代一路偷吃了多少仙桃。八戒掰着指头，好几个时辰也回答不出。

请设计一个 C++ 程序，为八戒计算一下，一路偷吃了多少仙桃。

题 2.3.22 编写程序，用递归方法求以下函数的前 10 项和。

$$\begin{cases} f(1)=1 & n=1 \\ f(2)=2 & n=2 \\ f(n)=f(n-1)+2\times f(n-2) & n>2 \end{cases}$$

题 2.3.23 编写程序，用递归方法测试下列函数。

$$p(n,x)=\begin{cases} 1 & (n=0) \\ x & (n=1) \\ ((2\times n-1)\times x\times p(n-1,x)-(n-1)\times p(n-2,x))/n & (n>1) \end{cases}$$

题 2.3.24 用二分法计算 x^n 的值(提示：当 n 为偶数时，使用 $x^n=x^p\times x^p$；当 n 为奇数时，使用 $x^n=x\times x^p\times x^p$)。

题 2.3.25 在口袋中放有手感相同的 3 只红球和 4 只白球。随机地从口袋中摸出 3 只球来，然后放回口袋中，共摸 500 次，问摸到 3 只都是红球和白球的几率各是多少？

题 2.3.26 用蒙特卡罗方法求 π 的近似值。

提示：蒙特卡罗方法(Monte Carlo Method)也称随机抽样技术或统计实验方法，是一种应用随机数进行仿真实验的方法。用蒙特卡罗方法计算 π 的近似值的基本思路是：

由于圆的面积有

$$S=\pi R^2$$

当 $R=1$ 时，$S=\pi$。而圆的方程为

$$x^2+y^2=1$$

如图 2.6 所示，1/4 的圆面积为 x 轴、y 轴和上述方程所包围的部分。如果在 1×1 的面积中均匀地落入随机点，则落入 1/4 圆中的点的概率就是 1/4 圆的面积。

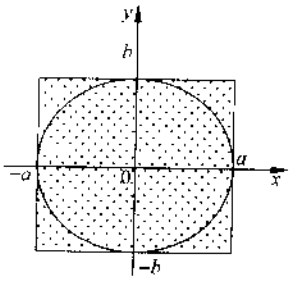

图 2.6 蒙特卡罗方法

2.3.2 参考答案

题 2.3.1 参考答案：

解答：

(1) func (int , int *);

不合法，函数必须明确指定返回值。

(2) int& fsum (int,int);

合法。

(3) void fcoor(x, y);

不合法，形参 x 和 y 必须声明。

(4) bool fdelete(int&,const int &);

合法。

题 2.3.2 参考答案：

解答：该程序的功能是从键盘上连续输入的若干个整数累加之和，当输入的是负数时表示累加结束，程序输出累加结果后结束运行。下面是运行一次该程序的情况：

```
Enter number
25
15
50
42
10
-1
sum=142
```

题 2.3.3 参考答案：

执行结果：

```
x=5    x=5    x=3    z=-1    z=3    z=-1
```

题 2.3.4 参考答案：

该程序出现的错误位置和性质，参见下列程序注释。

```
#include <iostream.h>
#define N 4
int main()
{
    for ( int  i = 0 ; i < N ; i++ )
        for ( int  j = 0,int  a[N][N]; j < N ; j++ )      //a 仅作用于里层循环
        * ( * ( a + i ) + j ) = i + j ;
    for ( int i = 0 ; i < N ; i++ )                       // i 重复声明错误
    {
        for ( int  j = 0 ; j < N ; j++ )
            cout << * ( * ( a + i ) + j ) << "  ";        // a 没有声明错误
    cout << endl;
    }
    return  0;
}
```

修改程序中所含错误后的参考程序如下：

```
#include <iostream.h>
#define N 4
int main()
{
    int  a[N][N];
    for ( int  i = 0 ; i < N ; i++ )
        for ( int  j = 0 ; j < N ; j++ )
```

```
            *(*(a+i)+j)=i+j;
    for(i=0;i<N;i++)
    {
        for(int j=0;j<N;j++)
            cout<<*(*(a+i)+j)<<"  ";
    cout<<endl;
    }
    return 0;
}
```

测试结果：

```
0  1  2  3
1  2  3  4
2  3  4  5
3  4  5  6
```

题 2.3.5 参考程序：

```
#include <iostream.h>
class IntegerDiv{
public:
    bool intdiv(int n)
    {
        if(n%3==0&&n%5==0&&n%7==0)
            return (true);
        else
            return (false);
    }
};
void main()
{
    int n;
    IntegerDiv id;
    cout<<"请输入一个整数:";
    cin>>n;
    if(id.intdiv(n))
        cout<<"数"<<n<<"能被 3、5、7 同时整除."<<endl;
    else
        cout<<"数"<<n<<"不能被 3、5、7 同时整除."<<endl;
}
```

测试结果：

```
请输入一个整数: 105
数 105 能被 3、5、7 同时整除.
```

题 2.3.6 参考程序：

```
#include <iostream.h>
class Daffodil{
public:
    void daffnum()
    {
        int x,y,z;
        for(int i=100;i<=999;i++)
        {
            z=i%10;
            y=(i%100-z)/10;
            x=(i-(y*10+z))/100;
            if(i==(x*x*x+y*y*y+z*z*z))
                cout<<i<<endl;
        }
    }
};
void main()
{
    Daffodil dnum;
    dnum.daffnum();
}
```

测试结果：

```
153
370
371
407
```

题 2.3.7 参考程序：

```
#include <iostream.h>
class CUniform{
public:
    void print()
    {
        int s;
        for(int i=2;i<1000;i++)
        {
            s=i*i;
            if(s<10&&s==i)
                cout<<i<<" 是同构数，"<<i<<"^2="<<s<<endl;
            else if(s<100&&i==s%10)
                cout<<i<<" 是同构数，"<<i<<"^2="<<s<<endl;
            else if(s<1000&&i==s%100)
                cout<<i<<" 是同构数，"<<i<<"^2="<<s<<endl;
```

```
        }
    }
};
void main()
{
        CUniform same;
        same.print();
}
```

测试结果：

```
5 是同构数，5^2=25
6 是同构数，6^2=36
25 是同构数，25^2=625
```

题 2.3.8 参考程序：

```
#include <iostream.h>
class CIntNum{
public:
    void intdiv()
    {
        int k=0;
        bool five=false;
        for(int i=1;i<=999;i++)
        {
            if(i%3==0)
            {
                if(i<10 && i==5)
                {
                    five=true;
                    k++;
                    cout<<i<<"  ";
                }
                else if((i<100) && ((i%10==5)||(i/10==5)))
                {
                    five=true;
                    k++;
                    cout<<i<<"  ";
                }
                else if((i<1000) && ((i%100==5)||(i/100==5)||
                                    ((i%100-i%10)/10==5)))
                {
                    five=true;
                    k++;
                    cout<<i<<" ";
```

```
                }
                if(five&&k%10==0)
                {
                    cout<<endl;
                    five=false;
                }
            }
        }
        cout<<endl;
    }
};
void main()
{
    CIntNum n35;
    n35.intdiv();
}
```

测试结果：

```
15   45  51  54  57  75 105 150 153 156
159 252 255 258 351 354 357 405 450 453
456 459 501 504 507 510 513 516 519 522
525 528 531 534 537 540 543 546 549 552
555 558 561 564 567 570 573 576 579 582
585 588 591 594 597 651 654 657 705 750
753 756 759 852 855 858 951 954 957
```

题 2.3.9 参考程序：

```
#include <iostream.h>
#include "math.h"
class CloseNum{
private:
    int x,y;
public:
    void findnum()
    {

        int i=1,k,z;
        cout<<"请输入预先给定的数:";
        cin>>x;
        cout<<"输入第"<<i<<"个数:";
        cin>>y;
        k=i;
        z=y;
```

```
        while(y!=-1)
        {
            i++;
            cout<<"输入第"<<i<<"个数:";
            cin>>y;
            if(fabs(x-y)<fabs(x-z))
            {
                k=i;
                z=y;
            }
        }
        cout<<"预先给定的数:"<<x<<endl;
        cout<<"最接近的数是第"<<k<<"个数:"<<z<<endl;
    }
};
void main()
{
    CloseNum cnum;
    cnum.findnum();
}
```

测试结果:

```
请输入预先给定的数:25
输入第 1 个数:66
输入第 2 个数:56
输入第 3 个数:34
输入第 4 个数:90
输入第 5 个数:24
输入第 6 个数:35
输入第 7 个数:26
输入第 8 个数:45
输入第 9 个数:-1
预先给定的数:25
最接近的数是第 5 个数:24
```

题 2.3.10 参考程序:

```
#include <iostream.h>
class CMultiple{
public:
    int minimum(int x,int y)
    {
        int i;
        for(x<y? i=x:i=y;i<=x*y;i++)
            if(i%x==0&&i%y==0)
```

```
            return(i);
        }
};
void main()
{
    int m,n;
    CMultiple mult;
    cout<<"m=";
    cin>>m;
    cout<<"n=";
    cin>>n;
    cout<<"两个正整数"<<m<<"和"<<n<<"的最小公倍数是:";
    cout<<mult.minimum(m,n)<<endl;
}
```

测试结果:

```
m=10
n=100
两个正整数 10 和 100 的最小公倍数是:100
```

题 2.3.11 参考程序:

```
#include <iostream.h>
class Fibonacci{
public:
    int fibnum(int n)
    {
        int f1=1,f2=1,f3;
        for(int i=3;i<=n;i++)
        {
            f3=f1+f2;
            f1=f2;
            f2=f3;
        }
        return f2;
    }
};
void main()
{
    int n;
    Fibonacci fib;
    cout<<"请输入项数 n 的值:";
    cin>>n;
    cout<<"第"<<n<<"项的值是:"<<fib.fibnum(n)<<endl;
}
```

测试结果：

```
请输入项数 n 的值:7
第 7 项的值是:13
```

题 2.3.12 参考程序：

```
#include <iostream.h>
class NumCube{
public:
    void oddnum(int m)
    {
        int i,k1=1,k2,sum=0;
        while(sum!=m*m*m)
        {
            sum=0;
            k2=k1;
            for(i=1;i<=m;i++)
            {
                sum+=k2;
                k2+=2;
            }
            k1+=2;
        }
        cout<<m<<"^3=";
        for(i=k1-2;i<k2-2;i+=2)
            cout<<i<<"+";
        cout<<k2-2<<endl;
    }
};
void main()
{
    int m;
    NumCube ncob;
    cout<<"请输入一个自然数:";
    cin>>m;
    ncob.oddnum(m);
}
```

测试结果：

```
请输入一个自然数:4
4^3=13+15+17+19
```

题 2.3.13 参考程序：

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

```
class OneYuan{
public:
    int coin521(int n)
    {
        int kind=0;
        for(int i=1;i<=n;i++)
          for(int j=1;j<=n;j++)
            for(int k=1;k<=n;k++)
              if(n==5*i+2*j+k) kind++;
          return kind;
    }
};
void main()
{
    OneYuan ayuan;
    cout<<"一元人民币换成5分、2分、1分的硬币有";
    cout<<ayuan.coin521(100)<<"种换法"<<endl;
}
```

测试结果：

一元人民币换成5分、2分、1分的硬币有461种换法

题 2.3.14 参考程序：

```
#include <iostream.h>
class Chicken{
public:
    void chicnum(int n)
    {
        cout<<"鸡翁   鸡母   鸡雏"<<endl;
        for(int i=1;i<=n;i++)
          for(int j=1;j<=n;j++)
            for(int k=1;k<=n;k++)
                if((n==5*i+3*j+k/3)&&(k%3==0)&&(n==i+j+k))
                    cout<<i<<"     "<<j<<"     "<<k<<endl;
    }
};
void main()
{
    Chicken chic;
    chic.chicnum(100);
}
```

测试结果：

鸡翁	鸡母	鸡雏
4	18	78
8	11	81
12	4	84

题 2.3.15 参考程序：

```
#include <iostream.h>
class CRopeWell{
public:
    void getLength()
    {
        int a,b,c,d,e,depth=50;    //设甲绠长为 a,乙绠长为 b,丙绠长为 c,丁绠长为 d,
                                   //戊绠长为 e。再假设水井深度为 50 寸≤depth≤1000 寸
                                   //则有关系:2a+b=3b+c=4c+d=5d+e=6e+a=depth
                                   //解上述 5 个方程得:721 c=148 depth
        while(depth<=1000||148*depth%721==0)
            depth++;               //对每个 depth 值穷举测试
        c=148*depth/721;
        d=depth-4*c;
        e=depth-5*d;
        a=depth-6*e;
        b=depth-2*a;
        cout<<"甲绠长"<<a<<"寸"<<endl;
        cout<<"乙绠长"<<b<<"寸"<<endl;
        cout<<"丙绠长"<<c<<"寸"<<endl;
        cout<<"丁绠长"<<d<<"寸"<<endl;
        cout<<"戊绠长"<<e<<"寸"<<endl;
        cout<<"井深"<<depth<<"寸"<<endl;
    }
};

void main()
{
    CRopeWell rw;
    rw.getLength();
}
```

测试结果：

```
甲绠长 425 寸
乙绠长 151 寸
丙绠长 205 寸
丁绠长 181 寸
戊绠长 96 寸
井深 1001 寸
```

题 2.3.16 参考程序：

```
#include <iostream.h>
class CharDigit{
public:
    void pchardig()
    {
        int E,G,A,L,i=1;
        for(E=0;E<=9;E++)
          for(G=0;G<=9;G++)
            for(A=0;A<=9;A++)
                for(L=0;L<=9;L++)
                    if((E*1000+G*100+A*10+L)*L==(L*1000+G*100+A*10+E))
                    {
                        cout<<" E="<<E<<" G="<<G<<" A="<<A<<" L="<<L<<" ";
                        if((i++)%4==0) cout<<endl;
                    }
    }
};

void main()
{
    CharDigit cd;
    cd.pchardig();
}
```

测试结果：

```
E=0 G=0 A=0 L=0 E=1 G=0 A=0 L=1 E=1 G=0 A=1 L=1 E=1 G=0 A=2 L=1
E=1 G=0 A=3 L=1 E=1 G=0 A=4 L=1 E=1 G=0 A=5 L=1 E=1 G=0 A=6 L=1
E=1 G=0 A=7 L=1 E=1 G=0 A=8 L=1 E=1 G=0 A=9 L=1 E=1 G=1 A=0 L=1
E=1 G=1 A=1 L=1 E=1 G=1 A=2 L=1 E=1 G=1 A=3 L=1 E=1 G=1 A=4 L=1
E=1 G=1 A=5 L=1 E=1 G=1 A=6 L=1 E=1 G=1 A=7 L=1 E=1 G=1 A=8 L=1
E=1 G=1 A=9 L=1 E=1 G=2 A=0 L=1 E=1 G=2 A=1 L=1 E=1 G=2 A=2 L=1
E=1 G=2 A=3 L=1 E=1 G=2 A=4 L=1 E=1 G=2 A=5 L=1 E=1 G=2 A=6 L=1
E=1 G=2 A=7 L=1 E=1 G=2 A=8 L=1 E=1 G=2 A=9 L=1 E=1 G=3 A=0 L=1
E=1 G=3 A=1 L=1 E=1 G=3 A=2 L=1 E=1 G=3 A=3 L=1 E=1 G=3 A=4 L=1
E=1 G=3 A=5 L=1 E=1 G=3 A=6 L=1 E=1 G=3 A=7 L=1 E=1 G=3 A=8 L=1
E=1 G=3 A=9 L=1 E=1 G=4 A=0 L=1 E=1 G=4 A=1 L=1 E=1 G=4 A=2 L=1
E=1 G=4 A=3 L=1 E=1 G=4 A=4 L=1 E=1 G=4 A=5 L=1 E=1 G=4 A=6 L=1
E=1 G=4 A=7 L=1 E=1 G=4 A=8 L=1 E=1 G=4 A=9 L=1 E=1 G=5 A=0 L=1
E=1 G=5 A=1 L=1 E=1 G=5 A=2 L=1 E=1 G=5 A=3 L=1 E=1 G=5 A=4 L=1
E=1 G=5 A=5 L=1 E=1 G=5 A=6 L=1 E=1 G=5 A=7 L=1 E=1 G=5 A=8 L=1
E=1 G=5 A=9 L=1 E=1 G=6 A=0 L=1 E=1 G=6 A=1 L=1 E=1 G=6 A=2 L=1
E=1 G=6 A=3 L=1 E=1 G=6 A=4 L=1 E=1 G=6 A=5 L=1 E=1 G=6 A=6 L=1
E=1 G=6 A=7 L=1 E=1 G=6 A=8 L=1 E=1 G=6 A=9 L=1 E=1 G=7 A=0 L=1
```

```
E=1 G=7 A=1 L=1 E=1 G=7 A=2 L=1 E=1 G=7 A=3 L=1 E=1 G=7 A=4 L=1
E=1 G=7 A=5 L=1 E=1 G=7 A=6 L=1 E=1 G=7 A=7 L=1 E=1 G=7 A=8 L=1
E=1 G=7 A=9 L=1 E=1 G=8 A=0 L=1 E=1 G=8 A=1 L=1 E=1 G=8 A=2 L=1
E=1 G=8 A=3 L=1 E=1 G=8 A=4 L=1 E=1 G=8 A=5 L=1 E=1 G=8 A=6 L=1
E=1 G=8 A=7 L=1 E=1 G=8 A=8 L=1 E=1 G=8 A=9 L=1 E=1 G=9 A=0 L=1
E=1 G=9 A=1 L=1 E=1 G=9 A=2 L=1 E=1 G=9 A=3 L=1 E=1 G=9 A=4 L=1
E=1 G=9 A=5 L=1 E=1 G=9 A=6 L=1 E=1 G=9 A=7 L=1 E=1 G=9 A=8 L=1
E=1 G=9 A=9 L=1
```

题 2.3.17 参考程序：

```
#include <iostream.h>
class CCow{
public:
    int getCow_Sum(int n)      //计算母牛头数的类函数
    {
        int cow_sum1=2,cow_sum2=3,cow_sum3=4,cow_sum=0;
        for(int i=4;i<=n;i++)
        {
            cow_sum=cow_sum1+cow_sum3;
            cow_sum1=cow_sum2;
            cow_sum2=cow_sum3;
            cow_sum3=cow_sum;
        }
        return cow_sum;
    }
};
void main()
{
    CCow cow;
    cout<<"第 20 年头上共有的母牛头数:"<<cow.getCow_Sum(20)<<endl;
}
```

测试结果：

```
第 20 年头上共有的母牛头数:2745
```

题 2.3.18 参考程序：

```
#include <iostream.h>
#include <iomanip.h>
class NRow{
public:
    void printrow(int n)
    {
```

```
        int t=0;
        for(int i=0;i<n;i++)
        {
            t=2*t;
            if(i%2==0)t++;
            cout<<setw(10)<<t;
            if(((i+1)%5)==0)cout<<endl;
        }
    }
};
void main()
{
    NRow row;
    row.printrow(30);
}
```

测试结果：

1	2	5	10	21
42	85	170	341	682
1365	2730	5461	10922	21845
43690	87381	174762	349525	699050
1398101	2796202	5592405	11184810	22369621
44739242	89478485	178956970	357913941	715827882

题 2.3.19 参考程序：

```
#include <iostream.h>
class Volleyboll{
public:
    double player(double x)
    {
        for(int i=1;i<8;i++)
            x=(x-0.25)*3.0/2.0;
        return x;
    }
};
void main()
{
    Volleyboll vollplay;
    cout<<"排球运动员第1次托起的高度是:"<<vollplay.player(3.5)<<endl;
}
```

测试结果：

```
排球运动员第1次托起的高度是:47.7363
```

题 2.3.20 参考程序：

```
#include <iostream.h>
class CBank{
public:
    double deposit_get()
    {
        double payin,interest=1;
        payin=200;
        for(int i=1;i<=12;i++)      //interest=(1+0.0032)^12
            interest*=1.0032;
        for(int j=1;j<5;j++)
            payin=payin/interest+200.0;
        return (payin/interest);
    }
};
void main()
{
    CBank bk;
    cout<<"当初存入银行的钱数是:"<<bk.deposit_get()<<endl;
}
```

测试结果：

```
当初存入银行的钱数是:892.663
```

题 2.3.21 参考程序：

```
#include <iostream.h>
class CPeach{
public:
    int eat_Peach()
    {
        int time=0,peach=2;
        for(int i=128;i>0;i/=2)     //计算八戒一路偷吃仙桃次数 time
                                    //(包括最后一里遇到师傅的一次)time
            time++;
        for(int j=1;j<time;j++)     //计算仙桃总数
            peach*=2;
        return (peach-2);           //剩下的两个仙桃没吃不计算
    }
};
void main()
{
    CPeach bajie;
    cout<<"八戒一路偷吃的仙桃个数是:"<<bajie.eat_Peach()<<" 个"endl;
}
```

测试结果：

八戒一路偷吃的仙桃个数是:254 个

题 2.3.22 参考程序：

```
#include <iostream.h>
#define N 10
class CRcursion{
public:
    int f(int n)
    {
        if(n==1)
            return 1;
        else if(n==2)
            return 2;
        else
            return f(n-1)+2*f(n-2);
    }
};
void main()
{
    CRcursion rcur;
    cout<<"前"<<N<<"项和是:"<<rcur.f(N)<<endl;
}
```

测试结果：

前 10 项和是:512

题 2.3.23 参考程序：

```
#include <iostream.h>
class CRcurTest{
public:
    double p(int n,double x)
    {
        if(n==0)
            return 1;
        else if(n==1)
            return x;
        else
            return((2*n-1)*x*p(n-1,x)-(n-1)*p(n-2,x))/n;

    }
};
void main()
```

```
{
    int n;
    double x;
    CRcurTest rtest;
    cout<<"输入正整数 n: ";
    cin>>n;
    cout<<"输入浮点数 x: ";
    cin>>x;
    cout<<"函数值是: "<<rtest.p(n,x)<<endl;
}
```

测试结果:

```
输入正整数 n: 10
输入浮点数 x: 1.5
函数值是: 2897.28
```

题 2.3.24 参考程序:

```
#include <iostream.h>
class xPower{
private:
    double x;
public:
    xPower(double X)
    {
        x=X;
    }
    double nMulti(int n)
    {
        if(n==1)
            return x;
        else if(n%2)
            return (x*nMulti(n/2)*nMulti(n/2));
        else
            return (nMulti(n/2)*nMulti(n/2));
    }
};
void main()
{
    xPower xn(2.0);                 // x 取值 2
    cout<<xn.nMulti(7)<<endl;       // n 取值 7
}
```

测试结果:

题 2.3.25 参考程序：

```
#include <iostream.h>
#include <stdlib.h>
class CBool{
private:
        int rb3,wb3;
public:
        CBool()
        {
            rb3=wb3=0;
        }
        void extract(int times,double &rx,double & wy)
        {
            for(int i=1;i<=times;i++)
        {
            int rb=0,wb=0;
            for(int j=1;j<=3;j++)
                (7 * rand() / RAND_MAX + 1)<=3? rb++:wb++;
            if(rb==3) rb3++;
            if(wb==3) wb3++;
        }
        rx=double(rb3)/double(times);
        wy=double(wb3)/double(times);
    }
};
void main()
{
    double r,w;
    CBool rwbool;
    rwbool.extract(500,r,w);
    cout<<"摸到 3 只都是红球的概率是:"<<100 * r<<" %"<<endl;
    cout<<"摸到 3 只都是白球的概率是:"<<100 * w<<" %"<<endl;
}
```

测试结果：

```
摸到 3 只都是红球的概率是:8.2 %
摸到 3 只都是白球的概率是:19.8 %
```

题 2.3.26 参考程序：

```
#include <iostream.h>
#include <stdlib.h>
class Circle_Pai{
public:
    double compai()
    {
        int m=0;
        double x,y;
        for(int i=1;i<5000;i++)
        {
            x=(100 * rand() / RAND_MAX );    //100 为精度比例尺
            y=(100 * rand() / RAND_MAX );
            if((x*x+y*y)/10000<=1)           //圆中的随机抽取的 x 和 y 应满足的关系
                m++;
        }
        return 4.0*m/5000.0;                 //m/5000 为 1/4 圆面积
                                             //令 R=1,则圆面积 S=4*m/5000=PI
    }
};
void main()
{
    Circle_Pai pi;
    cout<<"pi 的值是:"<<pi.compai()<<endl;
}
```

测试结果:

```
pi 的值是:3.1416
```

第3章 数据结构

3.1 编程知识

3.1.1 数组

1. 数组的特征

(1) 名字:用以对数组各元素的整体标识,这个名字称为数组名;
(2) 类型:数组各元素的类型;
(3) 大小:可容纳的数组元素个数(注意,不是字节数);
(4) 静态性:数组的大小必须是编译时可知的,不能在程序运行的过程中才决定;
(5) 存储:占有一个连续的内存空间。

2. 一维数组的定义和初始化

(1) 定义格式
数组的声明语句是定义性声明,格式如下:

类型 数组名[大小];

(2) 数组的初始化方式
在声明的同时将初始化表达式按元素顺序依次写在一对花括号内。默认规则如下:
① 初始化时也可以不指定数组的大小,编译器会根据初始值的个数自动决定数组的大小。
② 允许省略为0的元素值,但逗号数不能缺少。
③ 当最后的几个元素初值为0时,可以只写出前面的数列,但数组体积不可省略。

3. C++ 多维数组的特点

多维数组就是多级数组,它的元素要用数组名加 n 个下标来标识。

4. 对象数组

声明一个对象数组的目的是为它开辟一个存储空间。声明语句格式如下:

类名 数组名 [对象个数];

声明一个对象数组时,要求类定义中必须有一个显式的无参构造函数。C++ 执行自动对象数组的声明语句时,将自动为每一个数组元素调用一次无参构造函数(对静态对象数组是在 main()执行之前调用)。

在声明对象数组的同时,还可以对数组元素初始化。初始化用无名对象进行。

3.1.2 指针

1. 指针与变量的间接引用

指针是存储地址的变量。一个指针有两个属性：地址和类型。指针的地址用于表明所存储的数据的起始地址，是它的值；指针的类型表明它指向的空间用于存储什么类型的数据。

在声明语句中，一级指针使用记号 *，二级指针使用符号 ** 来标识。

一个变量可以直接用其标识符指称，也可以用指向它的指针指称。用指针指称它所指向的实体，称为指针的间接引用（递引用）。

对于语句

```
float f;
float * pf = &f;
```

应搞清下面一些表达式的含义：

f：变量；

pf：指向 f 的指针变量，其值为 f 的地址；

&f：变量 f 的地址，与 pf 等价；

* pf：pf 所指向的变量，与 f 等价；

*(&f)：与 * pf（即 f）等价；

&(* pf)：与 &f（即 pf）等价。

2. 指针的运算

指针运算的限制：

(1) 与整数相加、减运算；

(2) 同一数组中各元素地址间的关系运算与相减运算；

(3) 赋值运算。

3. 指向对象的指针与创建动态对象

一个对象一旦被创建，系统就给它分配一个存储空间，该存储空间的起点可以像数据对象的地址一样，使用指针变量操作。一个指向类的指针被初始化后便成为一个指向某对象的指针，即存有对象首地址的变量。也可以用一个对象地址来初始化一个对象指针，如

```
Person zh( "Zhang",50,'m' );      // 声明一个对象
Person * pzh = &zh;               // 用对象地址初始化对象指针
```

用对象指针引用对象成员要使用成员访问符 ->，如

```
Person * pzh = new Person( "Zhang",50,'m' );
pzh -> print();                   // 等价于 zh.print()
delete pzh;
```

为了创建动态对象必须声明一个指向其所在类的指针。

4. this 指针

this 指针是隐含在对象成员函数内的一种指针。当一个对象被创建后，它的每一个成员函数都含有一个系统自动生成的隐含的指针——this，用以保存这个对象的地址。因此 this 也称为"指向本对象的指针"，用它来存取类成员变量的格式为

this ->成员变量

3.1.3 数组与指针

1. 一维数组的指针形式

数组名是指向该数组起始元素的指针。

对于一维数组 a 来说，a 和 &a 的值相同，但它们所代表的意义不同，a+1 被解释为 a+1 * sizeof(int)，而 &a+1 被解释为 a+1 * sizeof(a)，其中 sizeof(a) 表示数组 a 的每个元素占用存储空间字节数的总和。

对于一维数组 a，有如下对应关系：

a+1	~	&a[1]	或	*(a+1)	~	a[1]
a+2	~	&a[2]		*(a+2)	~	a[2]
	…			…		
a+n	~	&a[n]		*(a+n)	~	a[n]

按照上述对应关系，可以分别通过两种方式引用数组元素：下标法（前面已经使用过的方法）和指针（地址）法。

图 3.1 为指向一维数组元素的指针与数组名以及下标变量间的对应关系示意图。

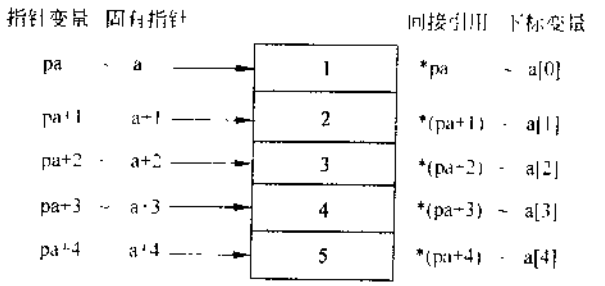

图 3.1 指向一维数组元素的指针与数组名以及下标变量间的对应关系

2. 二维数组的形式

二维数组中任一数组元素 a[i][j] 的地址可表示为：

a[i][j]的地址 = a[i] + j * sizeof(a 的基类型)
a[i] = a + i * sizeof(a[i])

其中，sizeof (a[i])表示 a[i]所代表的一维数组占用存储空间的字节数。

二维数组名 a 中的等价关系：

(1) 地址等价关系

二级指针表示		一级指针表示	地址
a	*a	a[0]	&a[0][0]
	*a+1	a[0]+1	&a[0][1]
	…	…	…
	*a+j	a[0]+j	&a[0][j]
a+1	*(a+1)	a[1]	&a[1][0]
	*(a+1)+1	a[1]+1	&a[1][1]
	…	…	…
	*(a+1)+j	a[1]+j	&a[1][j]
…	…	…	…
a+i	*(a+i)	a[i]	&a[i][0]
	*(a+i)+1	a[i]+1	&a[i][1]
	…	…	…
	*(a+i)+j	a[i]+j	&a[i][j]

(2) 引用等价关系

a[0]	* a	
a[1]	*(a+1)	
…		
a[i]	*(a+i)	
a[i][0]	* a[i]	**(a+i)
a[i][1]	*(a[i]+1)	*(*(a+i)+1)
…		
a[i][j]	*(a[i]+j)	*(*(a+i)+j)

3. 三维数组的指针形式

对三维数组来说，它指向第一页，如图 3.2 所示。数组名指针每增 1，则地址下移广义数组中的一个元素的位置。

4. 数组和指针参数

当要在两个函数间传送数组时，可以有两种方式：

(1) 传值传送：用下标变量进行；

(2) 传址传送：用数组名或指向数组的指针进行。

5. 动态内存分配的概念

(1) 意义

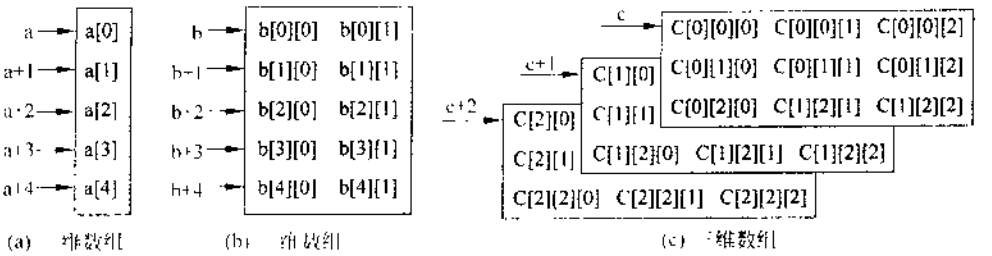

图 3.2 数组名指针的移动规律

程序员在任何时候都可以(用函数 new)从中获取自己需要大小的内存，也可以在任何需要的时候，按任意顺序(用函数 delete)释放在堆中占用的内存。

(2) 用 new 函数动态创建多维数组，其原型为

new 类型名[表达式 1][表达式 2]……[表达式 n]

其中，n 大于等于 1。除表达式 1 可以是任意整型表达式之外，其他表达式必须是常量表达式，它们分别表示各维的大小。数组元素的个数为各维表达式的乘积。

3.1.4 字符串

1. 字符串的特征

(1) 字符串是由一个有效字符序列组成的数据结构，在 C++ 源程序中以一对双引号表示其起止界限。

(2) C++ 中的字符串可以包括字母、数字、专用字符、转义字符等，并且每个字符串都要以'\0'(空字或 NULL) 为结束符。

(3) C++ 将一个字符串存储在一维字符型数组中，并且每个字符串都要以'\0'(空字或 NULL) 为结束符。

2. 字符串的指针形式

由于指针与一维数组的对应性，字符串也可以用指向字符的指针形式表示。表 3.1 为两种字符串形式的比较。

3.1.5 枚举

1. 枚举类型的特征

enum(枚举)类型是一组被命名的整型常数的集合，并且这些常数可以有自己的名字。

2. 枚举类型的定义形式

定义 enum 类型的基本格式为

enum 枚举类型名 { 枚举成员表列 };

表 3.1　两种字符串形式的比较

项　　目	字符数组形式	字符指针形式
本质	数组，每个元素存放一个字符	指针变量：存放字符串首地址
定义格式	char 字符数组名[ ] = "字符串"； 如 char s[20] = "string"；	char * 字符指针变量 = "字符串"； 如 char *s；
访问字符	使用下标变量直接访问	使用间接引用
输入与输出方式	1. 整体：数组名方式：cin >> s； cout << s； 2. 个体：(1) 下标变量方式 (2) 间接引用方式	1. 整体：指针方式：cin >> s； cout << s； 2. 个体：(1) 类下标变量方式 (2) 间接引用方式
赋值方式	不允许赋值，只能进行字符串拷贝 如 strcpy(s,"string")；	允许赋值：s="string"；也可以用字符串拷贝函数 如 s=new char[7]；strcpy(s,"string")；
运算限制	只可以加一个小整数	字符指针可以进行赋值、减，并可以加一个小整数

3. 枚举类型变量的声明

形式 1：

enum　枚举类型名　　变量名表列；

形式 2：

enum 枚举类型名 { 枚举成员表列 } 变量名表列；

4. 枚举变量的初始化

C++ 中枚举常量的类型隐含地规定为 unsigned char 或 int 类型，并且第一个枚举常量的值为 0，以后依次增 1。C++ 也允许程序员显式地在定义中指定部分或全部枚举常量的值。

5. 枚举变量的运算

枚举变量可以进行赋值（仅限于所在类型的枚举常量）或关系运算。

3.1.6　共用体

1. 共用体的概念

共用体是由不同数据类型组成的复合数据类型，它的成员共享一段内存空间。

2. 共用体的定义

共用体定义的一般形式为：

```
union 共用体名
{
    成员1定义;
    成员2定义;
    ...
    成员n定义;
};
```

3. 共用体变量的生成方法

(1) 用共用体类型名说明;
(2) 定义共用体类型的同时生成变量;
(3) 直接定义变量。

4. 共用体成员的引用

(1) 用成员运算符直接引用;
(2) 使用指向运算符引用;
(3) 用成员运算符间接引用。

3.2 例题示范

例 3.2.1 二分查询:在 N 个已排序(设为从小到大)的数据(数或字符串)中,查询某一个数据,如果找到了,就指出其在 N 个数据中的位置;否则给出无该数据的信息(如"-1")。

解:

1. 类界面设计

```
class BinSearch
{
private:
    double *Plist;
public:
    BinSearch(double *,int);
    void BinSrch(int ,int ,double );
    ~BinSearch();
};
```

2. 构造函数和析构函数设计

```
BinSearch:: BinSearch(double *list,int ListLong)
{
    Plist=new double[ListLong];
```

```
    for(int i=0;i<ListLong;i++)
        Plist[i]=list[i];
}
BinSearch::~BinSearch()
{
    delete []Plist;
}
```

3. 二分查询成员函数设计

当问题的规模较大时，一种行之有效的方法是，将问题分解为同类型的规模较小的问题，递归地进行求解。

采用二分法求解本问题的基本思路是：如图 3.3 所示，设数列为 $a_1, a_2, \cdots, a_n$，被查找的数据为 x，则查找首先对 $a_m(m=(1+n)/2)$进行，于是得到如下 3 种情形：

(1) 若 $x>a_m$，则 x 可能在区间$[a_{m+1}, a_n]$中；

(2) 若 $x<a_m$，则 x 可能在区间$[a_1, a_{m-1}]$中；

(3) 若 $x=a_m$，则以 a_m 为所查找的数据 x，求解结束。

这样，当一次搜索没有找到解时，便把搜索范围缩小一半，可以再在新的搜索范围内按上述原则进行搜索，如此不断二分，直到得出结论(找到或找不到)为止。

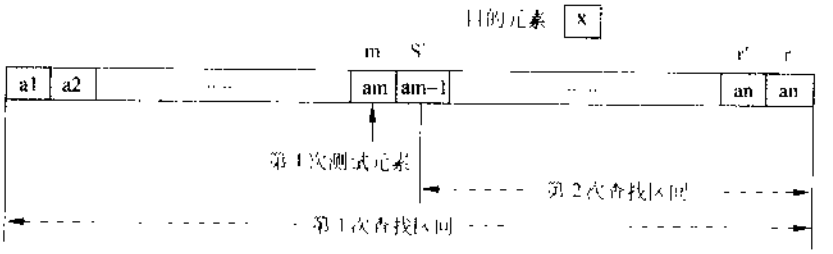

图 3.3 二分查找

下面来描述这个问题的递归算法：将原问题用函数 Binsrch()来描述：

binSrch(s,r,x)

其中，s 为数列的起始元素序号(开始时为 1)，r 为数列的终止元素序号(开始时为 n)，即数列为 $a_s, a_{s+1}, \cdots, a_r$，x 为要查找的元素。于是可以粗略地将 BinSrch(s,r,x)描述为：

```
BinSrch(s,r,x)
{
    m = (s + r)/2;
    if(x == am)    打印找到信息后返回;
    else if(s >= r)  打印找不到信息,结束程序执行;
    else if(x > am)    调用函数 binsrch(m + 1,r,x);
    else
        调用函数 binSrch(s,m - 1,x);
}
```

显然，这是一个递归函数。这里，if…else if 是 C 语言的一种特殊分支结构，它允许用一系列的 else if 语句，排列一组条件互斥的操作。由于前面 3 个条件为 x = = am，s >= r，x > am，所以最后一个 else 就相当于 else if (x < am)。

进一步细化得到如下二分查询函数：

```
#include <iostream.h>
#include <stdlib.h>
void BinSearch::BinSrch(int s,int r,double x)
{
    int m;
    m = (s + r)/2;
    if(Plist[m] == x)
    {
        cout << "The order of" << x << "is" << m+1 << "in this sequence.";
        return;
    }
    else if(s >= r)
    {
        cout << x <<"is not exist in the seguense.";
        exit(-1);
    }
    else if(x > Plist[m])
        BinSrch(m + 1, r, x);                    // 递归调用
    else
        BinSrch(s, m - 1, x);                    // 递归调用
}
```

4. 测试主函数设计

```
void main()
{
    double a[]={3.1,3.3,3.5,3.7,3.9,
                2.1,2.3,2.5,2.7,2.9};            // 数组(模拟表)声明及初始化
    double x = 2.3;
    int s = 0, r = 9;
    BinSearch bs(a,10);                          // 对象声明及初始化
    bs.BinSrch(s,r,x);                           // 调用成员函数
}
```

例 3.2.2 快速排序。快速排序(quick sort)是由著名计算机科学家 C. A. R. Hoare 根据分治策略给出的一种高效率的排序方法。它的基本思想是：在待排序序列中选定一个元素 X(可以任选，也可以选第一个元素)，称为划分元素，用它将原序列整理一——划分(partitioning)为两个子序列，使其右边的元素不比它小，使其左边的元素不比它大(小)；然后再对两个子序列进行上述划分；经过不断划分，直至将原序列整理完毕为止。图 3.4 为快

速排序示意图。

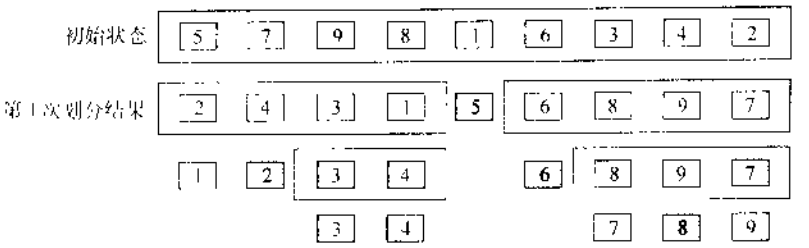

图 3.4　快速排序

解：

1. 类界面设计

```
class QuickSort
{
private:
    int low,high;
public:
    QuickSort( int , int );
    void qkSort();
    int part();
    void prnt();
};
```

2. 构造函数设计

```
QuickSort::QuickSort( int l , int h )
{
    low = l;
    high = h;
}
```

3. 快速排序成员函数设计

(1) 问题分析

一个快速排序算法很容易描述为：

```
qkSort(int m,int n)     // 序列由 m 到 n
{
    if(m < n)
    {
        调用划分函数 part(m,n,i);
        递归调用 qkSort(m, i - 1);
```

```
        递归调用 qkSort(i + 1, m);
    }
    else
    {
        输出排序结束信息
        return;
    }
}
```

下面进一步考虑如何描述函数 part()。基本思路如下：

步骤 1：准备。

步骤 1.1：设置两个指针 i 和 j，分别指向待划分序列的首尾元素 a[m]和 a[n]；

步骤 1.2：选取划分元素 p=a[k](k 属于[m,n])，将 a[m]移至 k，即令 a[k]=a[m]。

步骤 2：划分，当 i!=j 时，反复执行下面的操作(见图 3.5)。

步骤 2.1：逐步减小 j，将 a[j]与 p 比较，若 a[j]>p，则令 j--(j 左移一个数据位)；否则，令 a[i]=a[j](将 a[j]放在 i 位置)；

步骤 2.2：逐步增加 i，将 a[i]与 p 比较，若 a[i]<p，则令 i++(i 右移一个数据位)；否则，令 a[j]=a[i](将 a[i]放在 j 位置)。

步骤 3：放置划分元素，即令 a[i]=p。

根据上述分析，可以写出划分函数如下：

```
int part(int low,int high)
{
    int i,j;
    float p;
    i = low; j = high;                  // 准备
    P = a[low];
    while(i < j)
    {
        while(i < j && a[j] >= p)   // 逐步减小 j
            j--;
        a[i] = a[j];
        while(i < j && a[i] <= p)   // 逐步增大 i
            i++;
        a[j] = a[i];
    }
    a[i] = p;                           // 放置划分元素
    return(i);
}
```

(2) 快速排序成员函数

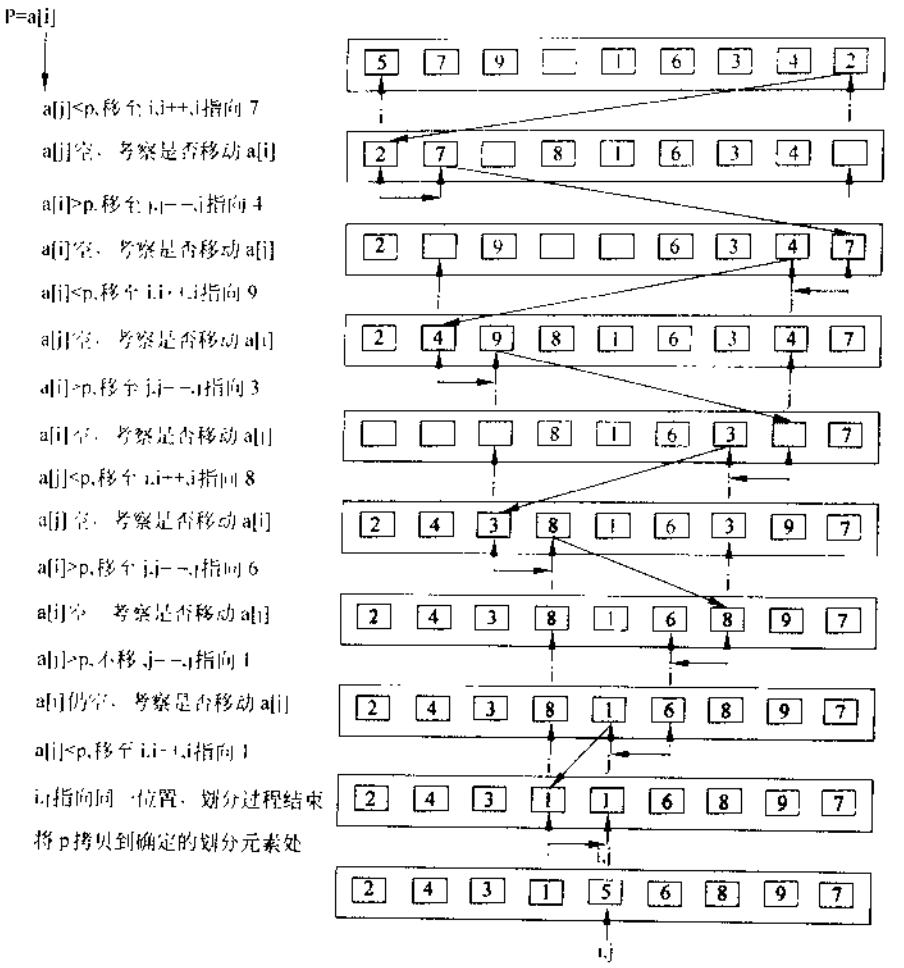

图 3.5　一次划分过程

```
void QuickSort::qkSort()
{
    int i;
    if( low < high )
    {
        i = part();
        high = i - 1;
        qkSort();
        low = i + 1;
        qkSort();
    }
    else
        return;
}
```

```
int QuickSort::part()
{
    int i,j;
    double p;
    i = low; j = high;
    p = a[low];
    while(i < j)
    {
        while(i < j && a[j] >= p)
            j--;
        a[i] = a[j];
        while(i < j && a[i] <= p)
            i++;
        a[j] = a[i];
    }
    a[i] = p;
    return(i);
}
```

4. 输出打印成员函数

```
#include <iostream.h>
void QuickSort::prnt()
{
    cout << "\nsorted sequence is:\n";
    for(int i = 0; i <= N - 1; i++)
        cout << a[i] << endl;
}
```

5. 测试主函数

```
#define N 10
double a[] = {2.1,3.7,3.5,2.9,2.7,2.3,3.3,2.5,3.1,3.9};
void main()
{
    int m = 0, n = N-1;
    QuickSort qs(m,n);
    qs.qkSort();
    qs.prnt();
}
```

例 3.2.3 堆栈。堆栈是一种"FILO"(先进后出)或"LIFO"(后进先出)的存储实体。如图 3.6 所示,它占有一片连续的存储单元,有两个端点:一个端点是固定的,称为栈底;一

个端点是活动的，称为栈顶。操作只能在栈顶进行。建立一个栈先要开辟堆栈空间，为了指示栈顶位置还要设置一个指针，称为栈顶指针(图 3.6 中的 sp)。

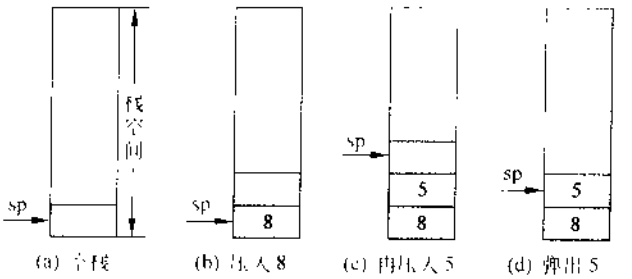

图 3.6 堆栈

栈有两种操作：push(压入)与 pop(弹出)。初建栈时，栈顶指针 sp 指向栈底，如图 3.6(a)所示。向栈内压入一个元素，如 8 时，先向栈顶写入 8，再执行 sp = sp + 1，如图 3.6(b)所示；再压入一个元素 5 时，先向栈顶写入 5，再执行 sp = sp + 1，如图 3.6(c)所示。总之，栈顶是用于存放下一个元素的。由栈内弹出一个元素 5 时，执行 sp = sp - 1，并可返回 sp 所指内容，如图 3.6(d)所示。当再压入新的元素时，将把 5 覆盖。也可以设置另外一种栈，使栈顶指针一直指向最后压入的元素。

解：

1. 类 myStack 的界面

由栈模型可知，栈应有如下两种成员数据：

buffer [MYSTACKSIZE]　(栈空间)

* sp　(栈顶指针)

还应有两个成员函数

push()　(压入操作)

pop()　(弹出操作)

考虑所建栈用于存储 long 型数据，则可以得到如下栈类界面：

```
class myStack
{
        long buffer [STACKSIZE];   // 开辟栈区
        long * sp;                 // 设置栈顶指针
    public:
        myStack();                 // 构造函数
        ~ MyStack();               // 释放函数
        void push(long);           // 压入数据
        long pop();                // 弹出数据
};
```

2. myStack 类的实现

(1) 构造函数

构造函数用以对类的初始化。栈在开始使用之前，是个空栈，栈顶指针应指向栈底，即指向数组的起始地址，所以构造函数应定义为

```
MyStack::MyStack()
{
    sp=buffer; //buffer 为数组起始地址
}
```

(2) 释放函数

释放函数用以撤销栈，程序中对它没什么特别要求。故可定义为：

```
MyStack::~MyStack()
{ }
```

(3) 压入函数 push()

对栈不能进行压入操作的条件是栈满，即

```
sp>=buffer+MYSTACKSIZE
```

具体的压栈操作分两步：

① 将(由参数传递来的)数据 data 存入栈顶指针所指向的单元，即

```
*sp=data;
```

② 修改栈顶指针，让栈顶指针指向下一个单元，以便下一次把数据存在那里，即

```
sp++;
```

以上两步用 C++ 可以描述为

```
* sp++=data;
```

于是可以得到如下压入函数：

```
void MyStack::push(long data)
{
    if(sp>=buffer+MYSTACKSIZE)
        cerr<<"Stack overflow! \n";
    else {
        * sp++=data;
        cout<<data<<"is pushed. \n";
    }
}
```

其中，cerr 是一个与标准错误输出设备相关联的流，它不能进行重定向，而 cout 可以进行重

定向。当程序中把标准输出设备重定向为其他设备，如文件时，用 cerr 仍能显示程序运行中的一些错误信息。

(1) 弹出函数 pop()

不能进行弹出操作的条件是栈空，即

```
sp <= buffer
```

具体的弹出操作是与压入相反的两步：

① 修改栈顶指针，使其指向前一个位置，即

```
--sp;
```

② 向外部返回栈顶元素值，即

```
return *sp;
```

用 C++ 可以把上述两步合并为

```
return *--sp;  //先自减 1,后引用
```

于是可以得到弹出函数的定义如下：

```
long MyStack::pop()
{
    if(sp<=buffer){
        cerr<<"MyStack is empty!";
        return 0;
    }
    return *--sp;
}
```

整理上述分析可以得到如下 MyStack 类的接口（文件 myStack.hpp）及其实现（文件 myStack.cpp）：

```
#define MYSTACKSIZE  10
class MyStack
{
        long buffer[MYSTACKSIZE];          //开辟栈区
        long *sp;                          //设置栈顶指针
    public:
                MyStack()
                {
                        sp=buffer;
                }
              ~MyStack(){}                 // 释放函数
```

```
        void push(long);                    // 压入数据
        long pop();                         // 弹出数据
};


#include <iostream.h>
#include "myStack.hpp"

void MyStack::push(long data)
{
        if(sp>=buffer+MYSTACKSIZE)
                cerr<<"Stack overflow! \n";
        else
        {
                *sp++=data;
                cout<<data<<"is pushed. \n";
        }
}

long MyStack::pop()
{
        if(sp<=buffer)
        {
                cerr<<"MyStack is empty!";
                return 0;
        }
        return *--sp;
}
```

3. 测试程序

依次压入下面的电话号码：351-7075461-3225，并依次弹出，可以使用下面的程序。

```
#include  <iostream.h>
#include  "myStack.hpp"
int main()
{
    MyStack a;

            a.push(351);
            a.push(7075461);
            a.push(3225);

            cout<<endl;
```

```
        cout<<a.pop()<<" is popped.\n";
        cout<<a.pop()<<" is popped.\n";
        cout<<a.pop()<<" is popped.\n";
        cout<<a.pop()<<" is popped.\n";
}
```

4. 测试结果

```
351 is pushed.
7075461 is pushed.
3225 is pished.

3225 is popped.
7075461 is popped.
351 is popped.
MyStack is empty!
0 is popped.
```

从测试结果可以看出栈的"FILO"机理。

例 3.2.4 链表。链表中的每个结点对象至少要有两部分成员:数据与指针。指针用以指向与其逻辑相邻、但存储位置不一定相邻的结点。当指向的对象类型都相同时,形成同质链表;否则形成异质链表。图 3.7 为由 3 个 Person 类对象互相链接形成同质链表的示意图。

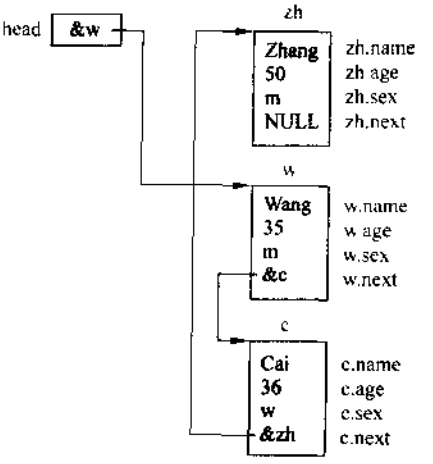

图 3.7 3 个 Person 对象形成的链表

上面的链接,形成 w→c→zh 的线性关系。其中,每个结点最多只有一个前驱及一个后继结点;首结点只有一个后继结点,没有前驱结点;最终结点只有一个前驱结点没有后继结点;中间的每个结点有一个前驱和一个后继结点。

解:

为了创建上述对象链表，应做两件事：

(1) 在每个对象中增加一个指针 next，以指向逻辑上的下一个对象。

(2) 设置一个头指针，以指向逻辑上的第一个对象。将逻辑上的最后一个对象中的 next 指针置空(NULL)，以表示后面不再指向任何对象。

1. 类界面的设计

```
#include <string.h>

class Person{
        char * name,sex;
        int age;
        Person * next;                          // 指向下一个对象结点的指针
    public:
        Person( char * n,int a,char s,Person * p = NULL)
        {
            name=new char[strlen(n)+1];// 立动态存储
            strcpy(name,n);
            age=a;
            sex=s;
            next=p;
        }
        ~Person(){delete name;}

        void disp();
};
```

2. 成员函数 disp 的实现

成员函数 disp 的功能是输出各结点的数据，其定义如下：

```
#include <iostream.h>
#include "person5.hpp"

void Person::disp()
{
    Person * ps;

    cout<<"Head->"
        <<name<<"-"                     // 处理第一个结点数据
        <<age<<"-"
        <<sex<<"->";

    if(next)                            // if(next != NULL)
```

```
    {                                   // 从第二个结点开始处理，直到 ps=NULL
        for(ps=next;ps;ps=ps->next)
        {
            cout<<ps->name<<"-"
                <<ps->age<<"-"
                <<ps->sex<<"->";
        }
    }
    cout<<"END\n";
}
```

disp()函数也可以定义成如下递归函数

```
void Person::disp()
{
    cout<<name<<":"<<age<<":"<<sex<<"->";

    if(next)
        next->disp();
}
```

3. 测试主函数及其测试结果

用下面的程序建立对象链表，并调用 disp()函数输出链表数据。

```
#include <iostream.h>
#include "person5.hpp"

void main(void)
{
    Person zh("Zhang",50,'m',NULL);     // 尾结点对象


    Person c("Cai",35,'w',&zh);
    Person w("Wang",34,'m',&c);
    Person *head=&w;                    // 头指针指向 w 对象

    head->disp();
}
```

输出结果如下：

```
Head->Wang-34-m->Cai-35-w->Zhang-50-m->END
```

4. 讨论

(1) 使用语句

```
w.disp();
```

也会输出同样结果。

(2) 在链表中增加或删除一个结点只须修改一个结点指针，如在结点 w 与 c 之间加入一个结点 l

```
Person l("Li",48,'w',&c);
```

时，只须修改 w 为

```
Person w("Wang",34,'m',&l);
```

这样的操作，对数组，尤其是对大数组来说，是非常麻烦的。请读者分析一下在一个数组中插入一个元素时的情形。

3.3 自我测试

3.3.1 自测习题

题 3.3.1 写出下面的数组声明语句中所定义的数组的数组名、类型、体积以及初始化情形。

(1) int x[4]={1,2,3,4};

(2) float y[3][4]={3.1,3.2,3.3,3.4,3.5,3.6,3.7,3.8};

(3) int a[2][3]={{ ,{4,5,6}},b[2][2][2]={{{ ,3,4},{ ,7,8}}};

(4) int a[3][3]={{1,2},{},{4,5,6}};

(5) char c1[10]={"Happy."},c2[ ][3]={{"*"},{"**"},{"*"}};

题 3.3.2 阅读下面的程序，指出输出结果。

(1)

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
    int i;
    int a[ ]={9,7,5,3,1};
    int b[ ]={-2,-4,-6,-8,-10};
    int c[5];
```

```
    for(i=0;i<5;i++)
        c[i]=a[i] | b[i];
    for(i=0;i<5;i++)
        cout << "%5d",c[i]);// cout <<setw(5)<<c[i];
}
```

(2)

```
#include <iostream.h>
void main()
{
int add(int m,int n,int arr[]);
    int total,a[3][4]={5,3,6,8,-2,-4,-7,9,1,0,7,2};
    total=add(3,4,a[0]);
    cout << "total= "<<total;
}

int add(int m,int n,int arr[ ])
{
    int i,j,sum=0;
    for(i = 0; i < m; i = i + m - 1)
        for(j = 0; j < n; j ++)
            sum + = arr[ i * n + j];

    for(j = 0; j < n; j = j + n - 1)
        for(i = I; i < m - 1; I ++)
            sum += arr[ i * n + j];
    return (sum);
}
```

(3)

```
#include <iostream.h>
#include <iomanip.h>
#define N 9
void main()
{
    int i,j,k,l,a[N][N];
for(i=0;i<N;i++)
        for(j=0;j<N;j++)
            a[i][j]=0;
    for(i=1;i<=(N+1)/2;i++)
        for(j=1;j<=N-i+1;j++)
            for(k=1;k<=N-i+1;k++)
```

```
                a[i][k]=i;
    for(k=1;k<=N;k++){
        for(l=1;l<=N;l++)
            cout <<setw(2)<<a[k][l];
        cout << "\n";
    }
}
```

(4)

```
#include <iostream.h>
void main()
{
    char    *pc=NULL;
    int     *pi=NULL;
    double  *pd=NULL;
    cout <<(int)(pc+1)<<(int)(pi+1)<<(int)(pd+1)<<endl
            <<(int)(pc+3)<<(int)(pi+5)<<(int)(pd+7)<<endl;
}
```

(5)

```
#include <iostream.h>
void fun(int b[]);
void main()
{
    char s[]="this is a book";
    char * p="this is a book";
    int    a[3];
    double d[5];
    cout << "\nsizeof(s)="<< sizeof(s)<<"\nsizeof(p)="<< sizeof(p)
         <<"\nsizeof(a)="<< sizeof(a)<<"\nsizeof(d)="<< sizeof(d)<<endl;
    fun(a);
}
void fun(int b[])
{
    printf("\nIn fun():sizeof(a)=%d",sizeof(b));
}
```

(6)

```
#include <iostream.h>
char *table[]={"first","second","third","fourth"};
char **pt[]={table+2,table,table+1};
char ***ppt[]={pt+2,pt,pt+1};
```

```
void main()
{
    cout << "\n "<< * table;
    cout << "\n "<< *** ppt;
    cout << "\n "<< * (table+1);
    cout << "\n "<< * table+1;
    cout << "\n "<< ** ppt[1]+2;
    cout << "\n "<< * ( * pt+1);
}
```

题 3.3.3 指出下面程序的功能，试输入任一整数，看程序将输出什么。

```
#include <iostream.h>
void main()
{
    char *[][10]=  {"","I","II","III","IV","V","VI","VII","VIII","IX",
                    "","X","XX","XXX","XL","L","LX","LXX","LXXX","XCC",
                    "","C","CC","CCC","CD","D","DC","DCC","DCCC","CM"};
    int n,t,i,m;
    cout << "Please enter number:";
    cin >> n;
    cout << n <<"=";
    for(m=0,i=1000;m<3;m++,i/=10)
    {
        t=(n%i)/(i/10);
        cout << a[2-m][t];
    }
    cout << "\n";
}
```

题 3.3.4 指出下面程序的功能，试任意一个英文星期的单词，看其输出什么。

```
#include <iostream.h>
char week_day[][10]={"sunday","Monday","Tuesday","Wednesday",
                     "Thursday","Friday","Saturday" };
void main()
{
    int i,j;
    char id[10],c;
    cout << "Enter a string:";
    cin >>id;
    for (i=0;i<7;i++){
    for (j=0;(c=week_day[i][j])==id[j]&&c!='\0';j++);
```

```
    if (c=='\0')break;
    }
    f (i<7)cout<< id <<" is "<<i<<endl;
    lse cout << "Error! \n";
}
```

题 3.3.5 编写程序，找出整型数组 a[10]中最大和最小的数及它们在数组中所在的位置。

题 3.3.6 数组 a 的长度是 $m+n$，编写程序把数组的前 m 个元素和后 n 个元素交换。

题 3.3.7 设计一个用筛选法求 100 以内的所有素数的程序。

题 3.3.8 设计一个模拟洗扑克牌，并给 4 个人发牌的程序。

题 3.3.9 编写函数，求 $n\times n$ 的矩阵两对角线元素值的和。

题 3.3.10 编写程序，输出用户指定行数的杨辉三角形。

题 3.3.11 归并排序(merging sort)："归并"的含义是将两个或两个以上的有序表列合并为一个新的有序表列。归并排序就是把一个具有 n 个数据的序列看作 n 个有序的子序列，不断两两归并，最后形成一个有序序列。显然这是一种分治策略。

归并排序的关键是归并。下面介绍一种归并算法。

设序列 a(low,high)由两个有序子序列组成：a(low,mid)和 a(mid,high)。归并按如下步骤进行：

(1) 设置一个与序列 a 大小相等的数组 b。

(2) 用 3 个指针 i、j、k 分别指向 a(low,mid)、a(mid,high)和 b 的首部。

(3) 重复执行下列操作：如果 a[i]<=a[j]，则执行 b[k]=a[i++]；否则执行 b[k]=a[j++]，直至有一个子序列取空。

(4) 将未取空的子序列放入 b 的尾部。

(5) 将 b 复制到 a。

请设计一个归并排序的 C++ 语言程序。

题 3.3.12 编写一个函数 squeeze(s1,s2)，能从字符串 s1 中删去所有与字符串 s2 中相同的字符。

题 3.3.13 设计一个函数 itoh(n,s)，能将一个无符号整数 n 转换为十六进制串。

题 3.3.14 设计一个函数，实现两个字符串的连接。

题 3.3.15 输入若干行，打印长度大于 30 的行。

题 3.3.16 编写函数 strecpy(char s,char t,int n)，只将源字符串 s 开始的 n 个字符复制到目的字符串 t 中。

题 3.3.17 编写函数 strecmp(char s,char t,int n)，只对两个参数字符串开始的 n 个字符进行比较。

题 3.3.18 设计一个函数，用指针实现两个字符串的连接。

题 3.3.19 求转置矩阵：将一个 $A(3\times5)$的矩阵，按转置矩阵 $B(5\times3)$输出。

题 3.3.20 有一个 3×5 的矩阵，设计一个 C++ 程序，求其最大元素。

题 3.3.21 给定一个后面的日期，设计一个倒计时的程序(只计日数)。

题 3.3.22 请设计一个将月份数转换为英文月份名的程序。

题 3.3.23 编写一个 C++ 程序，建立一个包含班级学生姓名的字符串数组并进行排序后打印出来。

题 3.3.24 编写一个 C++ 程序，建立一个包含班级学生姓名的向量，然后在向量中查找某一个学生，若找到，则输出信息"找到此人！"；否则，输出信息"查无此人！"。

题 3.3.25 大花猫是捕捉老鼠的能手，每天要抓到不少老鼠。但它吃老鼠之前，先要老鼠列队报数，第一批吃掉报单数的；剩下的老鼠重新报数，第一批仍吃掉报单数的……最后剩下的一只老鼠可以被保留，与第二天抓来的老鼠一起重新排队报数。一连好几天，大花猫发现最后留下来的都是一只机灵的小白鼠。当大花猫问他，想什么办法每天都能留下来时，小白鼠说："每天排队以前，我都先数一数你抓到了多少只老鼠，然后，我站在一个合适的位置上"。问题是小白鼠每天应站在什么位置上才不被吃掉？请编程求解。

题 3.3.26 编写一个 C++ 程序，利用下堆栈技术实现仅含整型数、加法和乘法运算的逆波兰表达式求值。

3.3.2 参考答案

题 3.3.1 参考答案：

(1) 数组名：x

类型：整型

体积：4 个元素×2=8 个字节

初始化情形：x[0]=1，x[1]=2，x[2]=3，x[3]=4

(2) 数组名：y

类型：浮点型

体积：(3×4)个元素×4=48 个字节

初始化情形：y[0][0]=3.1，y[0][1]=3.2，y[0][2]=3.3，y[0][3]=3.4
y[1][0]=3.5，y[1][1]=3.6，y[1][2]=3.7，y[1][3]=3.8
y[2][0]=0，y[2][1]=0，y[2][2]=0，y[2][3]=0

(3) int a[2][3]={{},{4,5,6}},b[2][2][2]={{},{3,4},{},{7,8}};

数组名：a

类型：整型

体积：(2×3)个元素×2=12 个字节

初始化情形：a[0][0]=0，a[0][1]=0，a[0][2]=0
a[1][0]=4，a[1][1]=5，a[1][2]=6

数组名：b

类型：整型

体积：(2×2×2) 个元素×2=16 个字节

初始化情形：b[0][0][0]=0，b[0][0][1]=0
b[0][1][0]=3，b[0][1][1]=4
b[1][0][0]=0，b[1][0][1]=0
b[1][1][0]=7，b[1][1][1]=8

(4) 数组名:a

类型:整型

体积:(3×3)个元素×2=18 个字节

初始化情形:a[0][0]=1, a[0][1]=2, a[0][2]=0

a[1][0]=0, a[1][1]=0, a[1][2]=0

a[2][0]=4, a[2][1]=5, a[2][2]=6

(5) char c1[10]={"Happy."}, c2[][3]={{"*"},{"**"},{"*"}};

数组名:c1

类型:字符型

体积:10 个字节

初始化情形:a[0]='H',a[1]='a',a[2]='p',a[3]='p',a[4]='y',a[5]='\0'

a[6],a[7],a[8],a[9]内容不确定。

题 3.3.2 参考答案:

(1) 输出结果:

```
7  3  -1  -5  -9
```

(2) 输出结果:

```
36
```

(3) 输出结果:

```
1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 0
4 4 4 4 4 4 0 0
5 5 5 5 5 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

(4) 输出结果:

```
148
32056
```

(5) 输出结果:

```
sizeof(s)=15
sizeof(p)=4
sizeof(a)=12
sizeof(d)=40
In fun():sizeof(a)=4
```

(6) 输出结果：

```
first
second
second
irst
ird
fourth
```

题 3.3.3 参考答案：

解：其功能是把从键盘上输入的 3 位以内的阿拉伯数字转换成等值的罗马数字，并在终端上输出。下面是运行一次该程序的情况：

```
Please enter number:512
512=DXII
```

题 3.3.4 参考答案：

解：其功能是把英文星期的单词名称转换成相应数码表示后输出。例如 Sunday 其数码表示为 0，Monday 其数码表示为 1，等等。下面是运行一次该程序的情况：

```
Enter a string: Friday
Friday is 5
```

题 3.3.5 参考程序：

```
#include <iostream.h>
class MaxMin{
public:
    void MaxMinp(int *a,int n,int &max,int &min,int &maxp,int &minp)
    {
        int i;
        max=min=a[0];
         maxp=minp=0;
        for(i=1;i<n;i++)
       {
           if(a[i]>max)
          {
              max=a[i];
              maxp=i;
          }
          if(a[i]<min)
          {
              min=a[i];
              minp=i;
          }
```

```
        }
    }
};
void main()
{
    int a[10]={2,5,3,8,9,10,4,7,1,6};
    int x,y,i,j;
    MaxMin maxmin;
    maxmin.MaxMinp(a,10,x,y,i,j);
    cout<<"最大数："<<x<<"  最大数位置："<<i<<endl;
    cout<<"最小数："<<y<<"  最小数位置："<<j<<endl;
}
```

测试结果：

```
最大数：10 最大数位置：5
最小数：1  最小数位置：8
```

题 3.3.6 参考程序：

```
#include <iostream.h>
#include <iomanip.h>
class Swap{
public:
    vord mSwipn(int *a,int m,int n)
    {
        int temp;
        for(int i=1;i<=m;i++)
        {
            temp=a[0];
            for(int j=0;j<m+n-1;j++) a[j]=a[j+1];
            a[m+n-1]=temp;
        }
    }
};
void main()
{
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    Swap swap;
    swap.mSwipn(a,6,4);
    for(int i=0;i<10;i++)
        cout<<setw(4)<<a[i];
    cout<<endl;
}
```

测试结果：

7 8 9 10 1 2 3 4 5 6

题 3.3.7 参考程序：

```
#include <iomanip.h>
#define N 99
class PrimeNum{
public:
    void SeekPrime(int *a,int n)
    {
        int i,k;
        for(i=1;i<N;i++)
        {
            k=2;
            while(a[i]%k)k++;
            if(a[i]!=k)a[i]=0;
        }
    }
};
void main()
{
    int i,a[N];
    for(i=0;i<N;i++) a[i]=i+2;
    PrimeNum prime;
    prime.SeekPrime(a,N);
    for(i=0;i<N;i++)
    if(a[i]!=0)cout<<setw(3)<<a[i];
    cout<<endl;
}
```

测试结果：

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

题 3.3.8 参考程序：

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
int pk[] = { 501,502,
             101,102,103,104,105,106,107,108,109,110,111,112,113,
             201,202,203,204,205,206,207,208,209,210,211,212,213,
             301,302,303,304,305,306,307,308,309,310,311,312,313,
             401,402,403,404,405,406,407,408,409,410,411,412,413,
             0,0};
class PackDeal
```

```
{
    public:
        void    makePackOnce();
        void    deal(int *,int *,int *,int *);
};
void PackDeal::makePackOnce()
{
      int temp,r,times;
    cout << "\n 确定拟洗牌次数:";
    cin >> times;                                   // 输入拟洗牌次数
      cout << endl;
    for(int j =1; j <= times; j ++)
    {
        cout<<"次数:"<<j<<endl;
        for(int i = 0; i < 54; i ++)                // 洗一次牌
        {
            r =int(float(54-i) * rand()/RAND_MAX)+i;
            temp = pk[i];
            pk[i] = pk[r];
            pk[r] = temp;
            cout << pk[i] << ",";                   // 打印洗好的牌
            if(((i%14) == 0&&i!=0))cout<<endl;
        }
        cout<<endl;
    }
}
void  PackDeal::deal(int * p1,int * p2,int * p3,int * p4)
{
    for(int i=0;i<14;i ++)
    {
        p1[i]=pk[i*4+0];
        p2[i]=pk[i*4+1];
        p3[i]=pk[i*4+2];
        p4[i]=pk[i*4+3];
    }
}

int main()
{
    int pa[14],pb[14],pc[14],pd[14],i;
    PackDeal pkd;
    pkd.makePackOnce();
    pkd.deal(pa,pb,pc,pd);
    cout<<"4 人分发牌情况如下:"<<endl;
    for(i=0;i<14;i+ +) cout<<pa[i]<<",";
```

```
        cout<<endl;
        for(i=0;i<14;i++) cout<<pb[i]<<",";
        cout<<endl;
        for(i=0;i<14;i++)cout<<pc[i]<<",";
        cout<<endl;
        for(i=0;i<14;i++)cout<<pd[i]<<",";      cout<<endl;
    return 0;
    }
```

测试结果：

```
确定拟洗牌次数：1
次数：1
501,303,111,404,306,301,208,409,405,402,203,407,401,307,212,
201,205,502,209,210,413,308,211,204,202,207,313,108,403,
102,103,406,101,113,105,304,410,309,112,104,412,310,206,
213,106,311,302,408,312,305,110,107,411,109,
4 人分发牌情况如下：
501,306,405,401,205,413,202,403,101,410,412,106,312,411,
303,301,402,307,502,308,207,102,113,309,310,311,305,109,
111,208,203,212,209,211,313,103,105,112,206,302,110,0,
404,409,407,201,210,204,108,406,304,104,213,408,107,0,
```

题 3.3.9 参考函数：

```
#include <iostream.h>
#define N 4
void DiagonalSum(int (*a)[N],int n,int &x,int &y)
{
    x=y=0;
    for(int i=0;i<n;i++)
    {
        x+=a[i][i];
        y+=a[i][n-i-1];
    }
}
```

测试函数：

```
void main()
{
    int a[N][N]={1,2,3,2,
                 2,1,2,3,
                 3,2,1,4,
                 2,3,4,1};
    int x,y;
```

```
    DiagonalSum(a,4,x,y);
    cout<<"左对角线之和："<<x<<endl;
    cout<<"右对角线之和："<<y<<endl;
}
```

测试结果：

```
左对角线之和：4
右对角线之和：8
```

题 3.3.10 参考程序：

```
#include <iostream.h>
#define MAX 30
class CYanghui{
private:
    int *x[MAX];
public:
    CYanghui()
    {
        for(int i=0;i<MAX;i++)
            x[i]=NULL;
    }
    ~CYanghui()
    {
        for(int i=0;i<MAX&&x[i];i++)
            delete x[i];
    }
    void print(int n)
    {
        if(n>MAX)
        {
            cout<<"指定行数超过"<<MAX<<"行!"<<endl;
            return;
        }
        for(int i=0;i<n;i++)
        {
            x[i]=new int[i+1];
            for(int j=0;j<=i;j++)
            {
                if(j==i||j==0)
                    *(x[i]+j)=1;
                else
                    *(x[i]+j)=*(x[i-1]+j-1)+*(x[i-1]+j);
                cout<<*(x[i]+j)<<"  ";
            }
```

```
                cout<<endl;
            }
        }
    };
    void main()
    {
        int m;
        CYanghui yh;
        cout<<"输入指定行数:";
        cin>>m;
        yh.print(m);
    }
```

测试结果:

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10  10  5  1
1  6  15  20  15  6  1
1  7  21  35  35  21  7  1
1  8  28  56  70  56  28  8  1
1  9  36  84  126  126  84  36  9  1
```

题 3.3.11 参考程序:

```
#include <iostream.h>
#define NUM 10
#define low 0
class MSort
{
public:
    Sort(int *a,int n);
};
MSort::Sort(int *a,int n)
{
    int i,j,k,mid_high,*b=new int[n];
    for(i=0;i<n-1;i++)
    {
        mid_high=i+1;
        for(j=k=low;k<=mid_high;k++)
        {
            if(a[j]<=a[mid_high])
                b[k]=a[j++];
```

```
            else
            {
                b[k]=a[mid_high];
                for(   ;j<mid_high;j++) b[++k]=a[j];
                break;
            }
        }
        for(j=0;j<=mid_high;j++)a[j]=b[j];
    }
}

void main()
{
    int x[NUM]={3,6,10,4,1,8,9,5,2,7};
    MSort St;
    St.Sort(x,NUM);
    for(int i=0;i<NUM;i++)
        cout<<x[i]<<" ";
    cout<<endl;
}
```

测试结果：

```
1  2  3  4  5  6  7  8  9  10
```

题 3.3.12 参考函数：

```
#include <iostream.h>
#include "string.h"
void squeeze(char *s1,char *s2)
{
    char *s1_top=s1,*s2_top=s2;
    bool sq;
    while(*s1)
    {
        sq=false;
        while(*s2)
        {
            if(*s1==*s2)
            {
                *s1='\0';
                strcat(s1_top,s1+1);
                sq=true;
                break;
            }
            else s2++;
```

```
        }
        if(! sq)s1++;
        s2=s2_top;
    }
}
```

测试函数：

```
void main()
{
    char a[]="abcdefgghiijkl",b[]="bdglk";
    squeeze(a,b);
    cout<<a<<endl;
}
```

测试结果：

```
a c e f h i i j
```

题 3.3.13 参考函数：

```
#include <iostream.h>
#define LENGTH 20
void itoh(int n,char *s)
{
    char b_h[LENGTH];
    int i=0,remainder;
    while(n>0)
    {
        remainder=n%16;
        switch(remainder)
        {
            case 0:{b_h[i++]='0';break;}
            case 1:{b_h[i++]='1';break;}
            case 2:{b_h[i++]='2';break;}
            case 3:{b_h[i++]='3';break;}
            case 4:{b_h[i++]='4';break;}
            case 5:{b_h[i++]='5';break;}
            case 6:{b_h[i++]='6';break;}
            case 7:{b_h[i++]='7';break;}
            case 8:{b_h[i++]='8';break;}
            case 9:{b_h[i++]='9';break;}
            case 10:{b_h[i++]='A';break;}
            case 11:{b_h[i++]='B';break;}
            case 12:{b_h[i++]='C';break;}
            case 13:{b_h[i++]='D';break;}
            case 14:{b_h[i++]='E';break;}
```

```
            case 15:b_h[i++]='F';
        }
        n/=16;
    }
    b_h[i]='\0';
    for(int j=0;j<i;j++)
        s[j]=b_h[i-(j+1)];
    s[j]='\0';
}
```

测试函数：

```
void main()
{
    int n=567890;
    char h[LENGTH];
    itoh(n,h);
    cout<<"十进制数"<<n<<"对应的十六进制数是"<<h<<endl;
}
```

测试结果：

十进制数 567890 对应的十六进制数是 8AA52

题 3.3.14 参考函数：

```
#include <iostream.h>
#include "string.h"
void stringcat(char str1[],char str2[])
{
    int m,n;
    m=strlen(str1);
    n=strlen(str2);
    for(int i=m;i<m+n;i++)
        str1[i]=str2[i-m];
}
```

测试函数：

```
void main()
{
    char s1[20]="this is ",s2[10]="string";
    cout<<"字符串"<<s1;
    stringcat(s1,s2);
    cout<<"和字符串"<<s2<<"的连接结果是："<<s1<<endl;
}
```

测试结果：

字符串 this is 和字符串 string 的连接结果是：this is string

题 3.3.15 参考程序：

```
#include <iostream.h>
#include "string.h"
#define N 80
class Exc30Line{
private:
    char (*ps)[N];
    int Lines;
public:
    void InputStr()
    {
        cout<<"Line number:";
        cin>>Lines;
        ps=new char[Lines][N];
        for(int i=0;i<Lines;i++)
        {
            cout<<"Line "<<i+1<<":"<<endl;
            cin>>ps[i];
        }
    }
    void Outputstr()
    {
        cout<<"The lines are larger than 30: "<<endl;
        for(int i=0;i<Lines;i++)
        {
            if(strlen(ps[i])>30)
            {
                cout<<"Line "<<i+1<<":"<<endl;
                cout<<ps[i]<<endl;
            }
        }
    }
};
void main()
{
    Exc30Line exceed30;
    exceed30.InputStr();
    exceed30.Outputstr();
}
```

测试结果：

```
Line number:3
Line 1:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Line 2:
bbbbbbbbbbbbbbbbbbbbbb
Line 3:
cccccccccccccccccccccccccccccccccccccc
The lines are larger than 30:
Line 1:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Line 3:
cccccccccccccccccccccccccccccccccccccc
```

题 3.3.16 参考函数：

```
#include <iostream.h>
#include "string.h"
void strecat(char s[],char t[],int n)
{
    int m;
    m=strlen(t);
    for(int i=m;i<m+n;i++)
        t[i]=s[i-m];
}
```

测试函数：

```
#define N 6
void main()
{
    char s1[20]="this is a ",s2[20]="string1string2.";
    cout<<"字符串"<<s1;
    strecat(s2,s1,N);
    cout<<"同字符串"<<s2<<"的前"<<N<<"个字符连接结果是:"<<endl;
    cout<<s1<<endl;
}
```

测试结果：

```
字符串 this is a 同字符串 string1string2.的前 6 个字符连接结果是
this is a string
```

题 3.3.17 参考函数：

```
#include <iostream.h>
#include "string.h"
int strecmp(char *s,char *t,int n)
```

```
{
    for(int i=0; *s==*t&&i<n;i++,s++,t++);
        if(i==n)return(0);
    return (*s-*t);
}
```

测试函数实例验证：

```
#define N 6
void main()
{
    int t;
    char phrase[5];
    char *s1="abcdefg", *s2="abcdexyz";
    t=strecmp(s1,s2,N);
    if(t==0)
        strcpy(phrase,"等于");
    else if(t>0)
        strcpy(phrase,"大于");
    else
        strcpy(phrase,"小于");
  cout<<"字符串"<<s1<<"的前"<<N<<"个字符"<<phrase;
  cout<<"字符串"<<s2<<"的前"<<N<<"个字符"<<endl;
}
```

测试结果：

字符串 abcdefg 的前 6 个字符小于字符串 abcdexyz 的前 6 个字符

题 3.3.18 参考函数：

```
#include <iostream.h>
#include "string.h"
void strptrcat(char *str1,char *str2)
{
    str1+=strlen(str1);
    while(*str1++=*str2++);
}
```

测试函数：

```
void main()
{
    char s1[20]="this is ",s2[10]="string";
    cout<<"字符串"<<s1;
    strptrcat(s1,s2);
    cout<<"和字符串"<<s2<<"的连接结果是："<<s1<<endl;
}
```

测试结果：

字符串 this is a 和字符串 string 的连接结果是：this is a string

题 3.3.19 参考程序：

```
#include <iomanip.h>
#define M 3
#define N 5
class Matrix{
private:
    int (*b)[M];
public:
    void transpose(int (*a)[N])
    {
        for(int i=0;i<M;i++)
            for(int j=0;j<N;j++)
                b[j][i]=a[i][j];
    }
    void OutputB()
    {
        for(int i=0;i<N;i++)
        {
            for(int j=0;j<M;j++)
                cout<<setw(3)<<b[i][j]<<"  ";
            cout<<endl;
        }
    }
};
void main()
{
    int a[M][N]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};
    Matrix tran;
    tran.transpose(a);
    tran.OutputB();
}
```

测试结果：

```
1    6   11
2    7   12
3    8   13
4    9   14
5   10   15
```

题 3.3.20 参考程序：

```
#include <iomanip.h>
class MatrixMax{
public:
    int Max(int (*x)[5])
    {
        int max;
        max=x[0][0];
        for(int i=0;i<3;i++)
            for(int j=0;j<5;j++)
                if(x[i][j]>max)max=x[i][j];
        return max;
    }
};
void main()
{
    int a[3][5]={{11,12,21,10,30},{12,13,99,89,70},{80,98,76,45,55}};
    MatrixMax aMax;
    cout<<"The max is "<<aMax.Max(a)<<endl;
}
```

测试结果：

```
The max is 99
```

题 3.3.21 参考程序：

```
#include <iomanip.h>
class Date{
private:
    int daytable[2][13];
    int daysum;
public:
    Date();
    int DayNumber(int,int,int,int,int,int);
    void DownTiming();
};
Date::Date()
{
    daytable[0][0]=365;
    daytable[1][0]=366;
    for(int i=1;i<13;i++)
        if(i==2)
        {
            daytable[0][i]=28;
            daytable[1][i]=29;
        }
```

```
        else if(i==4||i==6||i==9||i==11)
            daytable[0][i]=daytable[1][i]=30;
        else
            daytable[0][i]=daytable[1][i]=31;
}
int Date::DayNumber(int nowy,int nowm,int nowd,int futy,int futm,int futd)
{
    int i,j,year,day1,day2,frontm,behindm;
    daysum=0;
    for(year=nowy;year<=futy;year++)
    {
        i=(year%4==0&&year%100||year%400==0);
        frontm=(year==nowy)? nowm:0;
        day1=(frontm==0)? 0:nowd;
        for(j=1;j<frontm;j++)
            day1+=daytable[i][j];
        behindm=(year==futy)? futm:13;
        if(behindm==13)
            day2=daytable[i][0];
        else
            for(j=1,day2=futd;j<behindm;j++)
                day2+=daytable[i][j];
        daysum+=day2-day1+1;
        return (daysum);
    }
}
void Date::DownTiming()
{
    for(int i=daysum;i>0;i--)
    {
        cout<<i<<endl;
        for(double j=0;j<500000;j+=0.01);
    }
}
void main()
{
    Date d;
    cout<<"天数:"<<d.DayNumber(2003,6,25,2003,7,4)<<endl;
    cout<<"倒计时:"<<endl;
    d.DownTiming();
}
```

测试结果：

天数：10
倒计时：
10
9
8
7
6
5
4
3
2
1

题 3.3.22 参考程序：

```
#include <iomanip.h>
#include "string.h"
class Month{
private:
    char *eName[13];
public:
    Month();
    void MonthName(int,char *);
};
Month::Month()
{
    eName[0]="illegal month";
    eName[1]="January";
    eName[2]="February";
    eName[3]="March";
    eName[4]="April";
    eName[5]="May";
    eName[6]="June";
    eName[7]="July";
    eName[8]="August";
    eName[9]="September";
    eName[10]="October";
    eName[11]="November";
    eName[12]="December";
}
void Month::MonthName(int n,char *name)
{
    if(n<1||n>12)
        strcpy(name,eName[0]);
    else
```

```
        strcpy(name,eName[n]);
}
void main()
{
    int n;
    char  moname[14];
    cout<<"Month number:";
    cin>>n;
    Month month;
    month.MonthName(n,moname);
    cout<<"Month name:"<<moname<<endl;
}
```

测试结果：

```
Month number:10
Month name: October
```

题 3.3.23 参考程序：

```
#include <iomanip.h>
#include "string.h"
#define N 20
class SortNames{
public:
    void sort(char (*name)[N],int M)   // 冒泡排序的方法
    {
        char temp[N];
        for(int i=0;i<M;i++)
        {
            for(int j=M-1;j>i;j--)
            {
                if(strcmp(name[j-1],name[j])>0)
                {
                    strcpy(temp,name[j]);
                    strcpy(name[j],name[j-1]);
                    strcpy(name[j-1],temp);
                }
            }
        }
    }
};
void main()
{
    char names[][N]={"Tom","John","Rose","Owen","Sun"};
    SortNames sortn;
```

```
    sortn.sort(names,5);
    cout<<"排序后的姓名:"<<endl;
    for(int i=0;i<5;i++)
        cout<<names[i]<<endl;
}
```

测试结果:

```
排序后的姓名:
John
Owen
Rose
Sun
Tom
```

题 3.3.24 参考程序:

```
#include "string.h"
#define N 20
class SearchName{
public:
    int findname(char (*Sset)[N],int M)
    {
        bool flag;
        char Sname[N];
        cout<<"查询的学生姓名:";
        cin>>Sname;
        flag=false;
        for(int i=0;i<M&&! flag;i++)
            if(strcmp(Sset[i],Sname)==0){flag=true;

        if(flag)
        {
            cout<<"找到此人!"<<endl;
            return i;
        }
        else
        {
            cout<<"查无此人!"<<endl;
            return -1;
        }
    }
};
void main()
{
    int ptn;
```

```
    char names[][N]={"Tom","John","Rose","Owen","Sun" ;
    SearchName sn;
    ptn=sn.findname(names,5);
    if(ptn>=0) cout<<"该学生所在位置:"<<ptn<<endl;
}
```

测试结果:

```
查询的学生姓名:Rose
找到此人!
该学生所在位置: 3
```

题 3.3.25 参考程序:

```
#include <iomanip.h>
class CatEatMouse{
private:
        int *     mouse,              // 指向老鼠列队的指针
                  p,                  // 指向当前报数的老鼠号码
                  q,                  // 指向前一个报过数的老鼠号码
                  count;              // 指向老鼠报数计数器
public:
        int CountAndEat(int m)
    {
        mouse=new int[m+1];
        for(int i=1;i<m;i++)          // 下标 i 代表每只老鼠的编号,a[i]中存放第 p
            mouse[i]=i+1;             // 号老鼠报过数后下一只该报数的老鼠的号码
        mouse[m]=1;                   // 最后一只老鼠 m 的数组元素 a[m]中存
                                      // 放的下一只该报数的老鼠号码是 1
        q=p=m;
        count=0;
        do
        {
            p=mouse[p];
            count++;                  // count 增 1 表示 p 号老鼠报了一个数
            if(count%2)
                q=p;
            else
                mouse[q]=mouse[p];    // p 号老鼠被吃掉,因它下一轮不能再参加报数
                                      // 应把 mouse[p]中的老鼠号码传给 mouse[q]
        }
        while (p!=mouse[p]);          // p!=mouse[p]表明列队中不再有其他编号的老鼠存在
        return (mouse[p]);            // 返回剩下的老鼠编号
    }
};
void main()
```

```
{
    int n;
    CatEatMouse cem;
    cout<<"猫捕捉到的老鼠个数:";
    cin>>n;
    cout<<"每天都能留下来的小白鼠所站的位置:"<<cem.CountAndEat(n)<<endl;
}
```

测试结果:

```
猫捕捉到的老鼠个数:8
每天都能留下来的小白鼠所站的位置:1
```

题 3.3.26 参考程序:

```
#include <iomanip.h>
#include "stdlib.h"
#define Maxsiz 10
class PolishExp{
private:
    int OpStack[Maxsiz];          // 定义下堆栈,存放运算对象、中间结果和最终结果
    int sp;                       // 逻辑上的栈指针
public:
    PolishExp();
    void expcal(char *);
    void push(int);
    int pop();
};

PolishExp::PolishExp()            // 初始化栈和栈指针
{
    sp=0;
    for(int i=0;i<Maxsiz;i++)
        OpStack[i]=0;
}

void
PolishExp::expcal(char *polish)   // 表达式计算函数
{
    int t,op1,op2;
    int i=0;
    while(polish[i])
    {
        t=0;
        switch(polish[i])
        {
```

```
            case '+':
                op1=pop();
                op2=pop();
                push(op1+op2);          // 加法中间结果压栈
                break;
            case '*':
                op1=pop();
                op2=pop();
                push(op1*op2);          // 乘法中间结果压栈
                break;
        default:
                while(polish[i]!=',')   //把字符型数字串转换成整数
                {
                    t=10*t+(polish[i]-'0');
                    i++;
                }
                push(t);                //运算对象压栈
        }
        i++;
    }
}
void PolishExp::push(int x)
{
    if(sp>=Maxsiz-1)
    {
        cerr<<"The stack is full!";
        exit(-1);
    }
    else
    {
        OpStack[sp++]=x;
    }
}

int PolishExp::pop()
{
    if(sp)
        return OpStack[--sp];
    else
        cout<<"The stack is empty";
    return -1;
}
void main()
{
    char *p="50,10,+2,*80,+";       //每个数后面加逗号“,”作为结束标记
```

```
    PolishExp pole;
    pole.expcal(p);
    cout<<"The result:"<<pole.pop()<<endl;      //运算结果保存在栈底
}
```

测试结果：

```
The result: 200
```

第4章 重载与模板

4.1 编程知识

4.1.1 函数名重载

1. 函数名重载与静态绑定

C++允许一个函数名对应多个函数体，形成函数名的多态性。这种多态性可以简单地概括为"一个接口，多种方法"。绑定(binding)又称联编、聚束，是指程序在编译或运行时将一个标识解释为一个相应的程序实体。

2. 运算符重载的成员函数方式和友元函数方式

用户定义类型的重载运算符，要求能访问运算对象的私有成员。为此只能用成员函数或友元函数两种形式定义运算符重载。表4.1为它们的函数形式。

表4.1 运算符重载的成员函数方式和友元函数方式

表达式	成员函数方式	友元函数方式
一元运算表达式：obj (@ (int)	obj.operator(@ (0)；	operator(@ (obj,0)
(@ obj	obj.operator(@ ( )	operator (@ (obj)
二元运算表达式：obj1 (@ obj2	obj1.operator(@ (obj2)；	operator (@ (obj1，obj2)

3. 运算符重载规则

(1) 不可以重载的运算符种类：除表4.2中的5个运算符外，几乎所有的C++运算符都可以重载。

(2) 类运算符重载与友元运算符重载方式：一般说来，运算符重载可以采用成员函数方式或友元函数方式。但表4.3所示的4种运算符只能采用成员函数方式。

表4.2 不重载的运算符

运算符	语义
.	成员运算符
*	指向成员的指针
::	作用域运算符
?:	条件运算符
sizeof	计算类型空间

表4.3 只能使用成员函数方式的运算符

运算符	语义
[ ]	下标运算符
=	赋值运算符
( )	函数调用运算符
→	用指针访问对象成员

(3) 一个重载的运算符函数不能带有默认参数。

(4) 被重载的运算符必须保持预定义的优先级与结合性。因为重载时不对运算的优先级与结合性进行定义。

(5) 应尽量使重载的运算符语义自然，可理解好。特别是下面的 3 个运算符与预定义的语义必须一致。它们是

. = &

无特殊情形，只引用某个对象的地址时，可以使用预定义的 &，而无需重载定义。

(6) 注意各运算符之间的联系。如 C++ 有如下几个与指针有关的运算符：

[] * & ->

等，它们之间有一种等价关系。因此，重载 operator -> () 也应重载 operator [] () 和 operator * ()，以维持这种等价关系。

4.1.2 模板

1. 类属

类属(genericity)也称类型参数化，它把类型定义为参数，以实现代码的可重用性。

2. 函数模板(template 函数)

函数模板真正实现了函数参数的通用性。作为一种代码重用机制，可以大幅度地提高程序设计的效率。

3. 类模板

类模板也称类属类或类发生器。对象是类的特例，类又可以看作类模板的特例。

(1) 类模板的定义

定义类模板时应注意以下几点：

① 定义一个类模板必须以关键词 template 开始，后面是尖括号括起的类型参表列，然后是类名：

```
template <class Type>
class 类名
{ … };
```

② 每一个类模板的成员函数的外部定义，必须以类型形参开始，此外还应加以函数返回值类型、类范围、函数名、形参序列，如：

```
template <class Type>
void array <Type>::operator=  (Type temp) {…};
```

还要注意，类名后面必须加上类型形参。类型形参可以有多个，如

```
template < class T1, class T2, …>
class classname          {…}
```

内嵌函数定义中的关键词 inline 是在类型形参的定义之后。

③ 类模板是类的抽象类型，它的实例化是生成一个具体的类。

(2) 类模板中的友元函数

在类模板中可以声明 3 种友元函数：

① 一般的友元函数，这种友元函数中不使用任何类模板中的类型参数表列中的参数；

② 封闭型类模板友元函数 (bound template friend function)；

③ 开放型类模板友元函数 (unbound template friend function)。

4.2 例题示范

例 4.2.1 下列规则说明了合法的函数重载：

规则 #1 函数要求不同的参数列表，而不同的返回值类型或者形参有无默认值不能作为函数重载的依据；

规则 #2 枚举类型可以区别于整型作为函数重载的依据，而通过 typedef 定义的类型别名则不能区别于其原来的类型，因而不能作为函数重载的依据；

规则 #3 如果某个实际参数不能与形参匹配，则系统会自动进行类型转换，以匹配最合适的函数，除非导致二义性编译器，将拒绝转换。

根据以上规则请指出以下各例中哪些是合法的函数重载，哪些不是合法的函数重载，如果不是，请指出其错误。

(1) 假定以下代码用于重载函数 f. <function 1>：

```
int f(int x,int y)
{
    return x * y;
}
<function 2>
double f(int x,int y)
{
    return x * y;
}
<function 3>
int f(int x=1,int y=7)
{
    return x+y+x * y;
}
```

(2) 下面 4 个定义重载函数 max：

```
<function 1>
int max(int x,int y)
{
    return x>y? x:y;
}
```

```
<function 2>
double max(double x,double y)
{
    return x>y? x:y;
}
<function 3>
int max(int x,int y,int z)
{
    int lmax=x;
    if(y>lmax)lmax=y;
    if(z>lmax)lmax=z;
    return lmax;
}
<function 4>
int max(void)
{
    int a,b;
    cin>>a>>b;
    return abs(a)>abs(b)? a:b;
}
```

(3) 以下关于函数 read 的 3 个版本，用于区分整型和枚举类型资料的输入：

```
<function 1>
void read(int&x)
{
    cin>>x;
}
<function 2>
enum Boolean {FALSE,TRUE};
void read(Boolean& x)
{
    char c;
    cin>>c;
    x=(c=='T')? TRUE:FALSE;
}
<function 3>
typedef int Logical;
const int TRUE=1,FALSE=0;
void read(Logical& x)
{
    char c;
    cin>>c;
    x=(c=='T')? TRUE:FALSE;
}
```

解：

(1)中的3个函数违反规则#1，函数1和函数2具有相同的参数列表，而函数3中的默认值不能认为是重载的依据。

(2)符合规则#1，参数列表不同，构成函数重载。

(3)符合规则#2，由于枚举类型被认为是不同于其他类型的类型，所以函数1和函数2构成函数重载；而函数3中的通过typedef定义的int类型别名Logical，编译器仍将其解释为int型，故和函数1的参数列表相同，函数3和函数1不构成函数重载。

例4.2.2 在math.h中有标准库函数sin的说明，其参数是以弧度(实数)表示的，为了能用角度(实数)进行计算，请编写程序重载sin函数，实现用角度(实数)进行计算sin函数值。

解：

1. 类设计

无论是弧度，还是角度，都是以实数表示，故不能用来区分函数重载，为此应该定义一个表示角度的类Degree，以这个类的对象作为sin函数的参数。

重载的sin函数最好不要设计成类Degree的成员函数，应设计为类Degree的友元函数，以实现调用形式上名字的独立性。

Degree类具体设计如下

```
#include <iostream.h>
#include <math.h>
class Degree
{
        double deg;
public:
        Degree(double x=0):deg(x)
        {}
        friend double sin(Degree& x)
        {
            return   sin(x.deg * 3.14159265/180) ;        //实参是实型，
            //在此调用系统库函数
        }
};
```

2. 测试重载sin函数设计

```
int main()
{
        int n;
        for(n=0;n<=180;n+=30)
        {
            Degree x(n);                                  //定义一个对象x,取值角度n
```

```
            cout<<"sin("<<n<<")=";
cout<<sin(x)<<endl;          //x 是对象类型,在此调用类 Degree 的友元函数
        }
        return 0;
}
```

3. 测试结果

```
sin(0)=0
sin(30)=0.5
sin(60)=0.866025
sin(90)=1
sin(120)=0.866025
sin(150)=0.5
sin(180)=3.58979e-009
```

例 4.2.3 在 VB 等语言中有乘方运算符^,而在 C++ 中该运算符为异或位运算符,但也可以通过运算符重载使其具有乘方运算的功能。试设计 C++ 程序实现运算符^乘方重载。

解:

1. 重载运算符设计

在 C++ 语言中运算符"^",系统预定义了异或位运算功能,重载它使之与系统预定义的"异或位"运算区分开,要求其运算对象类型同系统默认类型有异。用类对象作为运算符"^"的运算对象,用类的成员函数重载运算符"^"使之实现乘方运算。类和重载成员函数设计如下:

```
class Power{
        double x;
public:
        Double(double X=0.):x(X)
        {}
        double operator^(double y)
        {
            return pow(x,y);
        }
};
```

由于重载运算符"^"表达式中运算对象扩充了类对象,这样一来表达式中其他运算符也要同类对象结合,即它们的运算对象也出现了类对象。为解决表达式中其他运算符类对象运算问题,用友元函数重载其他运算符来解决这个问题。上述 Power 类扩充设计如下:

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

```
class CPower
{
        double x;
public:
        CPower(double X=0.):x(X)
        {}
        double operator ^(double y)
        {
            return pow(x,y);
        }
        friend CPower operator+(double a,CPower&b)
        {
            return a+b.x;
        }
        friend CPower operator-(double a,CPower&b)
        {
            return a-b.x;
        }
        friend CPower operator*(double a,CPower&b)
        {
            return a*b.x;
        }
        friend CPower operator/(double a,CPower&b)
        {
            return a/b.x;
        }

};
```

2. 测试主函数及其测试结果

```
int main()
{
        double a,X,y;
        cout<<"a,x,y =";
        cin>>a>>X>>y;
        CPower x(X);
        cout<<"a+x^y="<<(a+x^y)<<endl;
        cout<<"a-x^y="<<(a-x^y)<<endl;
        cout<<"a*x^y="<<(a*x^y)<<endl;
        cout<<"a/x^y="<<(a/x^y)<<endl;
        return 0;
}
```

测试结果如下：

```
    a,x,y =5 2 3
  a+x^y=343
  a-x^y=27
  a*x^y=1000
a/x^y=15.625
```

3. 结果分析

对上述测试结果进行分析，会发现，在 a+x^y 等表达式中，先执行+运算符的运算，后执行^运算。虽然在数学中乘方运算的优先级高于加法等运算，但这儿使用的是重载的运算符，C++ 中重载之后运算符的优先级和结合性都不会改变，而运算符^原来是用于异或位运算的，其优先级低于运算符+、-、*和/。故表达式 a+x^y 等价于(a+x)^y，如果要先执行乘方运算则表达式应写为 a+(x^y)。

另外，在上面代码中整个表达式加了个括号，主要是考虑到^的优先级低于≪的优先级。

例 4.2.4 编写程序，通过重载运算符+、-，实现对有理数进行加法、减法的运算。

解：

有理数是一个可以化为一个分数的数，如 3/5、17/32、472/967 都是有理数，而$\sqrt{2}$是无理数。

由于 C++ 中无预定义的有理数，需要定义一个有理数类，分别存放分子和分母。有理数的加、减操作用重载运算符+、- 来实现。

两个有理数 a/b 和 c/d 相加或相减，可由下列算式来实现：

$$a/b+c/d=(a*d\pm b*c)/(b*d)$$

分子和分母分别存放：

分子 = a * d ± b * c

分母 = b * d

类中还应使用一个规范化函数，对有理数进行规范化处理，它的作用是使有理数约去公分母，使保存的有理数的分子和分母之间除 1 外没有公约数。

1. 有理数类 CRational 具体设计如下

```
#include "iostream.h"
#include "math.h"
class CRational{
        int numerator,denominator;
        void optimization();            //规范化有理数私有成员函数
public:
        CRational(int x=0,int y=1);
        friend CRational operator + (CRational,CRational);
        friend CRational operator - (CRational,CRational);
```

```
        void print();
};
CRational::CRational(int x,int y)
{
        numerator=x;
        denominator=y;
            optimization();                    //有理数规范化
}
CRational operator + (CRational n,CRational m)
{
        CRational * temp;
        temp=new CRational;
        temp ->numerator=n. numerator * m. denominator+
n. denominator * m. numerator;
        temp ->denominator=n. denominator * m. denominator;
        temp ->optimization();
        return * temp;
}

CRational operator - (CRational n,CRational m)
{
CRational * temp;
        temp=new CRational;
        temp ->numerator=n. numerator * m. denominator-
n. denominator * m. numerator;
        temp ->denominator=n. denominator * m. denominator;
        temp ->optimization();
        return * temp;
}

void CRational::print()
{
        cout<<numerator<<"/"<<denominator<<endl;
}
void CRational::optimization()
{
        int gcd;
        if(numerator==0)                       //若分子为0,则置分母为1后返回
        {
            denominator=1;
            return;
        }

gcd=abs(numerator)>abs(denominator)? abs(numerator):abs(denominator);
        if(gcd==0) return;
```

```
        for(int i=gcd;i>1;i--)      //找最大公约数
            if((numerator%i==0)&&(denominator%i==0)) break;
        numerator/=i;
        denominator/=i;
        if(numerator<0 && denominator<0)
        {
            numerator=abs(numerator);
        denominator=abs(denominator);
        }
        else if(numerator<0 || denominator<0)
        {
            numerator=-abs(numerator);
        denominator=abs(denominator);
        }
}
```

2. 测试主函数及其测试结果

```
void main()
{
        CRational r1(5,8),r2(8,16),r3;
        r1.print();
        r2.print();
        r3=r1+r2;
        r3.print();
        r3=r1-r2;
        r3.print();
}
```

测试结果如下：

```
5/8
1/2
9/8
1/8
```

例 4.2.5 实现一个模板函数。

```
template <class T>
int BinSearch(T A[], T key, int low, int high);
```

在数组 A 中对关键词 key 进行二分查找。数组 A 中的元素已经按升序排序。

解：

1. 模板函数

分析题中给出的模板声明，得知设计模板函数 BinSearch 的目的是，使用该函数是为了对任意类型的有序队列进行查找，找到返回所在位置。

具体模板函数 BinSearch 设计如下：

```
#include <iostream.h>
template <class T>
int BinSearch(T A[], T key, int low, int high)
{
        int mid;
        while(low<=high)
        {
            mid=(low+high)/2;
            if(A[mid]==key)
                return mid;
            else if(A[mid]<key)
                low=mid+1;
            else
                high=mid-1;
        }
        return -1;
}
```

2. 测试主函数和测试结果

```
void main()
{
        int intarr[]={10,20,30,40,50,60,70,80,90,100};
        cout<<"20 is located at "<<BinSearch(intarr,20,0,9)<<endl;
        cout<<"80 is located at "<<BinSearch(intarr,80,0,9)<<endl;

    double realarr[]={1.2,3.4,4.5,5.6,6.7,7.8,8.9,9.0};
        cout<<"5.6 is located at "<<BinSearch(realarr,5.6,0,7)<<endl;
        cout<<"9.0 is located at "<<BinSearch(realarr,9.0,0,7)<<endl;
        cout<<"0.9 is located at "<<BinSearch(realarr,0.9,0,7)<<endl;
}
```

测试结果如下：

```
20 is located at 1
80 is located at 7
5.6 is located at 3
9.0 is located at 7
0.9 is located at -1
```

例 4.2.6 定义一个类模板，实现对任意类型链表的插入、删除和查找等操作。

解：

1. 结点模板设计

为了节省存储空间，声明一个结构类型的模板，用来定义链表中的结点，它与类模板的用法相同。结构模板设计如下：

```
#include "iostream.h"
template <class T>
Struct Node{
        T value;
Node * next;
}
```

2. 类模板设计

类模板取结构模板同样的模板参数 T。在具体设计时应当注意：在每个模板定义之前都需要在前面加上模板声明 template <class T>；类模板和结构模板在使用时，必须在名字后面缀上模板参数<T>。类模板设计如下：

```
    template <class T>      //声明模板
class CList{                //定义类模板
        private:
        Node <T> * head;
        int  size;
    public:
        CList();
        bool insert(T);
        bool erase(T);
        int  search(T);
        void print();
        ~CList();
};
template <class T>
CList<T>::CList()
{
        head=0;
        size=0;
}
template <class T>
CList<T>::~CList()
{
        Node<T> * temp;
        Node<T> * p=head;
```

```
        while(p)
        {
            temp=p;
            p=p ->next;
            delete temp;
    }
}
template<class T>
bool CList<T>::insert(T x)
{
Node<T> * node=new Node<T>;
        if(node)
        {
            node ->value=x;
            node ->next=head;
            head=node;
            size++;
            return true;
        }
        return false;
}
template<class T>
bool CList<T>::erase(T x)
{
    Node<T> * temp;
        if(head ->value==x)
        {
            temp=head ->next;
            delete head;
            size--;
            head=temp;
            return true;
        }
        for(Node<T> * p=temp=head;p;temp=p,p=p ->next)
            if((p ->value==x)&&(p!=head))
            {
                temp ->next=p ->next;
                delete p;
                size--;
                return true;
            }
        return false;
}
template<class T>
```

```
int CList<T>::search(T x)
{
        int i=0;
    for(Node<T> * p=head;p;p=p ->next,i++)
            if(p ->value==x)
                return i+1;
            return 0;
}

template<class T>
void CList<T>::print()
{
        for(Node<T> * p=head;p;p=p ->next)
            cout<<p ->value<<" ";
        cout<<endl;
}
```

3. 测试类模板主函数和测试结果

```
void main()
{
    CList<int> intlist;
    for(int i=10;i<100;i+=10)intlist.insert(i);
        intlist.print();
        cout<<"50 在链表中的位置是:"<<intlist.search(50)<<endl;
        CList<double> doublelist;
        doublelist.insert(11.1);
        doublelist.insert(22.2);
        doublelist.insert(33.3);
        doublelist.insert(44.4);
        doublelist.insert(55.5);
    if(doublelist.erase(33.3))
            cout<<"33.3 在实型链表中且已经被删除"<<endl;
        doublelist.print();
        CList<char * > stringlist;
    stringlist.insert("list");
        stringlist.insert("chained");
        stringlist.insert("a");
    stringlist.insert("is");
        stringlist.insert("This");
        stringlist.print();
}
```

测试结果如下:

```
90 80 70 60 50 40 30 20 10
50 在链表中的位置是：5
33.3 在实型链表中且已经被删除
55.5 44.4 22.2 11.1
This is a chained list
```

4.3 自我测试

4.3.1 自测习题

题 4.3.1 指出以下每一组声明中哪些是合法的函数重载，哪些不是，如果不是，请指出其错误。

```
(1) int point(int, int);
    int point(int, float);
    int point(int, double);
(2) int get(const int);
    float get(const int);
    double get(const int);
(3) int flag(int, int);
    int flag(int &, int);
    int flag(int &, int &);
(4) void comp(int i, int j);
    void comp(int a,int b);
    void comp(int x,int y);
```

题 4.3.2 针对下列给出的各重载函数 fun：

```
// 1
int abc::fun(int x,int y)
{
    cout<<"fun_+"<<endl;
    return (x+y);
}
// 2
int abc::fun(int& x,int y=2)
{
    cout<<"fun_-"<<endl;
    return (x-y);
}
// 3
int abc::fun(int x,int& y)
{
```

```
    cout<<"fun_*"<<endl;
    return (x*y);
}
int abc::fun(int& x,int& y)
{
    cout<<"fun_/"<<endl;
    return (x/y);
}
```

指出以下各函数调用哪些是合法调用，哪些是不合法调用，如果不合法，请指出其错误。

(1) fun(5, 10)

(2) fun(i, j)

(3) fun(i)

(4) fun(i, 20)

题 4.3.3 设有下列重载函数 max：

```
//<function 1>
int max(int x,int y)
{
    return x>y? x:y;
}
//<function 2>
double max(double x,double y)
{
    return x>y? x:y;
}
//<function 3>
int max(int x,int y,int z)
{
    int lmax=x;
    if(y>lmax)lmax=y;
    if(z>lmax)lmax=z;
    return lmax;
}
//<function 4>
int max(void)
{
    int a,b;
    cin>>a>>b;
    return abs(a)>abs(b)? a:b;
}
```

使用上述的 max 重载函数并假定用户输入值为 m = - 29，n = 8。指出以下各种情况分别调用了哪个函数，以及返回值为多少。

(1)

```
cin >> m >> n;
max(m,n);
```

(2)

```
max( );
```

(3)

```
max(m, 10,30);
```

题 4.3.4 利用题 4.3.3 中的 max 函数，请指出以下各语句的输出是什么。

```
int a = 5,b = 99,c = 153;
int m, n;
double h1 = .01, h2 = .05;
long t = 3000, u = 70000, v = -100000;
cout<<"Maximum of a and b is "<<max(a,b);
cout<<"Maximum of a, b and c is "<<max(a,b,c);
cout<<"1.0+ max(h1,h2) = "<<1.0 + max(h1,h2);
cout<<"Maximum of t, u and v is "<<max(t,u,v);
```

题 4.3.5 请完成以下 max 函数的重载，分别用于求 2 个整数、2 个实数、2 个字符串的最大值。

```
#include <iostream.h>
#include <string.h>
//求整型数的最大值
int max(int x,int y)
{
        return (x>y? x:y);
}
//求实数的最大值
double max(double x,double y)
{
        return x>y? x:y;
}
//求字符串的最大值
  [        ①        ]
{
  [        ②        ]
}
int main()
{
    cout<<max(10,20)<<endl;
    cout<<max(1.56,1.24)<<endl;
    cout<<max("Candy","Rolla")<<endl;
    return 0;
}
```

题 4.3.6 以下函数是否可认为是重载函数，为什么？

(1)

```
enum E1{one, two};
enum E2{three, four};
int f(int x, E1 y);
int f(int x, E2 y);
```

(2)

```
typedef double scientific;
double f(double x);
scientific f(scientific x);
```

题 4.3.7 请实现以下重载函数的不同版本。

```
void swap(int& a, int& b);
void swap(float& x, float& y);
void swap(char * & s, char * & t);
```

题 4.3.8 在 math.h 中有标准库函数 cos 的说明，其参数是以弧度(实数)表示的，为了能用角度(实数)进行计算，请编写程序重载 cos 函数实现用角度(实数)进行计算 cos 函数值。

题 4.3.9 请说明使用成员函数和友元函数重载运算符的不同之处。

题 4.3.10 类 ModClass 仅有一个整型资料成员 dataval，其取值范围为 0.6。构造函数取任意正整数 v，将 v 除以 7 的余数赋值给 dataval。

dataval=v %7

加运算符实现两个对象的相加，将两个对象的资料成员的值相加并取除 7 后的余数。例如：

```
    ModClass  a(10);        //dataval in a is 3;
    ModClass  b(6);         //dataval in b is 6;
    ModClass  c;            //c= a + b has value (3+6)%7=2
class ModClass
{
private;
        int dataval;
public;
        ModClass(int v=0);
        ModClass operator+ (const ModClass& x);
        int GetValue(void) const;
};
```

(1) 实现以上类中的各成员函数。

(2) 以友元的形式声明并实现运算符 *，该运算符将两个对象的资料成员的值相乘并取除 7 后的余数。

(3) 写一个函数

```
ModClass Inverse(ModClass& x)
```

形参 x 的资料成员取非 0 值，该函数返回对象 y，使其满足 x * y=1(y 称做 x 的倒置)。(提示：将值为 1 到 6 的对象依次跟对象 x 相乘，其中之一就是 x 的倒置。)

(4) 为 ModClass 类重载流插入符(流输出)。

(5) 重载类型转换函数 operator int()，用于取代 GetValue。该运算符通过返回 dataval 的值将一个 ModClass 类型的对象转换为一个整数。

(6) 写一个函数

```
void Solve(ModClass a, ModClass& x, ModClass b);
```

用于求关于 x 的方程 $ax=b$。

题 4.3.11 定义复数类 complex，对 complex 类型的对象重载 +、-、*、/ 以及一元 - 运算符和插入运算符。要求采用友元和成员函数两种形式实现。

题 4.3.12 编写程序，通过重载运算符 *、/ 实现对有理数进行乘法、除法运算，并且可以将有理数转换为实数。

题 4.3.13 在练习 4.7(见教材)中定义了类 Degree，但为了能使用以下循环语句：

```
Degree n;
for(n=0;n<=180;n+=30)
```

还必须为该类重载运算符 += 和 <= 。请完成以下代码填充。另外，对于赋值语句 n=0；系统会自动调用构造函数将 0 转换为 Degree 类型的对象，因此没有必要显式转换。

```
#include <iostream.h>
#include <math.h>
class Degree
{
    double deg;
public:
    Degree(double x=0):deg(x)
    {}
    friend double sin(Degree& x)
    {
        return sin(x.deg * 3.14159265/180);
    }
    Degree& operator+=(const Degree& a)
    {
        ①
        return *this;
    }
    friend bool operator<=(const Degree& a,const Degree& b)
    {
```

```
            return [          ②          ];
        }
};
int main()
{
Degree n;
        for(n=0;n<=180;n+=30)
        {
cout<<sin(n)<<endl;
        }
        return 0;
}
```

题 4.3.14 以下代码定义了字符串类 String，请完善代码，实现 2 个字符串的连接。

```
#include <iostream.h>
#include <string.h>
class String
{
        char* str;
public:
        String(char* p=""){str=p;}
        friend ostream& operator<<(ostream& os,String& s)
        {
            os<<s.str<<endl;
            return os;
        }
    friend String operator+(const String&,const String&);
};
    [          ①          ]
{
        char* buf;
        int n;
        n=strlen(a.str)+strlen(b.str)+1;
        if( [          ②          ] )              //为连接后的字符串动态分配内存
        {
            strcpy(buf,a.str);
            strcat(buf,b.str);
            [          ③          ];               //返回连接后的字符串
        }
        else
            return a;                               //如果内存分配不成功，则返回第一个字符串
}
int main()
{
```

```
        String a("Lion"),b("Tiger"),c("Bear"),x,y;
        x=a+b;
        y=a+b+c;
        cout<<x<<y;
        return 0;
}
```

题 4.3.15

(1) 编写一个类属函数 Max,返回 2 个值中的较大者。

(2) 编写一个 Max 函数的重载版本,用于处理 C++ 字符串。以指向字符的指针作为参数,返回指向较大字符串的指针。

题 4.3.16 编写程序,重载运算符 +、-,对实数向量进行加法、减法运算,并且重载运算符[],用来返回向量元素的值。

题 4.3.17 编写程序,重载运算符 +、-,对实数矩阵进行加法、减法运算,并且重载运算符[],用来返回矩阵元素的值。

题 4.3.18 编写一个模板类 DataStore,该类具有以下一些成员函数:

int Insert(T elt);向具有 5 个 T 类型的元素的私有数组 dataElements 中插入元素 elt。数组中下一个可插入的位置由数据成员 loc 给出,loc 也是数组中数据的个数。如果数组中没有了剩余空间,则该函数返回 0。

int Find(T elt);在数组 dataElements 中寻找元素 elt,如果找到,返回该元素在数组中的下标,否则返回 -1。

int NumElts(void);返回数组 dataElements 中储存的元素的个数。

T& GetData(int n);返回数组中位置 n 处的元素,如果 n<0 或 n>4,则给出错误信息,并结束程序。

题 4.3.19 写一个函数

```
template<class T>
int Max(T Arr[], int n);
```

用于返回数组中最大值的下标。

题 4.3.20 编写下面的函数

```
template<class T>
void InsertOrder(T A[], int n, T elem);
```

用于往数组 A 中插入元素 elem,并使数组中元素保持升序。注:当找到插入点时,应使插入点后面的所有元素向后移动一个位置。

题 4.3.21 写一个模板函数 Copy

```
template<class T>
void Copy(T A[], T B[], int n);
```

该函数将 n 个元素从数组 B 复制到数组 A。写一个主程序用于测试 Copy。至少包括以下数组：

(1)

```
int Aint[6], Bint[6]={1,3,5,7,9,11};
```

(2)

```
struct Student
{
        int field1;
        double field2;
};
Student Astudent[3];
Student Bstudent[3]={{1,3.5},{3,0},{5,5.5}};
```

题 4.3.22 设有如下代码：

```
#include <iostream.h>
#include <string.h>
class T
{
char* p1;
public:
T(char* s1="");
~T();
    void print()
    {cout<<"p1="<<p1<<endl;}
};
T::T(char* s1)
{
p1=new char[strlen(s1)+1];
strcpy(p1,s1);
}
T::~T()
{
delete []p1;
}
void main()
{
T t1("good luck"),t2(t1),t3;
t1.print();
t2.print();
t3=t2=T("hi");
```

```
t2.print();
t3.print();
}
```

请问以上代码能否正常执行？如不能，在该类中添加哪些成员函数，写出添加的成员函数的实现，并写出最后的运行结果。

4.3.2 参考答案

题 4.3.1 参考答案：

(1)和(3)是合法的重载函数。

(2)和(4)是不合法的重载函数。

(2)的错误体现在，C++语言中函数重载是使用不同类型的参数或参数个数确定的，而不能使用函数返回值的类型。

(4)的错误体现在，函数重载与参数类型有关，而与参数名的相同和不同无关。

题 4.3.2 参考答案：

(1)和(3)是合法的函数重载。(1)调用第1个重载函数，(3)调用第1个重载函数。

(2)和(4)是不合法的函数重载。(2)第1、第2、第3和第4重载函数它都能够调用，而(3)能够调用第1和第2个重载函数。

题 4.3.3 参考答案：

(1) 调用函数 max(int,int)，返回 8

(2) 调用函数 max(void)，返回 −29

(3) 调用函数 max(int,int,int)，返回 30

题 4.3.4 参考答案：

```
Maximum of a and b is 99
Maximum of a, b and c is 153
1.0+max(h1,h2) = 1.05
Maximum of t, u and v is 70000
```

题 4.3.5 参考答案：

```
① char * max(char * x,char * y)
② return strcmp(x,y)>0? x:y;
```

题 4.3.6 参考答案：

(1) 构成函数重载，因为不同的枚举类型被认为是不同的数据类型。

(2) 不构成重载，因为 scientific 是 double 的别名，其实就是 double 型。

题 4.3.7 参考答案：

```
#include <iostream.h>
void swap(int& a, int& b)
{
        int t=a;
        a=b;
        b=t;
}
void swap(float& x, float& y)
{
        float t=x;
        x=y;
        y=t;
}
void swap(char * & s, char * & t)
{
        char *  p=s;
        s=t;
t=p;
}
```

题 4.3.8 参考程序：

```
#include <iostream.h>
#include <math.h>
class Degree
{
         double deg;
public:
         Degree(double x=0):deg(x)
         {}
         friend double cos(Degree& x)
         {
              return    cos(x.deg * 3.14159265/180) ;   //实参是实型，
                                                        //在此调用系统库函数
         }
};
int main()
{
         int n;
         for(n=-90;n<=90;n+=30)
         {
              Degree x(n)      ;              //定义一个对象 x 取值角度 n
              cout<<"cos("<<n<<")=";
cout<<cos(x)<<endl;                           //x 是对象类型，在此调用类 Degree 的友元函数
```

```
        }
        return 0;
}
```

测试结果：

```
cos(-90)=1.7949e-009
cos(-60)=0.5
cos(-30)=0.866025
cos(0)=1
cos(30)=0.866025
cos(60)=0.5
cos(90)=1.7949e-009
```

题 4.3.9 参考答案：

C++ 中可以通过 3 种方式重载运算符：用户定义的外部函数、类的成员函数以及友元函数。但 C++ 要求运算符重载中至少有一个参数是对象或者是对象的引用。

采用成员函数重载时，如果是单目运算符，则要求操作数必须是该类的对象；对于双目运算符，则要求左操作数必须是该类的对象。

而采用友元函数重载时，没有上述采用成员函数重载时的限制，只要求至少有一个参数是对象或者是对象的引用。如果要访问的资料成员是公开的，则没有必要采用友元函数形式，而可以直接采用外部函数的形式，因为采用友元无非是为了访问类中的私有成员。

当左操作数不是类的对象时，则只能采用友元（或外部函数，当资料成员公开时）的形式。

如果左操作数是类的对象，并且对于同一运算符既有成员函数又有友元函数的重载，则调用成员函数而不是友元函数。

题 4.3.10 参考程序：

```
#include <iostream.h>
class ModClass
{
private:
    int dataval;
public:
    ModClass(int v=0);
    ModClass operator+ (const ModClass& x);
    int GetValue(void) const;
    friend ModClass operator*(const ModClass& x,const ModClass& y);
    friend ostream& operator<<(ostream& os,const ModClass& x);
    operator int();
};
//类中的各成员函数的实现
ModClass::ModClass(int v):dataval(v%7) //确保整型资料成员 dataval,
```

```
{}                                                          //取值范围为 0..6
ModClass ModClass::operator +(const ModClass& x)            //运算符+重载,
{
    return ModClass(dataval+ x. dataval);
}
int ModClass::GetValue(void) const
{
    return dataval;
}
ModClass operator * (const ModClass& x,const ModClass& y)   //运算符 * 重载,
{
    return ModClass(x. dataval * y. dataval);
}
ostream& operator<<(ostream& os,const ModClass& x)          //运算符<<重载,

{
    os<<x. dataval<<endl;
    return os;
}
ModClass Inverse(ModClass& x)
{
    ModClass prod,value;
    for(int i=1;i<7;i++)
    {
        value=ModClass(i);
        prod=x * value;
        if(prod. GetValue()==1)
            break;
    }
    return value;
}

ModClass::operator int()                    //重载类型转换函数 int(),用于取代 GetValue
{
    return dataval;
}
void Solve(ModClass a, ModClass& x, ModClass b)             // 求解关于 x 的方程 ax=b。
{
    x=b * Inverse(a);
}

void main()
{
    ModClass a(10),b(6),c,d,x;
```

```
    c=a+b;
    d=a*b;
    cout<<"a="<<int(a)<<" b="<<int(b)<<endl;
    cout<<"a+b="<<c.GetValue()<<endl;
    cout<<"a*b="<<d;
    Solve(a,x,d);
    cout<<"x="<<x;
}
```

测试结果：

```
a=3 b=6
a+b=2
a*b=4
x=6
```

题 4.3.11 参考程序：

```
#include <iostream.h>
class complex
{
    double x,y;
public:
    complex(double r=0,double i=0)
    {
        x=r;y=i;
    }
    //binary operators
    complex operator+(const complex&);
    complex operator-(const complex&);
    complex operator*(const complex&);
    complex operator/(const complex&);
    //Negation
    complex operator-(void);
    //stream I/O operator
    friend ostream& operator<<(ostream& os,const complex&);
    friend istream& operator>>(istream& is,complex&);
    //binary operators using friend
    friend complex operator+(const complex&,const complex&);
    friend complex operator-(const complex&,const complex&);
    friend complex operator*(const complex&,const complex&);
    friend complex operator/(const complex&,const complex&);
    //negation using friend
    friend complex operator-(const complex&);
```

```
};
complex complex::operator +(const complex& rhs)
{
    return complex(x+rhs.x,y+rhs.y);
}
complex complex::operator -(const complex& rhs)
{
    return complex(x-rhs.x,y-rhs.y);
}
complex complex::operator *(const complex& rhs)
{
    return complex(x*rhs.x-y*rhs.y,x*rhs.y+y*rhs.x);
}
complex complex::operator /(const complex& rhs)
{
    double denom=rhs.x*rhs.x+rhs.y*rhs.y;
    return complex((x*rhs.x+y*rhs.y)/denom,(y*rhs.x-x*rhs.y)/denom);
}
complex complex::operator -(void)
{
    return complex(-x,-y);
}
ostream& operator<<(ostream& os,const complex& c)
{
    os<<'('<<c.x<<','<<c.y<<')';
    return os;
}
istream& operator>>(istream& is,complex& c)
{
    is>>c.x>>c.y;
    return is;
}
complex operator+(const complex& a,const complex& b)
{
    return complex(a.x+b.x,a.y+b.y);
}
complex operator-(const complex& a,const complex& b)
{
    return complex(a.x-b.x,a.y-b.y);
}
complex operator*(const complex& a,const complex& b)
{
    return complex(a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x);
}
```

```
complex operator/(const complex& a,const complex& b)
{
    double denom=b.x*b.x+b.y*b.y;
    return complex((a.x*b.x+a.y*b.y)/denom,(a.y*b.x-a.x*b.y)/denom);
}
complex operator -(const complex& c)
{
    return complex(-c.x,-c.y);
}
void main()
{
    complex a(2,3),b(5,1),c,d;
    cin>>c;
    cout<<"c="<<c<<endl;
    d=a+b;
    cout<<"a+b="<<d<<endl;
    d=c-b;
    cout<<"c-b="<<d<<endl;
    d=8*b;
    cout<<"8*b="<<d<<endl;
    d=a/b;
    cout<<"a/b="<<d<<endl;
    cout<<"-c="<<-c<<endl;
}
```

测试结果：

```
a=(2,3)
b=(5,1)
c=5.5 6.6
c=(5.5,6.6)
a+b=(7,4)
c-b=(0.5,5.6)
8*b=(40,8)
a/b=(0.5,0.5)
-c=(-5.5,-6.6)
```

题 4.3.12 参考程序：

```
#include "iostream.h"
#include "math.h"
class CRational{
    int numerator,denominator;
    void optimization();                    //规范化有理数私有成员函数
```

```
public:
    CRational(int x=0,int y=1);
    friend CRational operator * (CRational,CRational);
    friend CRational operator / (CRational,CRational);
    friend double Real(CRational);          //将有理数类对象转换为实数
    void print();
};
CRational::CRational(int x,int y)
{
    numerator=x;
    denominator=y;
    optimization();                         //有理数规范化
}
CRational operator * (CRational n,CRational m)
{
    CRational * temp;
    temp=new CRational;
    temp->numerator=n.numerator*m.numerator;
    temp->denominator=n.denominator*m.denominator;
    temp->optimization();
    return *temp;
}

CRational operator / (CRational n,CRational m)
{
    CRational * temp;
    temp=new CRational;
    temp->numerator=n.numerator*m.denominator;
    temp->denominator=n.denominator*m.numerator;
    temp->optimization();
    return *temp;
}

void CRational::print()
{
    cout<<numerator<<"/"<<denominator;
}
void CRational::optimization()
{
    int gcd;
    if(numerator==0)                        //若分子为0,则置分母为1后返回
    {
        denominator=1;
        return;
```

```
    }
    gcd=abs(numerator)>abs(denominator)? abs(numerator):abs(denominator);
    if(gcd==0) return;
    for(int i=gcd;i>1;i--)                          //找最大公约数
        if((numerator%i==0)&&(denominator%i==0)) break;
    numerator/=i;
    denominator/=i;
    if(numerator<0 && denominator<0)
    {
        numerator=abs(numerator);
        denominator=abs(denominator);
    }
    else if(numerator<0  | denominator<0)
    {
        numerator=-abs(numerator);
        denominator=abs(denominator);
    }
}
double Real(CRational x)                            //将有理数转换为实数
{
    return(double(x.numerator))/(double(x.denominator));
}

void main()
{
    CRational r1(5,8),r2(8,16),r3;
    cout<<"R1=";
    r1.print();
    cout<<"="<<Real(r1)<<endl;
    cout<<"R2=";
    r2.print();
    cout<<"="<<Real(r2)<<endl;
    cout<<"R1*R2=";
    r3=r1*r2;
    r3.print();
    cout<<"="<<Real(r3)<<endl;
    cout<<"R1/R2=";
    r3=r1/r2;
    r3.print();
    cout<<"="<<Real(r3)<<endl;
}
```

测试结果：

R1=5/8=0.625
R2=1/2=0.5
R1 * R2=5/16=0.3125
R1/R2=5/4=1.25

题 4.3.13 参考答案：

① deg+=a.deg;
② a.deg<=b.deg? true:false;

测试结果：

1
0.866025
0.5
1.7949e-009
-0.5
-0.866025
-1

题 4.3.14 参考答案：

① String operator+(const String& a,const String& b)deg+=a.deg
② buf=new char[n]
③ return String(buf)

测试结果：

a=Lion
b=Tiger
c=Bear
a+b=LionTiger
a+b+c=LionTigerBear

题 4.3.15 参考程序：

```
#include <iostream.h>
#include <string.h>
template <class T>
T Max(const T& x,const T& y)
{
        return (x<y)? y:x;
}
char * Max(char * x,char * y)
```

```
{
        return strcmp(x,y)<0? y:x;
}
void main()
{
        int x(10),y(20);
        double a(2.3),b(1.0);
        char * s1="def", * s2="ghi";
        cout<<"maximum of x and y is "<<Max(x,y)<<endl;
        cout<<"maximum of a and b is "<<Max(a,b)<<endl;
        cout<<"maximum of s1 and s2 is "<<Max(s1,s2)<<endl;
}
```

测试结果：

```
maximum of x and y is 20
maximum of a and b is 2.3
maximum of s1 and s2 is ghi
```

题 4.3.16 参考程序：

```
#include <iostream.h>
#define N 100
class CVector{                                  //声明向量类
    int lens;                                   //不同对象向量的元素个数
    double elems[N];                            //预定义一个 double 类型一维数组
public:
    CVector(int);
    double operator () (int);
    void setelem(int,double);
    friend CVector operator + (CVector,CVector);
    friend CVector operator - (CVector,CVector);
    void print();
};
CVector::CVector(int len)                       //定义构造函数
{
    lens=len;
}
double CVector::operator () (int i)             //重载运算符()的定义
{                                               //返回位置 i 的元素值
    return ((i>=1&&i<=lens)? elems[i]:0.0);
}
void CVector::setelem(int i,double val)         //为向量元素赋值
{
```

```
    if(i>=1&&i<=lens)
        elems[i]=val;
}
CVector operator + (CVector x,CVector y)          //重载运算符+的定义
{
    CVector m(x.lens);
    if(x.lens!=y.lens)
        return m;
    for(int i=1;i<=x.lens;i++)
            m.setelem(i,x(i)+y(i));
        return m;
}
CVector operator - (CVector x,CVector y)          //重载运算符-的定义
{
    CVector m(x.lens);
    if(x.lens!=y.lens)
        return m;
    for(int i=1;i<=x.lens;i++)
            m.setelem(i,x(i)-y(i));
        return m;
}

void CVector::print()                             //输出向量的各元素值
{
    for(int i=1;i<=this->lens;i++)
            cout<<(*this)(i)<<" ";
        cout<<endl;

}
void main()
{
    CVector a(4),b(4),c(4),d(4);
    a.setelem(1,0.1);                             //给向量对象a赋初值
    a.setelem(2,0.2);
    a.setelem(3,0.3);
    a.setelem(4,0.4);
    b.setelem(1,1.0);                             //给向量对象a赋初值
    b.setelem(2,2.0);
    b.setelem(3,3.0);
    b.setelem(4,4.0);
    cout<<"Vector A:"<<endl;
    a.print();
    cout<<"Vector B:"<<endl;
    b.print();
```

```
    c=a+b;                                  //两个向量相加
    d=b-a;                                  //两个向量相减
    cout<<"Vector A+B:"<<endl;
    c.print();
    cout<<"Vector B-A:"<<endl;
    d.print();
}
```

测试结果：

```
Vector A:
0.1  0.2  0.3  0.4
Vector B:
1  2  3  4
Vector A+B:
1.1  2.2  3.3  4.4
Vector B-A:
0.9  1.8  2.7  3.6
```

题 4.3.17 参考程序：

```
#include "iostram.h"
#define ROW 50
#define COL 50
class CMatrix{                              //声明矩阵类
    int rows,cols;                          //不同对象矩阵的行、列数
    double elems[ROW][COL];                 //预定义一个 double 类型数组
public:
    CMatrix(int,int);
    double operator () (int,int);
    void setelem(int,int,double);
    friend CMatrix operator + (CMatrix,CMatrix);
    friend CMatrix operator - (CMatrix,CMatrix);
    void print();
};
CMatrix::CMatrix(int rs,int cs)             //定义构造函数
{
    rows=rs;
    cols=cs;
}
double CMatrix::operator () (int row,int col)   //重载运算符()的定义
{                                           //返回 row 行 col 列的元素值
    return ((row>=1&&row<=rows&&col>=1&&col<=cols)?
        elems[row][col]:0.0);
```

```
}
void CMatrix::setelem(int row,int col,double val)          //为矩阵元素赋值
{
    if(row>=1&&row<=rows&&col>=1&&col<=cols)
            elems[row][col]=val;
}
CMatrix operator + (CMatrix x,CMatrix y)                   //重载运算符+的定义
{
    CMatrix m(x.rows,x.cols);
    if(x.rows!=y.rows||x.cols!=y.cols)
        return m;
    for(int r=1;r<=x.rows;r++)
        for(int c=1;c<=x.cols;c++)
            m.setelem(r,c,x(r,c)+y(r,c));
        return m;
}
CMatrix operator - (CMatrix x,CMatrix y)                   //重载运算符-的定义
{
    CMatrix m(x.rows,x.cols);
    if(x.rows!=y.rows||x.cols!=y.cols)
        return m;
    for(int r=1;r<=x.rows;r++)
        for(int c=1;c<=x.cols;c++)
            m.setelem(r,c,x(r,c)-y(r,c));
        return m;
}

void CMatrix::print()                                      //按行输出矩阵的各元素值
{
    for(int r=1;r<=this->rows;r++)
    {
        for(int c=1;c<=this->cols;c++)
            cout<<(*this)(r,c)<<" ";
        cout<<endl;
    }
}
void main()
{
    CMatrix a(2,3),b(2,3),c(2,3),d(2,3);
    a.setelem(1,1,1.0);                                    //给矩阵对象a赋初值
    a.setelem(1,2,2.0);
    a.setelem(1,3,3.0);
    a.setelem(2,1,4.0);
    a.setelem(2,2,5.0);
```

```
        a.setelem(2,3,6.0);
        b.setelem(1,1,0.1);                    //给矩阵对象 b 赋初值
        b.setelem(1,2,0.2);
        b.setelem(1,3,0.3);
        b.setelem(2,1,0.4);
        b.setelem(2,2,0.5);
        b.setelem(2,3,0.6);
        cout<<"Matrix A:"<<endl;
        a.print();
        cout<<"Matrix B:"<<endl;
        b.print();
        c=a+b;                                 //两个矩阵相加
        d=a-b;                                 //两个矩阵相减
        cout<<"Matrix A+B:"<<endl;
        c.print();
        cout<<"Matrix A-B:"<<endl;
        d.print();
}
```

测试结果：

```
Matrix  A:
1  2  3
4  5  6
Matrix  B:
0.1  0.2  0.3
0.4  0.5  0.6
Matrix  A+B:
1.1  2.2  3.3
4.4  5.5  6.6
Matrix  A-B:
0.9  1.8  2.7
3.6  4.5  5.4
```

题 4.3.18 参考答案：

```
#include <iostream.h>
#include <process.h>
template <class T>
class DataStore
{
        T dataElements[5];
        int loc;
public:
```

```
        DataStore();loc(0)
        {}
        int Insert(T elt);
        int Find(T elt);
        int NumElts(void);
        T& GetData(int n);
};
template <class T>
int DataStore<T>::Insert(T elt)
{
        if(loc<5)
        {
            dataElements[loc]=elt;
            loc++;
            return 1;
        }
        else
            return 0;
}
template<class T>
int DataStore<T>::Find(T elt)
{
    for(int i=0;i<loc;i++)
        if(dataElements[i]==elt)
            break;
    if(i<loc)
        return i;
    else
        return -1;
}
template<class T>
int DataStore<T>::NumElts(void)
{
        return loc;
}
template<class T>
T& DataStore<T>::GetData(int n)
{
        if(n<0||n>4)
        {
            cerr<<"location out of range."<<endl;
            exit(0);
        }
        else
```

```
        return dataElements[n];
}
```

题 4.3.19 参考函数：

```
#include <iostream.h>
template <class T>
int Max(T Arr[], int n)
{
        int currMaxIndex=0;
        for(int i=1;i<n;i++)
            if(Arr[currMaxIndex]<Arr[i])
                currMaxIndex=i;
        return currMaxIndex;
}
```

测试主函数：

```
void main()
{
        int arr[]={3,5,2,1,7,4};
        cout<<"The index of the maximum value in the array is ";
        cout<<Max(arr,6)<<endl;
}
```

测试结果：

```
The index of the maximum value in the array is 4
```

题 4.3.20 参考函数：

```
template <class T>
void InsertOrder(T A[], int n, T elem)
{
    int i=0;      //assume n>0
    while(i<n && A[i]<elem)      //locate insertion point
        i++;
    if(i<n)
        for(int j=n;j>i;j--)      //move tail to right
            A[j]=A[j-1];
    A[i]=elem;      //insert the element
}
```

测试主函数：

```
void main()
{
    int a[10]={1,3,5,7,9,11,13,15};
    cout<<"Before insert:"<<endl;
    for(int i=0;i<10;i++)
        cout<<a[i]<<"  ";
    cout<<endl;
    InsertOrder(a,10,10);
    cout<<"After insert:"<<endl;
    for(i=0;i<10;i++)
        cout<<a[i]<<"  ";
    cout<<endl;
}
```

测试结果：

```
Before insert:
1  3  5  7  9  11  13  15  0  0
After insert:
1  3  5  7  9  10  11  13  15  0
```

题 4.3.21 参考程序：

```
#include <iostream.h>
template <class T>
void Copy(T A[],T B[],int n)
{
    for(int i=0;i<n;i++)
        *A++=*B++;
}
struct Student
{
    int field1;
    double field2;
};
int main()
{
    int AInt[6],BInt[6]={1,3,5,7,9,11};
    Student AStudent[3],BStudent[3]={{1,3,5},{3,0},{5,5,5}};
    Copy(AInt,BInt,6);
    Copy(AStudent,BStudent,3);
    for(int i=0;i<6;i++)
        cout<<AInt[i]<<' ';
    cout<<endl;
```

```
    for(i=0;i<3;i++)
        cout<<AStudent[i].field1<<' '<<AStudent[i].field2<<' ';
    cout<<endl;
    return 0;
}
```

测试结果：

```
1  3  5  7  9  11
1  3.5  3  0  5  5.5
```

题 4.3.22 参考答案：

不能，需在类中添加复制构造函数和赋值运算符重载函数。

```
T::T(T& t)
{
        p1=new char[strlen(t.p1)+1];
        strcpy(p1,t.p1);
}
T& T::operator =(T& t)
{
        delete []p1;
        p1=new char[strlen(t.p1)+1];
        strcpy(p1,t.p1);
        return *this;
}
```

测试结果：

```
p1=good luck
p1=good luck
p1=hi
p1=hi
```

解释：因为程序中涉及动态内存分配，所以如果不添加上面两个函数，而使用默认的复制构造函数和默认的赋值运算符重载函数，则仅仅对数据成员进行复制，就会使两个对象的资料成员——指针变量 p1 的值相同，从而指向同一存储区域。当其中一个对象先消亡时，会调用析构函数释放其指针所指的存储区域，从而导致另一个对象的指针指向已经被释放的区域，当另一个对象消亡时，再次释放已经释放的区域，则会导致运行时错误。

通过添加上面两个函数，复制的不是指针变量本身的值，而是指针所指向的对象，使两个对象的指针指向各自的存储区域，释放的时候也是各自释放自己的存储区域，因而不会出现运行错误。

第5章　程序的类层次结构

5.1　编程知识

5.1.1　类的继承与派生

1. 派生方式

- public 派生
- private 派生
- protected 派生

2. 派生方式对成员访问属性的影响

图 5.1 为基类中的 3 种访问属性不同的成员，经 3 种派生后，在派生类中的访问属性的变化情况。

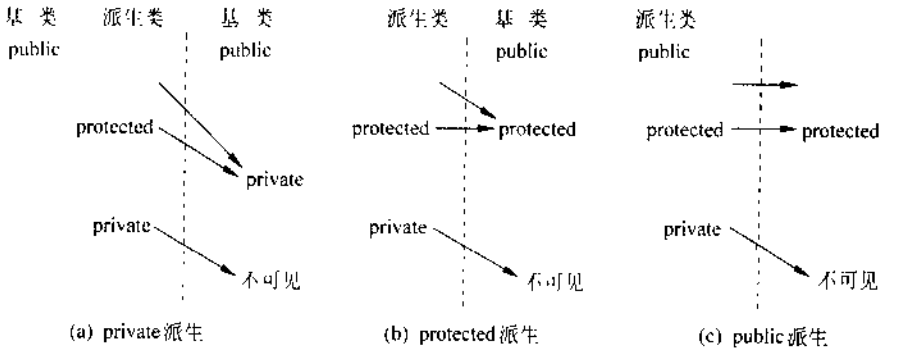

图 5.1　基类成员在派生类中访问属性的变化

3. 派生类中构造函数/释放函数的定义与调用

在类层次结构中，构造函数与释放函数是不能继承的。为此，对派生类必须重新定义构造函数与释放函数，并且要首先通过派生类的构造函数调用基类的构造函数，对基类成员进行初始化，然后再执行对在派生类中新增的成员进行初始化的操作。因此可以说，派生类对象是由其所有先辈类共同创建的。一般，派生类的构造函数有如下原型格式：

Y::Y(argX1,argX2,…,argY1,argY2,…):X(argX1,argX2,…)

其中，X 为 Y 的直接基类名，Y 为派生类名。这一格式是递归的。

4. 派生类对象的创建方式

(1) 用常数参数表创建；

(2) 由部分基类对象创建派生类对象。

5. 虚基类

在多基派生中，如果在多条继承路径上有一个公共的基类，便会产生来自不同路径的公共基类的多个拷贝。如果只想保留公共基类的一个拷贝，就必须使用关键字 virtual 把这个公共基类定义为虚基类。

虚基类在定义由基类直接派生的类时说明，说明的格式为

```
class 派生类名 : virtual 派生方式  基类名 {
    ...
};
```

6. 类层次中的成员名多重作用域

在类层次结构中形成成员名多重作用域：

(1) 基类成员作用域和派生类新增成员作用域形成相包含的关系，派生类新增成员作用域为内层，基类成员作用域在外层；

(2) 内层声明的标识符可以覆盖外层声明的同名标识符，使其不可见；

(3) 如果在几个基类中具有同名成员，但在派生类中没有与它们同名的新增成员名，则由于这些成员名具有相同的作用域，使系统无法判定到底是调用哪个基类成员，即用"对象名. 成员名"的方式无法惟一地标识它们，必须通过作用域分辨符来标识成员。

5.1.2 虚函数

1. 虚函数的概念及使用规则

虚函数是用关键字 virtual 修饰的某基类中的 protected 或 public 成员函数。它可以在派生类中重定义，以形成不同的版本。只有在程序执行过程中，依据指针具体指向哪个类对象，或依据引用哪个类对象，才能确定激活哪一个版本，实现动态绑定。表 5.1 为 C++ 中静态多态性与动态多态性的主要特征比较。

表 5.1 C++ 中静态多态性与动态多态性的主要特征比较

形 式	绑定方式	各版本的函数原型	绑 定 依 据	特 点
函数名重载	静态绑定	不同	参数数目及类型	高效
虚函数	动态绑定	相同	程序运行时引用或指针指向	高灵活性、抽象性和可扩充性

一个指向基类的指针，可以用来指向任何派生类对象中从基类继承来的成员。这是 C++ 实现动态多态性的关键。当在基类中把某函数定义为虚函数后，并在后辈类中提供了其不同版本(重新定义)时，使用指向基类的指针这样一个统一的操作接口，可以让其指向按实际生成的对象，以调用公开派生的后辈类中重新定义的函数，实现动态绑定。

虚函数声明与重定义的一般规则是：

(1) 在基类中，用关键字 virtual 可以将其 public 或 protected 部分的成员函数声明为虚函数；

(2) 如果派生类中没有对基类中说明的虚函数进行重定义，则它继承基类中的虚函数；

(3) 虚函数只能是类的成员函数，不能把虚函数声明为静态的或全局的；也不能把友元说明为虚函数，但是虚函数可以是另一个类的友元函数；

(4) 由类的成员函数访问该类层次中的虚函数　　要使用 this 指针。

2. 纯虚函数与抽象类

当在基类中不能为虚函数提供有实际意义的定义时，可以将这个虚函数声明为纯虚函数。声明纯虚函数的形式为

virtual　类型　函数名(参数列表)　=　0;

纯虚函数不可以被继承。当基类是抽象类时，在派生类中必须给出基类中纯虚函数的定义，或在该类中再声明其为纯虚函数。只有在派生类中给出了基类中所有纯虚函数的实现时，该派生类就不再成为抽象类。抽象类仅充当一个统一的接口的作用。

3. 虚释放函数

虽然构造函数不可以被声明为虚函数，但是释放函数可以被声明为虚函数。一般说来，如果一个类中定义了虚函数，释放函数也应定义为虚释放函数，尤其是释放函数要完成一些有意义的工作，如释放内存。

5.2 例题示范

例 5.2.1　写出图 5.2 表示的 C++ 类层次结构。

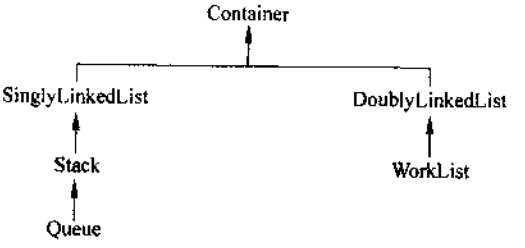

图　5.2

解：

```
class Container
{
    ...
}
class SinglyLinkedList:public Container
```

```
{
    ...
}
class Stack:public SinglyLinkedList
{
    ...
}
class Queue:public Stack
{
    ...
}
class DoublyLinkedList:public Container
{
    ...
}
class WorkList:public DoublyLinkedList
{
    ...
}
```

例 5.2.2 定义一个时间类层次结构：

(1) 类 CTime 以年、月、日、时、分、秒为数据元素；

(2) 由 CTime 类派生 STime 类，追加国名及其与格林威治时间的差。

解：

```
#include <iostream.h>
#include <string.h>
class CTime
{
    protected:      //注 1
        int year,month,day;
        int hour,minute,second;
    public:
        CTime(int y=2002,int m=1,int d=1,int h=0,int mi=0,int s=0)
        {
            year=y;month=m;day=d;
            hour=h;minute=mi;second=s;
        }
        friend ostream& operator<<(ostream& os,CTime& t)
        {
            os<<t.year<<"年"<<t.month<<"月"<<t.day<<"日"<<endl;
            os<<t.hour<<"时"<<t.minute<<"分"<<t.second<<"秒"<<endl;
            return os;
        }
};
class STime:public CTime
```

```
{
        char contry[20];
        int bh;                                          // 与标准时间的差
    public:
        STime(char * p,int a,int y=2002,int m=1,
                int d=1,int h=0,int mi=0,int s=0)        // 注 2
        :CTime(y,m,d,h,mi,s)
        {
            strcpy(contry,p);
            bh=a;
        }
        friend ostream& operator<<(ostream& os,STime& t)
        {
            os<<"国名:"<<t.contry<<endl;
            os<<(CTime&)t;                               //注 3
            os<<"与格林威治标准时间的差:"<<t.bh<<endl;
            return os;
        }
        void calc(STime& u)
        {
            year=u.year;month=u.month;day=u.day;
            hour=u.hour-u.bh+bh;minute=u.minute;second=u.second;
        }
};
void main()
{
        STime a("中国",8,2002,9,2,19,40,00);
        STime b("日本",9);
        cout<<a;
        b.calc(a);
        cout<<b;
}
```

测试结果:

```
国名:中国
2002 年 9 月 2 日
19 时 40 分 0 秒
与格林威治标准时间的差:8
国名:日本
2002 年 9 月 2 日
20 时 40 分 0 秒
与格林威治标准时间的差:9
```

解释:

(1) 为了能让派生类继承基类的数据成员,访问属性应使用 protected;

(2) 由于部分参数有默认值，故没有默认值的参数应放在参数列表的左边；

(3) 调用基类的友元插入运算符。

还有要注意的是类的命名不能跟库中已有的类重名，否则会造成类重复定义的错误。例如，本例中如果编写的是 MFC 程序，则在 MFC 中已经有一个名叫 CTime 的类，自己编写的类就不能再命名为 CTime；而这儿编写的是控制台应用程序，没有用到 MFC，故不会出现重复定义的错误。

另外，如果将 main 函数改为：

```
void main()
{
    STime a("中国",8,2002,9,2,13,40,00);
    STime b("夏威夷(美国)",-10);
    b.calc(a);
    cout<<b;
}
```

则输出结果为：

```
国名：夏威夷(美国)
2002 年 9 月 2 日
-5 时 40 分 0 秒
与格林威治标准时间的差：-10
```

这时时间出现了负值，这与习惯表述是不符合的，为此可以改写 calc 函数如下：

```
void calc(STime& u)
{
    int md[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
    year=u.year;month=u.month;day=u.day;
    hour=u.hour-u.bh+bh;minute=u.minute;second=u.second;
    if(year%4==0 && year%100!=0 || year%400==0)
        md[2]=29;
    else
        md[2]=28;
    if(hour<0)
    {
        hour=hour+24;
        day--;
        if(day<=0)
        {
            month--;
            if(month<=0)
            {
```

```
                year--;
                month=12;
            }
            day=md[month];
        }
    }
    if(hour>24)
    {
        hour-=24;
        day++;
        if(day>md[month])
        {
            month++;
            day=1;
            if(month>12)
            {
                year++;
                month=1;
            }
        }
    }
}
```

例 5.2.3 用 public 派生类对象访问带参派生类成员函数，并调用基底类成员函数的方法，求最大公约数。

解：

```
#include <iostream.h>
class BaseCL
{
protected:
    int m,n;
public:
    int GCD(void)
    {
        int r;
        while(r=m%n)
        {
            m=n;n=r;
        }
        return n;
    }
};
class DerivedCL:public BaseCL
```

```
}
public:
    int GCD(int x,int y)
    {
        m=x;n=y;
        return BaseCL::GCD();
    }
};
void main()
{
    DerivedCL d;
    cout<<"the GCD of 24 and 18 is: "<<d.GCD(18,24)<<endl;
}
```

例 5.2.4 用递归调用被继承的先辈基底类成员函数的方法求阶乘。

解：

```
#include <iostream.h>
#include <process.h>
class BaseCL
{
public:
    int fact(int n)
    {
        if(n<0)
        {
            cout<<"invalid number."<<endl;
            exit(0);
        }
        if(n==0)
            return 1;
        else
            return n*fact(n-1);
    }
};


class Derived:public BaseCL
{
    int n;
public:
    void input()
    {
        cout<<"input a number(>=0):"<<endl;
        cin>>n;
```

```
    }
    int fact()
    {
        return BaseCL::fact(n);
    }
    void output()
    {
        cout<<"the factorial of "<<n<<" is: "<<fact()<<endl;
    }
};

void main()
{
    Derived d;
    d.input();
    d.output();
}
```

例 5.2.5 试建立一个异种链表。

解：

本题假定链表中的异种对象具有共同的基类，由于具有共同的基类，所以可以用基类指针指引不同的对象。另外为简单起见，本题只考虑构建链表，而且结点总是插入在链表起始位置，而不考虑删除结点。

下面代码中，异种对象涉及的类层次结构为 Talent→Person。

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

class Person            //基类
{
    char name[24];
    int age;
    Person * next;   //指向后续结点
public:
    Person(char *n="",int a=0,Person *p=NULL)
    {
        strcpy(name,n);
        age=a;
        next=p;
    }

    virtual void print()
    {
```

```
        cout<<"姓名:"<<name<<endl;
        cout<<"年龄:"<<age<<endl;
    }

    void list()
    {
        Person * p=this;
        while(p!=NULL)
        {
            p->print();
            p=p->next;
        }
    }
};

class Talent:public Person
{
        char place[24];          //出生地
public:
        Talent(char * n="",int a=0,char * pl="",Person * p=NULL)
        :Person(n,a,p)
        {
            strcpy(place,pl);
        }

    void print()
    {
        Person::print();
        cout<<"出生地:"<<place<<endl;
    }
};

void main()
{
        Person a("小布什",56);
        Person b("布莱尔",54,&a);
        Person c("普京",60,&b);
        Talent d("萨达姆",58,"伊拉克",&c);
        Talent e("奥马尔",67,"阿富汗",&d);
        Talent f("本·拉登",67,"沙特阿拉伯",&e);
        f.list();
}
```

5.3 自我测试

5.3.1 自测习题

题 5.3.1 某一书的类层次结构如下，请将空白处填以适当的 C++ 语言成分。

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

class Book {
        char  bookname[24];          //书名
        char  author[20];            //著者
  public:
            Book(char *p="",char *q="")
              { strcpy(bookname,p);strcpy(author,q);}
              [          ①          ]
              { strcpy(bookname,x.bookname);strcpy(author,x.author);}
        void  print();
};

class  Series:public  Book {
        char  sername[24];           //系列名
        int   volume;                //卷数
   public:
            Series(char *p="",char *q="", char *sn="",int v=0)
            :[          ②          ]
             Series(Book x,char *sn="",int v=0)
            [          ③          ]
        void  print();
};

void Book::print()
{
        cout<<"书名:"<<bookname<<endl;
        cout<<"著者:"<<author<<endl;
}

void Series::print()
{
        Book::print();
```

```
        cout<<"系列名:"<<sername<<"第"<<volume<<"卷\n";
}
应用程序:
void main(void)
{
        Book a("现代日语听力教程","张展赫");
        Series b( [          ④          ] );
        b.print();
}
```

题 5.3.2 有如下表示国家(Country)、中东地区(Middle East)、海湾(Gulf)国家的类层次结构。在空白处填上适当的 C++ 成分。

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

class Country {
    protected:
        char    name[20];                       //国名
        char    capital[20];                    //首都
  public:
        Country (char *n="",char *cap="");
        void    print();
};

[              ①              ]  {        //中东地区
    protected:
        char    people[20];                     //民族
        int     percent;                        //百分比
    public:
            MiddleEast (char *n="",char *cap="",char *pp="",int pr=0);
            void    print();
};

[              ②              ]  {        //海湾国家
        int     oil;                            //原油产量
    public:
          Gulf(char *n="",char *cap="",char *pp="",int pr=0,int ol=0);
        void    print();
};
```

```
Country:: Country(char * n,char * cap)
{       strcpy(name,n);strcpy(capital,cap);}
void Country::print()
{
        cout<<"国名:"<<name<<",首都:"<<capital<<endl;
}
MiddleEast :: MiddleEast(char * n,char * cap, char * pp,int pr)
                :  [          ③          ]
void   MiddleEast::print()
{
          cout<<"国名:"<<name<<",首都:"<<capital<<endl;
          cout<<"民族:"<<people<<" "<<percent<<"%\n";
}

Gulf:: Gulf(char * n,char * cap,char * pp, int pr,int ol)
                :  [          ④          ]
void   Gulf::print()
{
            cout<<"国名:"<<name<<",首都:"<< capital<<endl;
            cout<<"民族:"<<people<<" "<<percent<<"%\n";
            cout<<"原油生产量:"<<oil<<"万桶\n";
}
```

题 5.3.3 分析下面的类层次结构：

```
struct  A {
        int     b;
        void    (int);
};
struct  B:A  {
        void    b();
        void    fun(double);
};
struct  C:A,B { };
```

对于声明

```
B   b;
C   c;
```

下面的语句各能访问哪些成员？为什么？

(1) b.b();

(2) b.fun(1,5);

(3) c.b();

(4) c.fun(5)。

题 5.3.4 将上题中的 struct 改为 class 后，该类层次结构应做如何修改？

题 5.3.5 定义 Location—Point—Circle 类层次结构：

Location 类：用 x,y 坐标描述一个平面位置，得到 x 坐标与得到 y 坐标。

Point 类：在 Location 类的基础上增加如下成员：

(1) 点是否可见(Visible)；

(2) 显示(Show)一个点；

(3) 抹掉(Hide)一个点；

(4) 得到点是否可见(IsVisible)的信息；

(5) 将这个点从当前位置移到(MoveTo)新的位置。

Circle 类：在 Point 类的基础上增加如下成员：

(1) 圆的半径(Radius)；

(2) 显示(Show)一个圆；

(3) 抹掉(Hide)一个圆；

(4) 放大(Expand)一个圆；

(5) 移动(MoveTo)一个圆；

(6) 缩小(Contract)一个圆。

题 5.3.6 用 OOP 方法设计一个输出 ASCII 代码表的 C++ 程序。

题 5.3.7 BitSet 类定义。集合就是相同类型中的不同对象的总体。在数学中，集合可以用两种方法描述：穷举法与描述法。集合有如下一些重要概念：

(1) 集合中任两个元素不全相同；

(2) 元素在集合中没有次序之分；

(3) 一个集合可以作为另一个集合的元素。

在程序设计中，可以设置一些二进制标志位码，表示元素在集合中的存在与否。使用这种表示方法表示整数元素最为简单。如下面的表示：

集合元素取值范围：	1	2	3	4	5	6	7	8
元素标志位码：	1	1	1	0	1	0	0	1

单字节的二进制码 11101001 表示一个如下的集合：

{1, 2, 3, 5, 8}

这样，任何可以用整数表示的对象都可以成为集合的元素。这种用二进制表示的集合类型称为位集(BitSet)。集合的处理首先要从定义 BitSet 类开始。

题 5.3.8 构造一个数组类，使其与 C++ 预定义的数组使用相一致。

题 5.3.9 用模板创建一个通用数组(Array)类。

题 5.3.10 由处理时间的 Time 类与处理日期的 Date 类，生成处理和登记文件的 FileStamp 类。

题 5.3.11 对图 5.3 中所示的类层次结构，分别为以下情况：

(1) 虚基类；

(2) A 为虚基类；

(3) B 为虚基类。

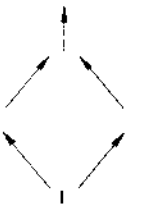

图 5.3

时，对于语句

A * pa, a;

B * pb, b;

C * pc, c;

D * pd, d;

E * pe, e;

下面的语句哪些是正确的？哪些是错误的？为什么？

```
pa = pb;
pb = pa;
pa = (A *)pb;
pb = (B *)pa;
a = b;
b = a;

pa = pc;
pc = pa;

pa = (A *)pc;
pc = (C *)pa;

a = c;
c = a;

pb = pc;
pc = pb;

pb = (B *)pc;
pc = (C *)pb;

b = c;
c = b;

pb = pe;
```

```
pe = pb;

pb = (B *)pe;
pe = (E *)pb;

b = e;
e = b;

pa = pe;
pe = pa;

pa = (A *)pe;
pe = (E *)pa;

a = e;
e = a;
```

题 5.3.12 模拟动物园的管理。这是一个豢养宠物的小型动物园。园中共有 20 个分隔栏，其中一半用于养猫，另一半用于养狗。

题 5.3.13 如果有以下基类，请问每个派生类的声明中有何错误？

```
class Base
{
    ...
public:
    Base(int a, int b);
    ...
};

(1)

class DerivedCL1: public Base
{
private:
    int q;
public:
    DerivedCL1(int z): q(z)
    {}
    ...
};
```

(2)

```
class DerivedCL2: public Base
{
private:
    ...
public:
    //DerivedCL2 has no constructor
    ...
};
```

题 5.3.14 下列程序说明了构造函数和析构函数在继承链中的执行次序。Base1、Base2 和 Derived 3 个类都有自己的构造函数和析构函数。请给出以下程序的输出。

```
#include <iostream.h>
class Base1
{
public:
    Base1(void)
    {
        cout<<"Base1 constructor called."<<endl;
    }
    ~Base1(void)
    {
        cout<<"Base1 destructor called."<<endl;
    }
};
class Base2
{
public:
    Base2(void)
    {
        cout<<"Base2 constructor called."<<endl;
    }
    ~Base2(void)
    {
        cout<<"Base2 destructor called."<<endl;
    }
};
class Derived:public Base1,public Base2
{
public:
    Derived(void):Base1(),Base2()
    {
```

```
        cout<<"Derived class constructor called."<<endl;
    }
    ~Derived(void)
    {
        cout<<"Derived class destructor called."<<endl;
    }
};
void main(void)
{
    Derived objD;
    {
        Base1 objB1;
        {
            Base2 objB2;
        }
    }
}
```

题 5.3.15 考虑以下继承链

```
class Base
{
    ...
public:
    void F(void);
    void G(int x);
    ...
};
class Derived: public Base
{
    ...
public:
void F(void);
void G(float x);
...
};
void Derived::G(float x)
{
    ...
    Base::G(10);    //在成员函数中使用作用域运算符
    ...
};
```

对于声明

Derived OBJ;

(1) 怎样引用基类中的函数 F;

(2) 怎样引用派生类中的函数 F;

(3) 编译器怎样处理语句 OBJ.G(20)?

题 5.3.16 请给出以下程序的输出:

```
#include <iostream.h>
#include <string.h>
class Base
{
private:
        char msg[30];
protected:
        int n;
public:
        Base(char s[],int m=0):n(m)
        {
            strcpy(msg,s);
        }
    void output(void)
    {
        cout<<n<<endl<<msg<<endl;
    }
};
class Derived1:public Base
{
private:
    int n;
public:
    Derived1(int m=1):Base("Base",m-1),n(m)
    {}
    void output(void)
    {
        cout<<n<<endl;
        Base::output();
    }
};
class Derived2:public Derived1
{
private:
```

```
    int n;
public:
        Derived2(int m=2):Derived1(m-1),n(m)
        {}
        void output(void)
        {
            cout<<n<<endl;
        Derived1::output();
        }
};
void main()
{
        Base B("Base Class",1);
        Derived2 D;
        B.output();
        D.output();
}
```

题 5.3.17 考虑以下类：

```
class BaseCL
{
protected:
    int one;
public:
    BaseCL(int a):one(a)
    {}
    virtual void Identify(void)
    {
        cout<<one<<endl;
    }
};
class DerivedCL:public BaseCL
{
protected:
    int two;
public:
    DerivedCL(int a,int b):BaseCL(a),two(b)
    {}
    virtual void Identify(void)
    {
        cout<<one<<" "<<two<<endl;
    }
};
```

和函数：

```
void Announce1(BaseCL x)
{
    x.Identify();
}
void Announce2(BaseCL& x)
{
    x.Identify();
}
void Announce3(BaseCL * x)
{
    x->Identify();
}
```

给出下列代码段的输出：

```
BaseCL A(7), *p, *arr[3];
DerivedCL B(3,5),C(2,4);
Announce1(A);
Announce1(C);
Announce2(B);
Announce3(&C);
p=&C;
p->Identify();
for(int i=0;i<3;i++)
    if(i==1)
        arr[i]=new BaseCL(7);
    else
        arr[i]=new DerivedCL(i,i+1);
for(i=0;i<3;i++)
    arr[i]->Identify();
```

题 5.3.18 请解释为什么在任何充当基类的类中析构函数应该声明为虚函数。

5.3.2 参考答案

题 5.3.1 参考答案：

① Book(Book& x)
② Book(p,q){strcpy(sername,sn);volume=v;}
③ :Book(x){strcpy(sername,sn);volume=v;}
④ a,"现代日语系列教程",3

题 5.3.2 参考答案：

① class MiddleEast:public Country
② class Gulf:public MiddleEast
③ Country(n,cap){strcpy(people,pp);percent=pr;}
④ MiddleEast(n,cap,pp,pr){oil=ol;}

题 5.3.3 参考答案：

(1) b.b()可以访问类 A 中的数据成员 b、成员函数 fun(int)，以及类 B 中的成员函数 fun(double)。因为对 struct 定义的类来说，成员的默认访问属性是公开的，派生方式默认时也是公开的，所以类 B 继承类 A 的所有成员，因而 b.b()可以访问类 B 的所有成员(包括继承得到的)。

(2) b.fun(1.5)可以访问类 A 中的数据成员 b、成员函数 fun(int)，以及类 B 中的成员函数 b()。原因同(1)。

(3) c.b()可以访问类 A 中的数据成员 b、成员函数 fun(int)，以及类 B 中的成员函数 fun(double)。因为类 C 继承了类 A 和类 B 的所有成员。

(4) c.fun(5)可以访问类 A 中的数据成员 b 以及类 B 中的成员函数 b()和成员函数 fun(double)。原因同(3)。

题 5.3.4 参考答案：

因为 class 中成员的默认访问属性是私有的，派生方式默认时也是私有的，为了不改变题 5.3.3 中各语句能够访问的成员，应该在题 5.3.3 的类层次结构中为各成员加上访问属性 public，在基类列表中为各基类加上派生方式 public。即：

```
class   A {
public:
        int     b;
        void    fun(int);
};
class   B: public A   {
public:
        void    b();
        void    fun(double);
};
class   C: public A, public B { };
```

题 5.3.5 参考答案：

```
#include <graphics.h>
class Location
{
    protected:
        int x,y;
    public:
        Location(int x,int y)
```

```
        {
            this ->x=x;
            this ->y=y;
        }
        int getx()
        {
            return x;
        }
        int gety()
        {
            return y;
        }
};
class Point:public Location
{
    protected:
        bool visible;
    public:
        Point(int x,int y):Location(x,y)
        {
            visible =false;                     //默认情况下点不可见
        }
        void Show()
        {
            visible=true;
            putpixel(x,y,getcolor());           //使用默认颜色画点
        }
        void Hide()
        {
            visible=false;
            putpixel(x,y,getbkcolor());         //使用背景颜色画点,即擦除该点
        }
        bool IsVisible()
        {
            return visible;
        }
        void MoveTo(int x,int y)
        {
            Hide();
            this ->x=x;
            this ->y=y;
            Show();
        }
};
```

```
class Circle:public Point
{
    protected:
        int radius;                          //圆的半径
    public:
        Circle(int x,int y,int r):Point(x,y)
        {
            radius=r;
        }
        void Show()
        {
            visible=true;
            circle(x,y,radius);
        }
        void Hide()
        {
            visible=false;
            unsigned int temp_color;         //用于保存当前颜色的临时变量
            temp_color=getcolor();           //保存当前颜色
            setcolor(getbkcolor());          //设置当前颜色为背景色
            circle(x,y,radius);              //用背景颜色画圆,即擦除圆
            setcolor(temp_color);
        }
        void MoveTo(int x,int y)
        {
            Hide();
            this->x=x;
            this->y=y;
            Show();
        }
        void Expand(int delta)
        {
            Hide();
            radius=radius+delta;
            if(radius<0)radius=0;            //避免半径为负数
            Show();
        }
        void Contract(int delta)
        {
            Expand(-delta);
        }
};

#include <conio.h>                           //利用其中的 getch()函数暂停
```

```
void main()
{
    int graphdriver=DETECT,graphmode;
    initgraph(&graphdriver,&graphmode,"");
    Circle circle(100,100,50);
    circle.Show();                          //显示圆
    getch();
    circle.MoveTo(200,200);                 //移动圆
    getch();
    circle.Expand(50);                      //放大圆
    getch();
    circle.Contract(60);                    //缩小圆
    getch();
    closegraph();
}
```

说明：

以上程序中涉及了图形显示，而控制台应用程序是面向 DOS 文本系统的，为了进入图形系统，必须调用库函数 initgraph 来初始化图形系统，然后调用库函数 putpixel 画像素，调用 circle 函数画圆，调用 closegraph 关闭图形系统。而所有这些跟图形有关的函数原形都在头文件 graphics.h 中说明，实现函数的代码在库文件 graphics.lib 中。为此需在代码开始部分用文件包含命令 include 将 graphics.h 文件包含进来。

但遗憾的是以上代码在面向 Windows 的开发环境（诸如 VC++、C++ Builder、Borland C++5.0 for windows 以上版本等）中无法编译通过，原因是这些开发环境主要是用来开发 Windows 应用程序的，而 Windows 本身就是一个图形用户界面，其绘图方法是通过设备描述表来实现的，所以在这些开发环境中不再提供 graphics.lib 库文件。

如果要运行以上程序，可以在 Borland C++ for DOS 或 Turbo C++ 等环境中运行。

题 5.3.6 参考答案：

```
#include <iostream.h>
#include <iomanip.h>
class ASCIITable
{
public:
    void print(void)
    {
        for(int i=0;i<128;i++)
        {
            if(i%8==0)
                cout<<endl;
            cout<<setw(5)<<i<<";";
            if(I<30)            //不可显示的控制符
```

```
                cout<<" ";
            else
                cout<<char(i);
        }
        cout<<endl;
    }
};
void main()
{
    ASCIITable a;
    a.print();
}
```

测试结果：

0:		1:		2:		3:		4:		5:		6:		7:	
8:		9:		10:		11:		12:		13:		14:		15:	
16:		17:		18:		19:		20:		21:		22:		23:	
24:		25:		26:		27:		28:		29:		30:	_	31:	
32:		33:	!	34:	"	35:	#	36:	$	37:	%	38:	&	39:	'
40:	(	41:	)	42:	*	43:	+	44:	,	45:	-	46:	.	47:	/
48:	0	49:	1	50:	2	51:	3	52:	4	53:	5	54:	6	55:	7
56:	8	57:	9	58:	:	59:	;	60:	<	61:	=	62:	>	63:	?
64:	@	65:	A	66:	B	67:	C	68:	D	69:	E	70:	F	71:	G
72:	H	73:	I	74:	J	75:	K	76:	L	77:	M	78:	N	79:	O
80:	P	81:	Q	82:	R	83:	S	84:	T	85:	U	86:	V	87:	W
88:	X	89:	Y	90:	Z	91:	[	92:	\	93:	]	94:	^	95:	_
96:	`	97:	a	98:	b	99:	c	100:	d	101:	e	102:	f	103:	g
104:	h	105:	i	106:	j	107:	k	108:	l	109:	m	110:	n	111:	o
112:	p	113:	q	114:	r	115:	s	116:	t	117:	u	118:	v	119:	w
120:	x	121:	y	122:	z	123:	{	124:	|	125:	}	126:	~	127:	□

题 5.3.7 参考答案：

分析：由题义可知，一个字节可以代表 8 个集合元素，如果采用无符号短整型数组来存储的话，则 N 个数组元素可以存放 N * 16 个集合元素，因为无符号短整型占用 2 个字节。所以，如果集合元素取值范围从 0 到 setrange - 1 的话，则所需的数组大小 arraysize = (setrange+15)≫4，也就是(setrange - 1)/16 + 1，其中 setrange 为集合中元素最多时的个数。

由于集合的大小是在创建对象时才确定，故无法预先确定所需的数组大小，所以程序中采用了动态内存分配。为了确定集合元素在数组中的对应位置，在类定义中提供了 2 个私有成员函数 ArrayIndex 和 BitMask。其中前者用于确定集合元素位于数组中的哪个元素中，而后者用于确定在数组元素中的哪个位上。要知道集合元素在哪个数组元素中，只需将该集合元素的值除以 16 即可，商对应数组下标，而余数对应数组元素中的位。集合元素对

应该数组元素哪个位可以用位掩码表示，也就是对应位用 1 表示，而其余位都用 0 表示。为此只需将 1 左移余数位，即 1<< (elt% 15)。

向集合插入元素，就是在数组元素的对应位上置 1，也就是用掩码跟对应的数组元素进行位或运算，即 member[ArrayIndex(elt)] |= BitMask(elt)。

从集合删除元素，就是在数组元素的对应位上置 0，也就是用掩码的反码跟对应的数组元素进行位与运算，即 member[ArrayIndex(elt)] &= ~BitMask(elt)。

集合常见的操作有并、交、差、补、属于等。其中属于操作用成员函数 IsMember 实现，而前 4 个操作通过重载运算符(+、*、-、~)来实现。其中并、交、差 3 个运算为双目运算，要求参与运算的 2 个集合大小相同，而补运算为单目运算。

并运算通过对 2 个集合的对应位进行或运算实现；

交运算通过对 2 个集合的对应位进行与运算实现；

补运算通过按位求反实现；

差运算可以通过补运算和交运算实现，即 X - Y 等价于 X * (~Y)

以上运算的结果是一个集合，因此返回值为对象。当函数返回值为对象时会调用复制构造函数，如果类中没有提供自己的复制构造函数，则调用系统提供的默认复制构造函数。而对于指针数据成员，如果调用默认的复制构造函数，则只会复制指针值，而不会复制指针所指的对象，为此类中提供了自定义的复制构造函数。同理，为了能够进行正确的赋值运算，类中也重载了赋值运算符。

为了方便输入输出，类中还重载了提取和插入运算符。

输入时按{,,,}形式输入，程序中用输入流对象的成员函数 get 循环取出每一个字符，判断该字符是否是数字，如果是，则将该字符放回流，然后将整个数字串提取出来，插入集合，再用 get 循环提取后续字符，当遇到数字时，再采用前述方法将元素插入集合，依次进行，直到遇到字符}为止。

输出时也按{,,,}形式输出，采用循环依次判断 0 到 setrange - 1 数值范围内的每一个数值是否属于集合，是，则将该数值输出。

参考程序：

```
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>
class Set
{
    //max number of elements in the set
    int setrange;
    //number of bytes in bit array and the array pointer
    int arraysize;
    unsigned short *member;
    //implement the mapping of set elements to bits
    //within 16-bit short integers
    int ArrayIndex(const int& elt) const;
```

```
    unsigned short BitMask(const int& elt) const;
public:
    //constructor. create an empty set
    Set(int setrange);
    //copy constructor
    Set(Set& x);
    //destructor
    ~Set(void);
    //assignment operator
    Set& operator=(const Set& rhs);
    //element membership
    int IsMember(const int& elt);
    //set union
    Set operator+(const Set& x)const;
    //set intersection
    Set operator*(const Set& x)const;
    //set difference
    Set operator-(const Set& x)const;
    //complement
    Set operator~()const;
    //set insertion/deletion operations
    void Insert(const int& elt);
    void Delete(const int& elt);
    //set stream I/O operators
    friend istream& operator>>(istream& is,Set& x);
    friend ostream& operator<<(ostream& os,Set& x);
};

//determine the index of the array element
//containing the bit representing elt
int Set::ArrayIndex(const int& elt) const
{
    return elt>>4;
}

//create an unsigned short value with a 1
//in the bit position of elt
unsigned short Set::BitMask(const int& elt)const
{
    //use & to find remainder after dividing by 16.0
    //stays in right-most bit,15 goes on far left
    return 1<<(elt&15);
}
```

```
Set::Set(int sz):setrange(sz)
{
    //number of unsigned shorts needed to hold set elements
    arraysize=(setrange+15)>>4;
    //allocte the array
    member=new unsigned short[arraysize];
    if(! member)
    {
        cout<<"out of memory."<<endl;
        exit(0);
    }
    //create an empty set by setting all bits to 0
    for(int i=0;i<arraysize;i++)
        member[i]=0;
}

Set::Set(Set& x)
{
    setrange=x.setrange;
    arraysize=x.arraysize;
    //allocte the array
    member=new unsigned short[arraysize];
    if(! member)
    {
        cout<<"out of memory."<<endl;
        exit(0);
    }
    for(int i=0;i<arraysize;i++)
        member[i]=x.member[i];
}

Set::~Set(void)
{
    delete[]member;
}

Set& Set::operator =(const Set& rhs)
{
    //the sets must have the same range
    if(setrange!=rhs.setrange)
    {
        cout<<"sets different size."<<endl;
        exit(0);
    }
```

```
    for(int i=0;i<arraysize;i++)
        member[i]=rhs.member[i];
    return *this;
}

int Set::IsMember(const int& elt)
{
    //is elt in range 0 to setrange-1?
    if(elt<0||elt>=setrange)
    {
        cout<<"invalid member reference."<<endl;
        exit(0);
    }
    //return the bit corresponding to elt
    return member[ArrayIndex(elt)] & BitMask(elt);
}

Set Set::operator +(const Set& x)const
{
    //the sets must have the same range
    if(setrange!=x.setrange)
    {
        cout<<"sets different size."<<endl;
        exit(0);
    }
    //form the union in tmp
    Set tmp(setrange);
    //each array element of tmp is the bitwise
    //OR of the current object and x
    for(int i=0;i<arraysize;i++)
        tmp.member[i]=member[i] | x.member[i];
    //return the union
    return tmp;
}

Set Set::operator *(const Set& x)const
{
    //the sets must have the same range
    if(setrange!=x.setrange)
    {
        cout<<"sets different size."<<endl;
        exit(0);
    }
    //form the intersection in tmp
```

```
    Set tmp(setrange);
    //each array element of tmp is the bitwise
    //AND of the current object and x
    for(int i=0;i<arraysize;i++)
        tmp.member[i]=member[i] & x.member[i];
    //return the intersection
    return tmp;
}

Set Set::operator ~()const
{
    Set tmp(setrange);
    for(int i=0;i<arraysize;i++)
        tmp.member[i]=~member[i];
    return tmp;
}

Set Set::operator -(const Set& x)const
{
    //the sets must have the same range
    if(setrange!=x.setrange)
    {
        cout<<"sets different size."<<endl;
        exit(0);
    }
    //form the intersection in tmp
    Set tmp(setrange);
    tmp= *this * (~x);
    return tmp;
}

void Set::Insert(const int& elt)
{
    //is elt in range 0 to setrange-1?
    if(elt<0||elt>=setrange)
    {
        cout<<"invalid member reference."<<endl;
        exit(0);
    }
    //set bit corresponding to elt
    member[ArrayIndex(elt)] |= BitMask(elt);
}

void Set::Delete(const int& elt)
```

```
{
    //is elt in range 0 to setrange-1?
    if(elt<0||elt>=setrange)
    {
        cout<<"invalid member reference."<<endl;
        exit(0);
    }
    //reset bit corresponding to elt
    member[ArrayIndex(elt)] &= ~BitMask(elt);
}

istream& operator>>(istream& is,Set& x)
{
    int elt;
    char c;
    cout<<"input elements for a set in form of {,,,}"<<endl;
    while(is.get(c) && c!='}')
    {
        if(isdigit(c))
        {
            is.putback(c);
            is>>elt;
            x.Insert(elt);
        }
    }
    return is;
}

ostream& operator<<(ostream& os,Set& x)
{
    bool first(true);
    os<<"{";
    for(int i=0;i<x.setrange;i++)
        if(x.IsMember(i))
            if(first)
            {
                first=false;
                os<<i;
            }
            else
                os<<","<<i;
    os<<"}"<<endl;
    return os;
}
```

```
void main()
{
    Set A(10),B(10),C(10);
    cin≫A≫B;
    cout≪"A:"≪A≪"B:"≪B;
    C=A+B;
    cout≪"the union of A and B is: "≪C;
    C=A*B;
    cout≪"the intersection of A and B is: "≪C;
    C=A-B;
    cout≪"the difference of A and B is: "≪C;
}
```

测试结果：

```
    input elements for a set in form of {,,,}
{1,2,3,4,5}
input elements for a set in form of {,,,}
{4,5,6,7,8}
A:{1,2,3,4,5}
B:{4,5,6,7,8}
the union of A and B is: {1,2,3,4,5,6,7,8}
the intersection of A and B is: {4,5}
the difference of A and B is: {1,2,3}
```

题 5.3.8 参考答案：

类的声明：

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

enum ErrorType
{
    invalidArraySize,memoryAllocationError,indexOutofRange
};

char* errorMsg[]=
{
    "数组大小非法","内存分配错误","引用了非法下标："
};

class Array
{
```

```
private:
    //动态分配存储单元用于包括 size 个元素
    int* alist;
    int size;

    //错误处理方法
    void Error(ErrorType et,int badIndex=0)const;

public:
    //构造函数和析构函数
    Array(int sz=0);
    Array(const Array& x);
    ~Array(void);

    //赋值运算符,下标运算符,以及指针转换
    Array& operator=(const Array& rhs);
    int& operator [](int i);
    operator int*(void) const;

    //有关数组大小的操作
    int ListSize(void) const;      //读取数组的大小
    void Resize(int sz);           //改变数组的大小
};
```

分析:

该程序中假定数组用于存放整型元素。

使用重载的下标运算符[ ]以及类型转换运算符,使得 Array 对象具有类似于一般数组的功能。而重载的赋值运算符使得一个对象可以向另一个对象赋值。一般的数组是不能进行整体赋值的。

Resize 方法允许改变数组的大小。如果新的大小 sz 大于当前大小 size,则原有数组元素保留,并在后面附加新的空间。如果新的大小 sz 小于当前大小 size,则保留原数组的前 sz 个元素,余下的元素被删除。

类的实现:

```
//错误处理函数
void Array::Error(ErrorType et,int badIndex)const
{
    cout<<errorMsg[et];
    if(et==indexOutofRange)cout<<badIndex;
    cout<<endl;
    exit(0);
}
```

```
//构造函数
Array::Array(int sz)
{
    //检查数组大小是否合法
    if(sz<=0)
        Error(invalidArraySize);
    //给 size 赋值,并动态分配内存
    size=sz;
    alist=new int[size];
    //确定内存分配是否成功
    if(! alist)
        Error(memoryAllocationError);
}

//复制构造函数
Array::Array(const Array& x)
{
    //将对象 x 的 size 赋值给当前对象
    int n=x.size;
    size=n;

    //为当前对象分配内存,并进行错误检查
    alist=new int[n];      //动态分配内存
    if(! alist)
        Error(memoryAllocationError);

    //将对象 x 的数据项复制给当前对象
    int * srcptr=x.alist;
    int * destptr=alist;
    while(n--)
        *destptr++=*srcptr++;
}

//析构函数
Array::~Array(void)
{
    delete[] alist;
}

//重载赋值运算符
Array& Array::operator=(const Array& rhs)
{
    if(size!=rhs.size)
    {
```

```
        delete[] alist;
        size=rhs.size;
        alist=new int[size];
        if(!alist)
            Error(memoryAllocationError);
    }

    int n=size;
    int* srcptr=rhs.alist;
    int* destptr=alist;
    while(n--)
        *destptr++=*srcptr++;
    return *this;
}

//下标运算符重载
int& Array::operator [](int i)
{
    //进行数组边界检查
    if(i<0 || i>size-1)
        Error(indexOutofRange,i);
    //返回对应数组元素
    return alist[i];
}

//将数组对象转换为指针类型
Array::operator int*(void) const
{
    return alist;
}

//读取数组的大小
int Array::ListSize(void) const
{
    return size;
}

//改变数组的大小
void Array::Resize(int sz)
{
//测试新的大小,如果 size<=0,则结束程序
if(sz<=0)
    Error(invalidArraySize);
    //如果大小没有改变,则什么也不做
```

```
    if(sz==size)
        return;

    //申请新的内存,并判断是否成功
    int* newlist=new int[sz];
    if(!newlist)
        Error(memoryAllocationError);


    int n=(sz<=size)? sz:size;

    //从旧的数组复制 n 个元素到新的数组
    int* srcptr=alist;
    int* destptr=newlist;
    while(n--)
    *destptr++=*srcptr++;

    //删除旧的数组
    delete[] alist;

    //将 alist 指向新的数组,并更新数组大小
    alist=newlist;
    size=sz;
}
```

测试主函数

```
void main()
{
    Array a(10),b(10);
    cout<<"数组 A 的元素:";
    for(int i=0;i<10;i++)
    {
        a[i]=i+1;
        cout<<a[i]<<"  ";
    }
    cout<<endl;
    b=a;            //重载的赋值运算符"=",使数组 a 向数组 b 进行整体赋值
    cout<<"数组 B 的元素:";
    for(i=0;i<10;i++)
        cout<<b[i]<<"  ";
    cout<<endl;
}
```

测试结果:

```
数组 A 的元素:1    2    3    4    5    6    7    8    9    10
数组 B 的元素:1    2    3    4    5    6    7    8    9    10
```

题 5.3.9 参考答案:

```
#include <iostream.h>
#include <stdlib.h>
enum ErrorType
{
    invalidArraySize,memoryAllocationError,indexOutofRange
};
char* errorMsg[]=
{
    "数组大小非法","内存分配错误","引用了非法下标:"
};

template<class T>
class Array
{
private:
    //动态分配存储单元用于包括 size 个元素
    T* alist;
    int size;

    //错误处理方法
    void Error(ErrorType et,int badIndex=0)const;
public:
    //构造函数和析构函数
    Array(int sz=0);
    Array(const Array<T>& x);
    ~Array(void);
    //赋值运算符,下标运算符,以及指针转换
    Array& operator=(const Array<T>& rhs);
    T& operator [](int i);
    operator T*(void) const;
    //有关数组大小的操作
    int ListSize(void) const;      //读取数组的大小
    void Resize(int sz);           //改变数组的大小
};

//错误处理函数
template<class T>
void Array<T>::Error(ErrorType et,int badIndex)const
{
```

```
    cout<<errorMsg[et];
    if(et==indexOutofRange)cout<<badIndex;
    cout<<endl;
    exit(0);
}

//构造函数
template<class T>
Array<T>::Array(int sz)
{
    //检查数组大小是否合法
    if(sz<=0)
        Error(invalidArraySize);
    //给 size 赋值,并动态分配内存
    size=sz;
    alist=new T[size];
    //确定内存分配是否成功
    if(! alist)
        Error(memoryAllocationError);
}

//复制构造函数
template<class T>
Array<T>::Array(const Array& x)
{
    //将对象 x 的 size 赋值给当前对象
    int n=x.size;
    size=n;
    //为当前对象分配内存,并进行错误检查
    alist=new T[n];     //动态分配内存
    if(! alist)
        Error(memoryAllocationError);
    //将对象 x 的数据项复制给当前对象
    int* srcptr=x.alist;
    int* destptr=alist;
    while(n--)
        *destptr++=*srcptr++;
}

//析构函数
template<class T>
Array<T>::~Array(void)
{
    delete[] alist;
```

```
}

//重载赋值运算符
template<class T>
Array<T>& Array<T>::operator=(const Array& rhs)
{
    if(size!=rhs.size)
    {
        delete[] alist;
        size=rhs.size;
        alist=new T[size];
        if(!alist)
            Error(memoryAllocationError);
    }
    int n=size;
    int* srcptr=rhs.alist;
    int* destptr=alist;
    while(n--)
        *destptr++=*srcptr++;
    return *this;
}

//下标运算符重载
template<class T>
T& Array<T>::operator[](int i)
{
    //进行数组边界检查
    if(i<0 || i>size-1)
        Error(indexOutofRange,i);
    //返回对应数组元素
    return alist[i];
}

//将数组对象转换为指针类型
template<class T>
Array<T>::operator T*(void) const
{
    return alist;
}

//读取数组的大小
template<class T>
int Array<T>::ListSize(void) const
{
```

```
    return size;
}

//改变数组的大小
template<class T>
void Array<T>::Resize(int sz)
{
    //测试新的大小,如果 size<=0,则结束程序
    if(sz<=0)
        Error(invalidArraySize);
    //如果大小没有改变,则什么也不做
    if(sz==size)
        return;

    //申请新的内存,并判断是否成功
    int * newlist=new T[sz];
    if(!newlist)
        Error(memoryAllocationError);

    int n=(sz<=size)? sz:size;

    //从旧的数组复制 n 个元素到新的数组
    int * srcptr=alist;
    int * destptr=newlist;
    while(n--)
        *destptr++=*srcptr++;

    //删除旧的数组
    delete[] alist;

    //将 alist 指向新的数组,并更新数组大小
    alist=newlist;
    size=sz;
}
```

测试主函数

```
void main()
{
    Array<int> A(10),B(5);
    for(int i=0;i<10;i++)
        A[i]=i;
    cout<<"A:";
    for(i=0;i<10;i++)
        cout<<A[i];
    cout<<endl;
```

```
    B=A;
    cout<<"B:";
    for(i=0;i<10;i++)
        cout<<B[i];
    cout<<endl;
    Array<int> C(B);
    cout<<"C:";
    for(i=0;i<10;i++)
        cout<<C[i];
    cout<<endl;
}
```

测试结果：

```
A: 0123456789
B: 0123456789
C: 0123456789
```

题 5.3.10 参考答案：

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

class Time
{
protected:
    int hours,minutes,seconds;
public:
    Time(int h,int m,int s)
    {
        hours=h;
        minutes=m;
        seconds=s;
    }
    Time(time_t systime)
    {
        struct tm * tim=localtime(&systime);
        hours=tim->tm_hour;
        minutes=tim->tm_min;
        seconds=tim->tm_sec;
    }
    virtual void display()
    {
```

```
        cout<< hours<<":"<<minutes<<":"<<seconds;
    }
};

class Date
{
protected:
    int month,day,year;
public:
    Date(int m,int d,int y)
    {
        month=m;
        day=d;
        year=y;
    }
    Date(time_t systime)
    {
        struct tm * tim=localtime(&systime);
        month=tim->tm_mon+1;
        day=tim->tm_mday;
        year=tim->tm_year+1900;
    }
    virtual void display()
    {
        cout<<month<<"/"<<day<<"/"<<year;
    }
};

class FileStamp:public Time, public Date
{
protected:
    char filename[15];
public:
    FileStamp(char * fn,int mm,int dd,int yy,int hr,int mn,int sc)
:Time(hr,mn,sc),Date(mm,dd,yy)
    {
        strcpy(filename,fn);
    }
    FileStamp(char * fn,time_t systime):Time(systime),Date(systime)
    {
        strcpy(filename,fn);
    }
    void display();
};
```

```
void FileStamp::display()
{
    cout<<filename<<' ';
    Date::display();
    cout<<' ';
    Time::display();
}

void main()
{
    time_t systime=time(NULL);
    FileStamp fs("Cplusplus.dat",systime);
    fs.Date::display();
    cout<<endl;
    fs.Time::display();
    cout<<endl;
    fs.display();
    cout<<endl;
}
```

测试结果：

```
  3/28/2003
16:46:1
Cplusplus.dat 3/28/2003 16:46:1
```

题 5.3.11 参考答案：

分析：

本题涉及类层次结构中赋值兼容的问题。赋值兼容是指在需要基类对象的任何地方都可以使用公有派生类的对象来替代，包括以下一些情况：

派生类的对象可以赋值给基类对象。

派生类的对象可以初始化基类的引用。

派生类对象的地址可以赋值给指向基类的指针。

在替代之后，派生类对象就可以作为基类的对象使用，但只能使用从基类继承的成员。

当涉及多基重复继承、无虚基类时，由于会合点后的派生类可以从多条路径继承间接基类的成员，因而在派生类中就有多个间接基类的成员的备份，在派生类中引用这些成员就会造成二义性，克服这种二义性的方法就是对指针显式地指明全路径，或将指针先强制转换到不会产生二义性的基类。

解：

（1）无虚基类时

```
  pa = pb;          //正确,指向派生类的指针可以赋值给指向直接基类的指针
//  pb = pa;        //错误,基类指针必须经强制转换才能赋值给派生类指针
  pa = (A *)pb;     //正确
  pb = (B *)pa;     //正确,因为经过了强制类型转换
  a = b;            //正确,派生类对象可以赋值给直接基类对象
//  b = a;          //错误,基类对象不能赋值给派生类对象,因为派生类中含有基类没有的成员
                    //这部分成员没法通过基类对象进行赋值
  pa = pc;          //正确,因为从 C 到 A 只有一条继承路径
//  pc = pa;        //错误,必须经过强制类型转换

  pa = (A *)pc;     //正确
  pc = (C *)pa;     //正确

  a = c;            //正确
//  c = a;          //错误,基类对象不能赋值给派生类对象

  pb = pc;          //正确
//  pc = pb;        //错误,需要经过强制类型转换

  pb = (B *)pc;     //正确
  pc = (C *)pb;     //正确

  b = c;            //正确
//  c = b;          //错误

//  pb = pe;        //错误,因为从 E 到 B 有两条路径,所以必须显式地指明全路径
                    //或将指针先强制转换到不会产生二义性的基类
//  pe = pb;        //错误,需经强制类型转换,且必须给出全路径

//  pb = (B *)pe;   //错误,必须给出全路径,即 pb = (B *)(C *)pe;
                    //或 pb = (C *)pe;
//  pe = (E *)pb;   //错误,必须给出全路径

//  b = e;          //错误,不能给间接重复继承的基类对象赋值
//  e = b;          //错误,基类对象不能赋值给派生类对象

//  pa = pe;        //错误,需强制转换类型,直到不产生二义性
//  pe = pa;        //错误,需强制转换类型

//  pa = (A *)pe;   //错误,需强制转换类型,直到不产生二义性
                    //如:pa = (A *)(C *)pe;
//  pe = (E *)pa;   //错误,需强制转换类型,如 pe = (E *)(D*)(B*)pa;

//  a = e;          //错误,不能给间接重复继承的基类对象赋值
```

```
//  e = a;          //错误,基类对象不能赋值给派生类对象
```

对于含有公共虚基类的类层次结构,使用虚基类,在它的几条派生路径的汇合处,只产生其一个拷贝。所以在该汇合处可以从任何一条路径访问虚基类的成员,并且都使用相同的虚基类实例。因此,可以进行如下的类变换:

① 派生类对象的地址可以直接赋给间接公虚基类的指针,并且不需进行强制类型转换;

② 一个虚基类的引用,可以引用一个派生类的对象;

③ 但是,相反的转换是不允许的,即使用指定路径的强制类型转换也不可。

(2) A 为虚基类

E 重复继承 B 和 A,但由于 A 是虚基类,所以在 E 中只产生一个拷贝,而 B 不是虚基类,所以在 E 中产生两个拷贝。

```
  pa = pb;          //正确,派生类指针可以赋给直接基类的指针
//  pb = pa;          //错误,基类指针不能赋给派生类指针
  pa = (A *)pb;     //正确
//  pb = (B *)pa;     //错误,因为 A 是虚基类,对虚基类来说相反的转换是不允许的
  a = b;            //正确
//  b = a;            //错误

  pa = pc;          //正确,因为 C 中只有 A 的一个拷贝
//  pc = pa;          //错误,因为 A 是虚基类,即使经强制类型转换也不可以

  pa = (A *)pc;     //正确
//  pc = (C *)pa;     //错误

  a = c;            //正确
//  c = a;            //错误

  pb = pc;          //正确
//  pc = pb;          //错误,需要经强制类型转换

  pb = (B *)pc;     //正确
  pc = (C *)pb;     //正确,因为 B 不是虚基类,可以经强制转换赋给派生类指针

  b = c;            //正确
//  c = b;            //错误,基类对象不能赋给派生类对象

//  pb = pe;          //错误,由于 B 不是虚基类,E 中有 B 的 2 个拷贝,会导致二义性
//  pe = pb;          //错误,需要经强制类型转换

//  pb = (B *)pe;     //错误,会导致二义性,
                      //应转换到不产生二义性为止,如:pb = (D *)pe
```

```
//  pe = (E *)pb;  //错误,应给出全路径,如:pe = (E *)(C *)pb

//  b = c;         //错误,不能给间接重复继承的基类对象赋值
//  e = b;         //错误,基类对象不能赋值给派生类对象

  pa = pe;         //正确,因为A是虚基类,在E中只产生一个拷贝
//  pe = pa;       //错误,因为A是虚基类,哪怕经全路径强制类型转换也不行

  pa = (A *)pe;    //正确
//  pe = (E *)pa;  //错误

  a = e;           //正确
//  e = a;         //错误,基类对象不能赋值给派生类对象
```

(3) B为虚基类

E重复继承B和A,由于B是虚基类,所以A和B在E中都只产生一个拷贝。

```
  pa = pb;         //正确
//  pb = pa;       //错误,需要经强制类型转换
  pa = (A *)pb;    //正确
  pb = (B *)pa;    //正确
  a = b;           //正确
//  b = a;         //错误

  pa = pc;         //正确
//  pc = pa;       //错误,因为C通过虚基类B间接继承A,所以哪怕通过全路径转换也不行

  pa = (A *)pc;    //正确
//  pc = (C *)pa;  //错误

  a = c;           //正确
//  c = a;         //错误

  pb = pc;         //正确
//  pc = pb;       //错误,因为B是虚基类,哪怕经强制转换也不行

  pb = (B *)pc;    //正确
//  pc = (C *)pb;  //错误

  b = c;           //正确
//  c = b;         //错误

  pb = pe;         //正确,因为E中只有B的一个拷贝
```

```
//  pe = pb;          //错误,因为B是虚基类,哪怕经全路径转换也不行

  pb = (B *)pe;       //正确
//  pe = (E *)pb;     //错误

  b = e;              //正确,因为E中只有B的一个拷贝
//  e = b;            //错误

  pa = pe;            //正确,因为E中只有A的一个拷贝
//  pe = pa;          //错误,因为当中经过了虚基类B,哪怕经全路径转换也不行

  pa = (A *)pe;       //正确
//  pe = (E *)pa;     //错误

  a = e;              //正确,因为E中只有A的一个拷贝
//  e = a;            //错误
```

题 5.3.12 参考答案：

分析：

动物园中有 20 个分隔栏，这 20 个分隔栏可以用教材《C++ 程序开发教程》中的题 5.13 中定义的通用数组类的对象表示。由于数组中要求所有数组元素的数据类型应该一致，而这儿要求一半用于养猫，一半用于养狗，为此可以定义一个宠物类 Pet，作为猫和狗的基类。由于派生类的对象可以用指向基类型的指针来指向它们，因而数组元素可以采用指向基类的指针类型这一统一的数据类型。

对动物园的管理可以假设有收容宠物和领养宠物等。收容宠物就是将宠物关入笼子，如果笼子已满，则不能再收容，例如，如果收容了 10 只猫，现在想再收容 1 只猫，则即使还有空余笼子也不能再收容猫了，因为剩余的笼子是养狗的。

为了能够统计动物园内现有猫狗的数量，在动物园类 Zoo 中有两个数据成员 numCat 和 numDog，用于分别维护猫和狗的数量。

如果有人来动物园领养宠物，则将该宠物从数组中删除。删除时按笼子号删除，为了确定删除的是猫，还是狗，在 Pet 类中有一个数据成员 pettype 宠物类型，其取值为枚举值 cat 和 dog。知道了删除的宠物类型，就可以对数据成员 numCat 和 numDog 进行维护了。另外为了便于数组维护，假定宠物总是从 0 号笼子开始连续存放。所以删除某个笼子中的宠物后，应将其后的所有宠物依次往前移动一个笼子。

最后当程序结束时，也就是动物园被销毁时，应将其中收养的所有宠物都释放掉。而数组所占存储空间的回收则由数组类中的析构函数负责。

解：

通用数组类的定义见题 5.13(见教材)。

宠物的类层次结构：

```
enum pet{cat,dog};

class Pet      //宠物的基类
{
    pet pettype;
public:
    Pet(pet pt)
    {
        pettype=pt;
    }
    pet gettype()
    {
        return pettype;
    }
    virtual void speak()=0;
};

class Cat:public Pet
{
public:
    Cat():Pet(cat)
    {}
    void speak()
    {
        cout<<"I am a cat."<<endl;
    }
};

class Dog:public Pet
{
public:
    Dog():Pet(dog)
    {}
    void speak()
    {
        cout<<"I am a dog."<<endl;
    }
};
```

动物园类的定义：

```
class Zoo
{
    int numCat,numDog;
```

```
    Array<Pet * > cage;      //豢养宠物的 20 个笼子
public:
    Zoo():numCat(0),numDog(0),cage(20)
    {}
    void Insert(pet p)          //收容宠物
    {
        if(p==cat)
        {
            if(numCat>=10)
            {
                cout<<"the cage for cat is full."<<endl;
                exit(0);
            }
            cage[numCat+numDog]=new Cat;
            numCat++;
        }
        if(p==dog)
        {
            if(numDog>=10)
            {
                cout<<"the cage for dog is full."<<endl;
                exit(0);
            }
            cage[numCat+numDog]=new Dog;
            numDog++;
        }
    }
    void Delete(int cageno)     //按笼子号释放宠物
    {
        if(cageno>=numCat+numDog)
        {
            cout<<"cage: "<<cageno<<" has no pet."<<endl;
            exit(0);
        }
        if(cage[cageno] ->gettype()==cat)
            numCat--;
        else
            numDog--;
        delete cage[cageno];
        //将后面笼子中的宠物依次往前移一格
        for(int i=cageno;i<numCat+numDog;i++)
            cage[i]=cage[i+1];
    }
```

```
    //得到第 cageno 号笼子中的宠物
    Pet * getPet(int cageno)
    {
        if(cageno>=numCat+numDog)
        {
            cout<<"cage: "<<cageno<<" has no pet."<<endl;
            exit(0);
        }
        return cage[cageno];
    }

    //返回动物园中宠物的个数
    int numPetsinZoo()
    {
        return numCat+numDog;
    }
    int numCatsinZoo()
    {
        return numCat;
    }
    int numDogsinZoo()
    {
        return numDog;
    }

    //动物园破坏的时候,释放所有园中宠物
    ~Zoo()
    {
        for(int i=0;i<numCat+numDog;i++)
            delete cage[i];
    }
};
```

测试程序(模拟动物园的管理):

```
void main()
{
    Zoo zoo;
    for(int i=0;i<4;i++)
        zoo.Insert(cat);
    for(i=0;i<10;i++)
        zoo.Insert(dog);
    cout<<"Now the zoo has "<<zoo.numCatsinZoo()<<" cats. ";
```

```
    cout<<"and "<<zoo.numDogsinZoo()<<" dogs."<<endl;
//  zoo.Insert(dog);
//  zoo.Delete(14);
    zoo.Delete(2);
    zoo.Delete(1);
    zoo.Delete(3);
    cout<<"After delete some pets, the zoo now has "<<zoo.numCatsinZoo()<<" cats. ";
    cout<<"and "<<zoo.numDogsinZoo()<<" dogs."<<endl;
    zoo.Insert(cat);
    zoo.Insert(dog);
    cout<<"After insert some pets, the zoo now has "<<zoo.numCatsinZoo()<<" cats. ";
    cout<<"and "<<zoo.numDogsinZoo()<<" dogs."<<endl;
    cout<<"The pet in cage:"<<zoo.numPetsinZoo()-1<<" speaks ";
    zoo.getPet(zoo.numPetsinZoo()-1)->speak();
    cout<<"The pet in cage:"<<zoo.numPetsinZoo()-2<<" speaks ";
    zoo.getPet(zoo.numPetsinZoo()-2)->speak();
}
```

测试结果：

```
Now the zoo has 4 cats. and 10 dogs.
After delete some pets, the zoo now has 2 cats. and 9 dogs.
After insert some pets, the zoo now has 3 cats. and 10 dogs.
The pet in cage:12 speaks I am a dog.
The pet in cage:11 speaks I am a cat.
```

题 5.3.13 参考答案：

（1）派生类的构造函数没有在初始化表中调用基类的构造函数，这就意味着调用基类中默认的构造函数，也就是没有参数的构造函数，或参数都有默认值的构造函数。而现在基类中没有提供默认构造函数。

（2）派生类没有提供构造函数，也就意味着派生类使用的是系统提供的默认构造函数，而这默认的构造函数是不会去调用基类的构造函数的。

题 5.3.14 参考答案：

输出结果：

```
Base1 constructor called.
Base2 constructor called.
Derived class constructor called.
Base1 constructor called.
Base2 constructor called.
Base2 destructor called.
Base1 destructor called.
Derived class destructor called.
```

```
Base2 destructor called.
Base1 destructor called.
```

说明：

当派生类对象创建时，构造函数的调用次序是：先调用基类的构造函数，当涉及多基派生时，调用顺序按照它们被继承时声明的顺序(从左到右)，而不是派生类构造函数中初始化表中的顺序；再执行派生类的构造函数体中的内容。

而析构函数的执行顺序刚好与构造函数的执行顺序相反。

题 5.3.15 参考答案：

(1) OBJ.Base::F();

(2) OBJ.F();

(3) 调用派生类中的成员函数 G(float x)，将 20 转换为 float 型，传递给 x。因为派生类如果有与基类同名的成员函数，则派生类的函数将覆盖掉基类的同名函数，而不是构成函数重载。如果要引用基类中的函数 G，则应为：OBJ.Base::G(20);

题 5.3.16 参考答案：

输出结果：

```
1
Base Class
2
1
0
Base
```

说明：

本题主要说明在派生类中怎样调用同名基类成员函数。

B.output()调用基类函数输出：

```
1
Base Class
```

D.output()：调用类 Derived2 中的 output()函数，首先输出自己数据成员的值 2，接着执行 Derived1::output()：调用类 Derived1 中的 output()函数，而该函数首先输出自己数据成员的值 1，接着执行 Base::output()；调用类 Base 中的 output()函数，输出 0 和 Base。

题 5.3.17 参考答案：

输出结果：

```
7
2
3  5
2  4
```

```
2  4
0  1
7
2  3
```

说明：

本题主要说明动态多态性的效果，函数 Announce1 采用静态绑定，而函数 Announce2 和 Announce3 采用动态绑定。因此 Announce1(A)调用基类的函数输出 7，Announce1(C)；也调用基类的函数输出 2，Announce2(B)；调用派生类的函数输出 35，Announce3(&C)；调用派生类的函数输出 24，p=&C；p->Identify()采用动态绑定调用派生类的函数输出 24，而对于 arr[i]->Identify()；也采用动态绑定，当 i 等于 0 和 2 时，调用派生类的函数，当 i=1 时，调用基类的函数。

题 5.3.18 参考答案：

当一个类涉及动态内存分配时必须定义析构函数。如果这个类被作为基类，则析构函数应该声明为虚函数。如果不是虚函数，则引用派生类对象的基类不会执行派生类的析构函数。下面的例子说明了这个问题。假定类 BaseCL 的构造函数动态分配了具有 7 个整型元素的数组，而析构函数用来释放数据。

```
class BaseCL
{
        ...
    public:
        BaseCL(...);        //分配 7 个元素的数组
        ~BaseCL(void);      //释放数据(非虚函数)
};
```

类 DerivedCL 继承类 BaseCL 并执行相同的操作。

```
class DerivedCL : public BaseCL
{
        ...
    public:
        DerivedCL(...);
        ~DerivedCL(void);
};
```

假定 p 是 BaseCL 指针，指向动态创建的 DerivedCL 对象，如果我们调用 delete 函数：

```
BaseCL * p= new DerivedCL();
delete p;
```

则由派生类生成的动态数据不会被销毁。如果基类析构函数声明为虚函数，则根据动态绑定，调用派生类析构函数。在这种情况下，基类析构函数也会被调用，只不过要等到派生类析构函数执行后。

将基类析构函数声明为虚函数后可以保证，如果派生类有自己的析构函数，则该析构函数将被执行。

第6章 I/O流类库

6.1 编程知识

6.1.1 流类库及其结构

1. 流类库结构

流类库是C++特有的I/O库，它由一个用于I/O操作的类层次结构组成。图6.1为它的基本类层次结构。图6.1中：

(1) streambuf类：用以管理流的缓冲区；

(2) ios类：提供格式、错误检测和状态信息；

(3) ostream类：用来创建输出流；

(4) istream类：用来创建输入流；

(5) iostream类：用来创建输入输出流。

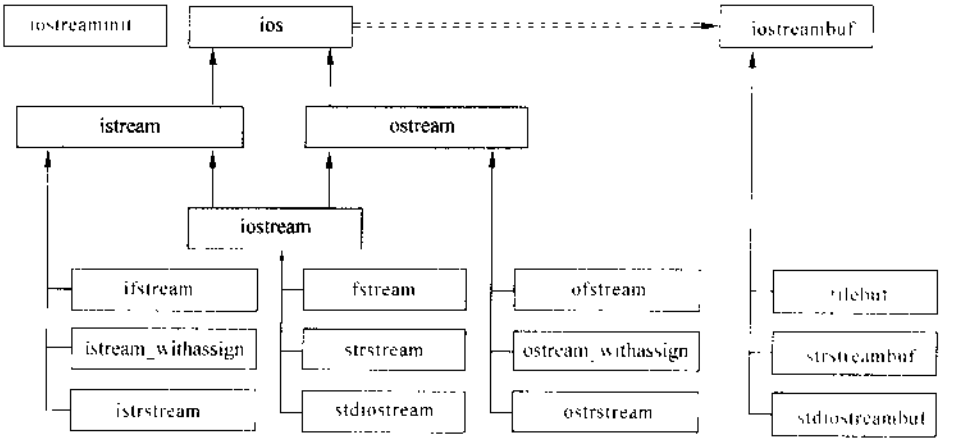

图6.1 流类库的基本层次结构

ios类是istream类与ostream类的虚基类；iostream类是istream类与ostream类的公共派生类。

在程序中要引用这些类，需要将含有它们的简要说明和声明的头文件包含在程序中。表6.1为有关类及其对应的头文件。

2. 定义流对象

在iostream.h文件中，把4个预定义的开放流声明为withassign类的对象：

```
extern istream_withassign cin;
extern ostream_withassign cout;
extern ostream_withassign cerr;
extern ostream_withassign clog;
```

其中：

(1) cin 是 istream_withassign 类的对象，称为标准输入流，默认时键盘为数据源，也可以重定向为其他设备。

(2) cout 是 ostream_withassign 类的对象，称为标准输出流，默认时显示器为数据池，也可以重定向为其他设备。

(3) cerr 和 clog 是 ostream_withassign 类的对象，称为标准错误输出流，固定关联到显示器。其中，cerr 没有缓冲，发给它的任何内容都立即被输出；clog 有缓冲，发给它的内容要先存储在缓冲区中，缓冲区满时才被输出。

表 6.1 流类库中的类及其对应的头文件

头文件	类名	含义
iostream.h	ios	流抽象类，I/O 在用户方的接口，提供基本 I/O 高层操作，包含一个指向 streambuf 的指针
	streambuf	抽象流缓冲区类，I/O 在物理设备方的接口，具有缓冲区起始地址、读写指针、缓冲区读写操作
	istream	通用输入流类，提供各种输入方式和提取操作　从缓冲区取字符
	ostream	通用输出流类，提供各种输出方式和插入操作　向缓冲区存字符
	iostream	istream 和 ostream 的组合
	istream_withassign	标准输入流类
	ostream_withassign	标准输出流类
	iostream_initifstream	预定义的流初始化类
fstream.h	ifstream	输入文件流类
	ofstream	输出文件流类
	fstream	输入输出文件流类
	filebuf	磁盘文件的流缓冲区类
strstrea.h	istrstream	输入字符串流类
	ostrstream	输出字符串流类
	strstream	输入输出字符串流类
	strstreambuf	字符串流缓冲区类
stdiostr.h	stdiostream	标准 I/O 文件的输入输出类
	stdiostreambuf	标准 I/O 文件的流缓冲区类

当程序包含头文件 iostream.h 的程序被执行时，这 4 个标准流对象（预定义流对象）的构造函数都要被自动调用一次，使这 4 个预定义流对象可以直接使用。

3. 类 istream 的定义

istream 的定义如下：

```
class istream:virtual public ios {
  public:
    istream(streambuf * );
    // 6 个 get()函数
    // 从流中输入多个字符
    istream & get(signed char * ,int,char ='\n');
    istream & get(unsigned char * ,int,char ='\n');
    istream & get(streambuf &, char ='\n');
    // 从流中输入单个字符
    istream & get(unsigned char &);
    istream & get(signed char &);
    //从流中输入下一个字符或 EOF
    int get();
    // 从流中输入字符到给定的 char *
    istream & getline(signed char * ,int,char ='\n');
    istream & getline(unsigned char * ,int,char ='\n');
    // 从流中提取给定数目的字符到一个数组
    istream & read (signed char * ,int);
    istream & read (unsigned char * ,int);
    // 不输入并返回下一个字符
    int peek();
    // 返回上次输入的字符数
    int gcount();
    // 回放字符到流中
    istream & putback(char);
    // 从输入流中越过 n 个字符，并在碰到 delim 时停止
    istream & ignore(int =1,int =EOF);
    // 在输入流中移动文件指针
    istream & seekg(streampos);
    istream & seekg(streamoff,ios::seek_dir);
    // 返回当前文件指针位置
    streampos tellg();
    int sync();
    // 提取运算符重载
    istream & operator≫(signed char * );
    istream & operator≫(signed char &);
    istream & operator≫(int &);
    //……对系统所有预定义类型都给出了"≫"运算符的重载定义
};
```

4. 类 ostream 的定义

istream 的定义如下：

```
class ostream:virtual public ios{
  public:
    ostream(streambuf * );
    ostream & flush();
    istream & seekp(streampos);
    istream & seekp(streamoff,ios::seek_dir);
    ostream & put(char);
    ostream & write (const signed char * ,int);
    ostream & write (const unsigned char * ,int);
    streampos tellg();
    // 插入运算符重载
    ostream & operator<< (short);
    ostream & operator<< (int);
    ostream & operator<< (float);
    //……对系统所有预定义类型都给出了"<<"运算符的重载定义
};
```

6.1.2 流的格式化控制

1. 使用 ios 类定义的格式化成员函数

表 6.2 为 ios 类中定义的几个格式化 I/O 成员函数。表 6.2 中，参数 flags 是格式控制字。

表 6.2 ios 类中定义的几个用于格式化 I/O 的成员函数

函数原型	功能
long ios::setf(long flags);	设置格式标志 flags
long ios::unsetf(long flags);	清除格式标志，并返回前标志
long ios::flags();	测试格式标志
long ios::flags(long flags);	设置标志 flags，并返回前标志
int ios::width(int w);	指定最小域宽 w
int ios::precision(int p);	设置浮点数的小数位数 p
char ios::fill(char ch);	指定填充字符 ch

2. 使用 I/O 操纵算子

C++ 预定义的操纵算子如表 6.3 所示。每个操纵算子有一个流引用参数并返回一个指向同一流的引用，因而可以嵌入到插入或提取表达式中来改变流的状态。这些操纵算子

的原型说明一部分(无参的)定义或声明在 iostream.h 中，一些(有参的)定义或声明在头文件 iomanip.h 中。

表 6.3 C++ 流类库定义的 I/O 操纵符

操纵算子	功能
dec	I/O：格式为十进制数据
Endl	O：插入一个换行符并刷新此流
Ends	O：插入一个空字符
Flush	O：刷新一个流
Hex	I/O：格式为十六进制数据
Oct	I/O：格式为八进制数据
Ws	I：跳过开头的空白符
resetiosflags(long f)	I/O：关闭 f 声明的格式化标志
setbase(int base)	O：设置数据的基指示符为 base
setfill(int ch)	I/O：设置填充符为 ch
setiosflags(long f)	I/O：设置 f 声明的格式化标志
setprecision(int p)	I/O：设置数据显示小数点后 p 位
setw(int w)	I/O：设置域宽为 w

3. 自定义 I/O 操纵算子

(1) 自定义无参 I/O 操纵算子

所有无参数的输入操纵算子都有如下框架：

```
istream & manip_name(istream & stream)
  {
    … // 你的代码
    return stream;
  }
```

(2) 自定义带参 I/O 操纵算子

带参 I/O 操纵算子要由类 SMANIP、OMANIP、IMANIP 或 IOMANIP 创建。这 4 个类分别对应着类 ios、ostream、istream 与 iostream，它们都定义在头文件 iomanip.h 中。

通常对输出操纵算子，可以用下面的形式定义：

```
ostream & manip_name(ostream & stream,type par)
{
    … // 代码
    return stream;
}
```

```
OMANIP(type)manip_name(type par)
{
    return OMANIP(type)(manip_name, par);
}
```

6.1.3 流的出错处理

一旦错误发生，输入符与输出符都不能改变流的状态。因此应在程序的适当点上测试流的状态，并在排除相应的错误后，清除出错状态位。表 6.4 为 ios 类中可用以测试或设置出错状态位的成员函数。

表 6.4 测试与设置出错位的成员函数

成员函数	功能
int rdstate();	返回当前出错状态字
void clear(int i=0);	置出错位为 i，例如 str.clear(ios::failbit, str.rdstate());置流 str 中的 failbit 位，不影响其他位
int good();	若出错状态字没有置位，则返回非 0
int fail();	若 failbit 被置位，则返回非 0
int bad();	若 badbit 被置位，则返回非 0
int eof();	若 istream 的 eofbit 被置位，则返回非 0
int operator! ();	若 failbit、badbit 或 hardbit 被置位，则返回非 0
int operator void * ();	若 failbit、badbit 或 hardbit 被置位，则返回 0

6.1.4 文件流操作

1. 文件流

这里讨论的是后者。

C++ 的 I/O 操作是基于流的，进行文件 I/O，必须创建与之相应的文件流：

(1) 文件输入流；

(2) 文件输出流；

(3) 文件输入输出流。

每个文件流都是相应文件流类的实例对象。C++ 系统通过对流类进一步扩展，提供了支持文件处理的能力。如图 6.2 所示，它在头文件 fstream.h 中定义了如下 3 个文件流类：

(1) ifstream；

(2) ofstream；

(3) fstream。

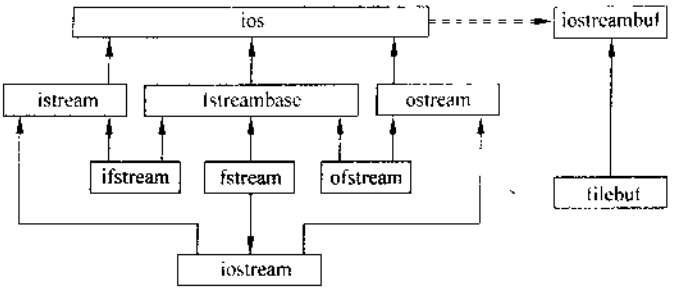

图 6.2　有关文件流的类层次结构

2. 文件的打开与关闭

流是 C++ 程序建立的一种输入输出机制，而文件是一种数据载体，要让流作用于文件，必须建立流与文件之间的连接。建立这种连接的过程称为文件的打开。

为了建立流与文件的连接，在流一方，要指定是哪种流：输出文件流，输入文件流，还是 I/O 文件流。在文件一方要指定更多的参数。

(1) 文件名：磁盘设备与控制台设备不同的一点是，一个设备中可以存储多个文件。这些文件用文件名与路径标识。文件名必须符合所使用操作系统的规定。

(2) 文件的存储位置：为了便于在磁盘上找到所需的文件，每个文件名都有一个对应的地址。

(3) 文件长度与文件指针：文件指针指示了文件操作的位置，如进行读(输入)操作时，读出的将是文件指针所指向的数据。

(4) 文件的存储模式：是文本文件，还是二进制文件。

(5) 文件的存取属性：它与操作系统有关，在 DOS 环境下，具有表 6.5 所示的 5 种属性。

表 6.5　文件的存储属性

属性代码	0	1	2	4	8
含　义	般文件	只读文件	隐藏文件	系统文件	档案文件

(6) 其他。

每一个磁盘文件都要"挂牌"标示出上述这些文件的属性，存储这些属性的信息组织称为 FCB(文件控制块)。对一个磁盘来说，磁盘中的各文件的 FCB 被集中在 0 磁道上，形成文件的目录区。

打开文件的具体操作就是把磁盘目录区中指定文件的 FCB 读到相应的文件缓冲区中，以控制文件的读写，同时建立一个相应的流对象与文件缓冲区相连接。

关闭文件就是把文件的 FCB 中已改变的内容，如文件长度，写回到磁盘中的文件目录中，以便下次使用；对流状态字清零并切断流对象与文件之间的联结。

3. 打开文件的方法

C++ 允许程序用以下方法打开文件：

(1) 用成员函数 open()在 C++ 程序中，文件的打开操作用函数 open()显式地进行。

(2) 用构造函数(创建对象)。

① 用 open()函数打开文件

open()函数是类 fstreambase 的成员函数，又在其派生类 ifstream、ofstream、fstream 中被重定义。它的原型为：

```
void open(const char * filename, int filemode, int = filebuf :: openprot);
```

其中：

- 参数 filename 为文件名，包含文件路径。
- 参数 filebuf::openprot 是文件的存取属性，一般默认。
- 参数 filemode 为文件的打开模式。文件的打开模式对文件的操作影响重大，为此 ios 定义了一些枚举常量来描述文件打开的基本模式。

```
enum open_mode {
  in          = 0x01,  // 为读打开，用 ifstream 创建的流为隐含的
  out         = 0x02,  // 为写打开，用 ofstream 创建的流为隐含的
  ate         = 0x04,  // 打开时文件指针定位到文件尾(at end)
  app         = 0x08,  // 添加模式(append)——只能写到文件尾
  trunc       = 0x10,  // 打开时清空已有文件(若设置了 ios::out，但未设置
                       // ios::ate 或 ios::app，则这种方式为隐含的)
  nocreate    = 0x20,  // 文件不存在时，打开失败(除非设置 ios::ate 或 ios::app)
  noreplace   = 0x40,  // 如果文件存在，否则打开操作失败
  binary      = 0x80   // 以二进制方式打开文件，默认时为文本方式
};
```

如果要打开一个可读/写的二进制文件，则应将第二个参数设置为

```
ios::in | ios::out | ios::binary
```

② 用构造函数打开文件，即创建与文件相联系的文件流

在 ifstream 类、ofstream 类和 fstream 类中各有一个构造函数：

```
ifstream::ifstream(char *, int = ios::in, int = filebuf::openprot);
ofstream::ofstream(char *, int = ios::out, int = filebuf::openprot);
fstream::fstream(char *, int, int = filebuf::openprot);
```

这里，3 个参数意义与前面介绍的 open()函数的 3 个参数的意义完全相同，只是在创建流对象的同时可以打开一个相应的文件。

4. 关闭文件

当与文件相连接的流对象的生命期结束时，它们的释放函数将关闭与这些流对象相连接的文件。另外，也可以使用 close()函数显式地关闭文件。close()函数的原型为

```
void fstreambase::close();
```

它是一个无参且不须指定返回值的成员函数。在 ex651 与 ex652 中也可以增加如下两个语句：

```
input.close();
output.close();
```

5. 文本文件的读/写

一个文件打开后，就与对应的流连接起来了。这时文件的读操作，就是从流中提取一个元素，文件的写操作就是向流中插入一个元素。只要建立了与文件相连接的流，向文本文件的读/写，就会像控制台 I/O 一样方便。

6. 二进制文件的读/写

C++ 的 I/O 流是基于字符的，其默认的存储模式是文本方式。二进制文件与文本文件对数据的解释及存储形式不同，要往二进制文件中写数据，应将每个字节的内容当作一个字符去写。对于单字符，可以用 put()写；对于转义字符或数值数据，则应把它们当作字节数组用 write()去写。

读二进制文件是写二进制文件的逆过程。单字节读，可使用 get()函数；多字节读，应使用 read()函数。

7. 文件的随机读/写

在默认的情况下，文件以写方式打开时，文件指针总是指向文件尾；以读方式打开时，文件指针总是指向文件首，并且总是每读几个字符，文件指针后移几个字符位置。这样的读/写操作是很被动的，程序员只能对文件进行顺序地访问。为了增加对文件访问的灵活性，C++ 在 istream 类及 ostream 类中定义了与在输入输出流中随机移动文件指针相关的 3 对成员函数，使程序员能够非常便利地对文件进行读/写操作。这 3 对函数的功能是：

① 文件指针的随机偏移；

② 测试文件指针当前位置；

③ 文件指针的绝对位移。

下面分别介绍这 3 对成员函数：

(1) 文件指针的随机偏移

为移动一个文件的指针，要指明以下 3 点：

① 流对象名。

② 偏移量：文件指针偏移是文件指针相对于当前位置向前或向后移动一个偏移量——字节数。C++ 中，偏移量被定义为 streamoff 类型。定义语句为：

```
typedef long streamoff;
```

即偏移量为长整型。

③ 偏移参照位置：C++ 给出 3 种参照位置：文件首(beg)、文件尾(end)、当前指针位置(cur)，并在 ios 类中把它们定义为一个枚举类型 seek_dir：

```
enum seek_dir {beg=0,cur=1,end=2};
```

C++ 文件指针的移动视输入流与输出流分别用函数 seekg()与 seekp()来执行。这一对函数分别定义在 istream 类与 ostream 类中，它们的原型为：

```
istream & istream::seekg (streamoff,seek_dir);
ostream & ostream::seekp (streamoff,seek_dir);
```

(2) 测试文件指针当前位置

在与输入流输出流中相连的文件中测试文件指针当前位置，分别采用 tellg()/tellp()函数。它们的原型分别为：

```
stream istream::tellg();
stream ostream::tellp();
```

(3) 文件指针绝对位移

文件指针绝对位移是给出文件指针的绝对地址。在 C++ 中，把文件指针的绝对地址定义为 streampos 型。

相对于输入流输出流的一对使文件指针按绝对地址位移的成员函数的原型为：

```
istream & istream::seekg (streampos);
ostream & ostream::seekp (streampos);
```

在程序中使用的文件指针的绝对值，常与 tellg()或 tellp()的返回值有关。

下面给出一个含有上述 3 种函数的演示程序。

当对文件进行随机访问时，文件的打开方式最好指定为二进制方式，以防止将 0X0A 的字符转换成 0X0D 和 0X0A 两个字符，影响文件指针的正确定位。

8. 设备文件的使用

(1) 操作系统下的重定向

当用户根据需要改变标准流时，要利用操作系统提供的重定向技术。标准错误输出不可以重定向，它一定要输出到屏幕上。

(2) 程序中的使用

由于设备也是一种文件，在 C++ 程序中，可以通过创建与 I/O 设备相联系的流，来使用设备。

在程序中也可以通过流间的赋值进行重定向。

6.2 例题示范

例 6.2.1 试使用流的格式化状态字与 I/O 操作算子，设计输出左对齐或右对齐杨辉三角形。

解：

关于输入输出格式，系统默认是左对齐，如果需要右对齐则需采用格式化状态字或 I/O 操作算子进行设置。

```
#include "iostream.h"
#include <iomanip.h>          //因为用到了带参的 I/O 操作算子
#define MAXLINE 50

class YangHui{
  int *y[MAXLINE];
  int n;
public:
  YangHui(int);
  ~YangHui();
  void lalign_out();
  void ralign_out();

};
YangHui::YangHui(int n)
{
  YangHui::n=n;
  for(int i=0;i<n;i++)
  {
      y[i]=new int[sizeof(int)*(i+1)];
      for(int j=0;j<=i;j++)
      {
         if(j==0||j==i)
              *(y[i]+j)=1;
         else
              *(y[i]+j)=*(y[i-1]+j-1)+*(y[i-1]+j);
     }
  }
}
void YangHui::lalign_out()
{
cout.setf(ios::left);          //使用流的格式化状态字确定数据左对齐
  for(int i=0;i<n;i++)
  {
     for(int j=0;j<=i;j++)
     {
         cout.width(4);
         cout<<*(y[i]+j);
     }
     cout<<endl;
```

```
        }
    }
    void YangHui::ralign_out()
    {
    cout<<setiosflags(ios::right);//使用流的 I/O 操作算子确定数据右对齐
    for(int i=0;i<n;i++)
      {
        for(int j=0;j<=i;j++)
        {
            if(j==0)cout.width((n-i)*5-j*n);
            else cout.width(5);
            cout<<*(y[i]+j);
        }
        cout<<endl;
      }
    }

    YangHui::~YangHui()
    {
      for(int i=0;i<n;i++)
        delete y[i];
    }

    int main()
    {
      YangHui trianhle(10);
      char lrt;
      cout<<"输出左对齐还是右对齐杨辉三角形(L/R)?:";
      cin>>lrt;
      if(lrt=='l')
          trianhle.lalign_out();
      else if(lrt=='r')
          trianhle.ralign_out();
      return 0;
    }
```

测试结果 1:

```
输出左对齐还是右对齐杨辉三角形(L/R)?:l
1
1   1
```

```
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10  10  5  1
1  6  15  20  15  6  1
1  7  21  35  35  21  7  1
1  8  28  56  70  56  28  8  1
1  9  36  84  126  126  84  36  9  1
```

测试结果 2：

```
输出左对齐还是右对齐杨辉三角形(L/R)?:r
                                    1
                                  1  1
                                1  2  1
                              1  3  3  1
                            1  4  6  4  1
                          1  5  10  10  5  1
                        1  6  15  20  15  6  1
                      1  7  21  35  35  21  7  1
                    1  8  28  56  70  56  28  8  1
                  1  9  36  84  126  126  84  36  9  1
```

例 6.2.2 编写一文本文件的读写程序，拷贝一个文本文件的数字字符到另一个文本文件中。

解：

文本文件又称 ASCII 文件，它的每一个字节存放一个 ASCII 代码，代表一个字符，这样便于对字符进行处理，也便于输出。

在 C++ 中，要进行文件的输入输出，需做 3 件事。

(1) 在程序中要包含头文件 fstream.h，它包含了输入文件流 ifstream、输出文件流 ofstream、输入输出文件流 fstream 类的定义。

(2) 声明流对象。

(3) 使用 open()函数打开文件，即建立文件同流的联系。

```
#include "iostream.h"
#include "fstream.h"
class CFileCopy{
  ifstream inf; // 声明一个输入流对象，用来操作拷贝源文件
```

```
    fstream ouf;                //声明一个输出流对象,用来操作拷贝目的文件
    char file1[20],file2[20];

  public:
    CFileCopy();                //打开源文件,建立目的文件
    ~CFileCopy();               //关闭目的文件
    void copyfile();            //读源文件,写目的文件
    void pifile();              //输出源文件
    void pofile();              //输出目的文件
  };

  CFileCopy::CFileCopy()
  {
    cout<<"源文件路径名:";
    cin>>file1;
    cout<<"目的文件路径名:";
    cin>>file2;
    inf.open(file1);
    if(! inf)
    {
      cout<<"can not open source file:"<<file1<<endl;
      return;
    }
    ouf.open(file2,ios::out|ios::in,0);
    if(! ouf)
    {
      cout<<"can not create target file:"<<file1<<endl;
      return;
    }

  }

  CFileCopy::~CFileCopy()
  {
    ouf.close();
  }

  void CFileCopy::copyfile()
  {
    char ch;
    inf.get(ch);
    while(! inf.eof())
    {
```

```
    if(ch>='0'&&ch<='9')
        ouf.put(ch);
    inf.get(ch);
  }
  inf.close();
}

void CFileCopy::pifile()
{
  char ch;
  inf.open(file1);
  inf.get(ch);
  while(! inf.eof())
  {
    cout<<ch;
    inf.get(ch);

  }
  cout<<endl;
}

void CFileCopy::pofile()
{
  char ch;
  ouf.seekg(0);      //使文件指针定位在文件的首位
  ouf.get(ch);
  while(! ouf.eof())
  {
    cout<<ch;
    ouf.get(ch);

  }
  cout<<endl;
}
void main()
{
  CFileCopy fcopy;
  fcopy.copyfile();
  cout<<"源文件:"<<endl;
  fcopy.pifile();
  cout<<"目的文件:"<<endl;
  fcopy.pofile();
}
```

测试结果：

```
源文件路径名:test1
目的文件路径名:test2
源文件:
a1,b2,c3;
e4,f5,e6;
x7,y8,z9;
目的文件:
123456789
```

例 6.2.3 编写一个 C++ 程序，实现两个文件的连接。

解：这里所指的两个文件的连接，就是把两个已存在的文件内容顺次拷贝到一个新的文件中。在本程序设计中我们采用输入输出流类 fstream 声明了 3 个文件流对象，用以分别操作上述 3 个文件。这时其输入输出方式应自行设置，而不能采用系统默认方式。

```
#include "fstream.h"

class CFileLink{
  fstream file1,file2,file3; // 声明 3 个 I/O 流对象
  char fn1[20],fn2[20],fn3[20];

public:
  CFileLink();          //打开对接的两个文件,建立连接后的结果文件
  ~CFileLink();         //关闭文件
  void linkfile();      //读两个源文件,写结果文件
  void prtfile();       //输出结果文件
};

CFileLink::CFileLink()
{
  cout<<"请输入两个连接的文件名和结果文件名"<<endl;
  cout<<"连接文件 1 路径名:";
  cin>>fn1;
  cout<<"连接文件 2 路径名:";
  cin>>fn2;
  cout<<"结果文件路径名:";
  cin>>fn3;

  file1.open(fn1,ios::in,0);
  if(! file1)
  {
```

```
        cout<<"can not open file:"<<fn1<<endl;
        return;
    }
    file2.open(fn2,ios::in,0);
    if(! file2)
    {
        cout<<"can not open file:"<<fn2<<endl;
        return;
    }
    file3.open(fn3,ios::out|ios::in,0);
    if(! file3)
    {
        cout<<"can not open file:"<<fn3<<endl;
        return;
    }

}
CFileLink::~CFileLink()
{
    file1.close();
    file2.close();
    file3.close();
}
void CFileLink::linkfile()
{
    char ch;
    file1.get(ch);
    while(! file1.eof())
    {
        file3.put(ch);
        file1.get(ch);
    }
    file2.get(ch);
    while(! file2.eof())
    {
        file3.put(ch);
        file2.get(ch);
    }
}

void CFileLink::prtfile()
{
    char ch;
```

```
    file3.seekp(0);
    file3.get(ch);
    while(! file3.eof())
    {
        cout<<ch;
        file3.get(ch);

    }
    cout<<endl;
}
void main()
{
    CFileLink flink;
    flink.linkfile();
    cout<<"结果文件内容:"<<endl;
    flink.prtfile();
}
```

设 test1 文件内容是：

ABCDEFGHIJKLMNOPQRSTUVWXYZ;

test2 文件内容是：

0 1 2 3 4 5 6 7 8 9;

则程序的测试结果如下：

```
请输入两个连接的文件名和结果文件名
连接文件 1 路径名:test1
连接文件 2 路径名:test2
结果文件路径名:test3
结果文件内容:
ABCDEFGHIJKLMNOPQRSTUVWXYZ;
0 1 2 3 4 5 6 7 8 9;
```

例 6.2.4 编写一个二进制文件的读写程序，建立一个二进制文件存放从键盘输入的 10 个实数，再从该文件中读取这些实数并输出其中最小和最大的数。

解：二进制文件是把内存中的数据，按其在内存中的存储形式原样写在磁盘上，用二进制形式向磁盘上写数据，可节省空间和转换时间。例如，整常数 2468，二进制形式仅占 2 个字节的存储空间，而文本方式则要占用 4 个字节的存储空间，且需要转换，但本质上仍然是字节流，所以在读写文件时仍然以字节为单位进行读写，但应清楚这时一个字节并不对应一个字符。

```
#include "iostream.h"
#include "fstream.h"
class ioBinary{
  fstream iof;
public:
  ioBinary();
~ioBinary();
  void wfile();
  void rfile();
};
ioBinary::ioBinary()                                    //构造函数建立一个二进制文件
{
  iof.open("test",ios::binary|ios::in|ios::out,0);
  if(!iof)
  {
      cout<<"can not open file!"<<endl;
return;
  }
}
ioBinary::~ioBinary()                                   //析构函数关闭二进制文件
{
  iof.close();
}

void ioBinary::wfile()
{
  float fnum;
  cout<<"Input 10 number:"<<endl;
  for(int i=1;i<=10;i++)
  {
      cin>>fnum;
      iof.write((char*)&fnum,sizeof(fnum));             //将变量 fnum 的内容写入文件
  }
}
void ioBinary::rfile()
{
  float fmin,fmax,ftemp;
  iof.seekg(0);                                         //使文件指针定位在文件的首位
  iof.read((char*)&ftemp,sizeof(ftemp));                //从文件中读取第一个实数
  fmin=fmax=ftemp;
  for(int i=1;i<10;i++)
  {                                                     //从文件中读取后 9 个实数
      iof.read((char*)&ftemp,sizeof(ftemp));
      if(ftemp<fmin)
```

```
        fmin=ftemp;
    if(ftemp>fmax)
        fmax=ftemp;
  }
  cout<<"The minimum number is:";
  cout<<fmin<<endl;
  cout<<"The maximum number is:";
  cout<<fmax<<endl;
}

void main()
{
  ioBinary iobin;
  iobin.wfile();
  iobin.rfile();
}
```

测试结果：

```
Input 10 number:
5.5  3.2  23.6  78.0  99.8  2.3  1.5  77.2  33.4  4.78
The minimum number is: 1.5
The maximum number is: 99.8
```

6.3 自我测试

6.3.1 自测习题

题 6.3.1 试分别使用流的格式化状态字与 I/O 操作算子，各写一个三角形的九九乘法表。

题 6.3.2 使用成员函数 width 和 precision，请在空格处填入适当代码，使程序输出以下所示结果。

```
#include <iostream.h>
void main()
{
  cout<<"…+…+…+…\n";
  cout.width(10);cout<<"abcd";
  cout.width(5);cout<<10<<endl;
  cout.flags(ios::fixed);
  [        ①        ];
  [        ②        ];
  cout<<123.456789<<endl;
}
```

输出结果：

```
…+…+…+…
      abcd   10
123.4568
```

题 6.3.3 以下程序用于反复输入姓名和年龄，直到输入 Ctrl+Z 为止。请在空格处填入适当代码。

```
//eof 函数
#include <iostream.h>
void main()
{
  char name[20];
  int age;
  while(cout<<"姓名 年龄 ?",[      ①      ],[      ②      ])
      cout<<name<<" : "<<age<<endl;
}
```

输出结果：

```
姓名  年龄 ? candy 21
candy  : 21
姓名  年龄 ? eluza 19
eluza : 19
姓名  年龄 ? ^Z
```

题 6.3.4 编写一个 C++ 程序，实现一个文件的拷贝。

题 6.3.5 编写一个 C++ 程序，实现两个文件的对接，所谓对接就是把一个文件的内容追加到另一个文件的后面而不产生第三个文件。

题 6.3.6 写一个能对一文件的各行加行号(后空一格)的 C++ 程序。

题 6.3.7 写一个程序，从开始处比较两个文件，给出开始不同的位置。

题 6.3.8 编写一个二进制文件的读写程序，建立一个二进制文件存放带有整型序号的 10 个实数，再从该文件中读取这些实数，并输出其中最小和最大数的序号。

题 6.3.9 编写一个程序，建立存放学生记录的二进制文件 student.bin，其中每条记录包含姓名、性别、年龄和平均成绩等信息。

题 6.3.10 编写一个程序，对题 6.9(教材)中建立的二进制文件 student.bin 追加记录。

题 6.3.11 编写一个程序，输出二进制文件 student.bin 所有记录。

题 6.3.12 编写一个程序，对二进制文件 student.bin 按学生姓名进行查询。

6.3.2 参考答案

题 6.3.1 参考答案：

解 1：

```
/*****使用流的格式化状态字写一个输出三角形的九九乘法表的类*****/
#include "iostream.h"
class CMulTable{
public:
  pmultable()
  {
      cout.setf(ios::right);  //使用流的格式化状态字确定数据右对齐
      for(int i=1;i<10;i++)
      {
         for(int j=1;j<=i;j++)
         {
             cout<<j<<"×"<<i<<"=";
             cout.width(2);  //乘积输出占 2 位
             cout<<i*j<<" ";
         }
         cout<<endl;
      }
  }
};
```

解 2：

```
/*****使用流的 I/O 操作算子写一个输出三角形的九九乘法表的类*****/
#include "iostream.h"
class CMulTable{
public:
  pmultable()
  {
      cout<<setiosflags(ios::right);     //使用流的 I/O 操作算子确定数据右对齐
      for(int i=1;i<10;i++)
      {
         for(int j=1;j<=i;j++)
         {
             cout<<j<<"×"<<i<<"=";
             cout.width(2);             //乘积输出占 2 位
             cout<<i*j<<" ";
         }
         cout<<endl;
      }
  }
};
```

测试主函数：

```
void main()
{
  CMulTable mtable;
  mtable.pmultable();
}
```

测试结果：

```
1×1= 1
1×2= 2  2×2= 4
1×3= 3  2×3= 6  3×3= 9
1×4= 4  2×4= 8  3×4=12  4×4=16
1×5= 5  2×5=10  3×5=15  4×5=20  5×5=25
1×6= 6  2×6=12  3×6=18  4×6=24  5×6=30  6×6=36
1×7= 7  2×7=14  3×7=21  4×7=28  5×7=35  6×7=42  7×7=49
1×8= 8  2×8=16  3×8=24  4×8=32  5×8=40  6×8=48  7×8=56  8×8=64
1×9= 9  2×9=18  3×9=27  4×9=36  5×9=45  6×9=54  7×9=63  8×9=72  9×9=81
```

题 6.3.2 参考答案：

① cout.width(10)
② cout.precision(4)

题 6.3.3 参考答案：

① cin≫name≫age
② ! cin.eof()

题 6.3.4 参考答案：

```
#include "iostream.h"
#include "fstream.h"

class CFileCopy{
  fstream file1,file2;
  char fn1[20],fn2[20];

public:
  CFileCopy();        //打开一个拷贝源文件,建立拷贝目的文件
  ~CFileCopy();       //关闭文件
  void copyfile ();   //读源文件,写目的文件
  void prtfile();     //输出目的文件
};
```

```
CFileCopy::CFileCopy()
{
  cout<<"请输入源文件的文件名和目的文件名"<<endl;
  cout<<"源文件路径名:";
  cin>>fn1;
  cout<<"目的文件路径名:";
  cin>>fn2;
  file1.open(fn1,ios::in,0);
  if(file1.fail())
  {
      cout<<"can not open file:"<<fn1<<endl;
      return;
  }
  file2.open(fn2,ios::out|ios::in,0);
  if(file2.fail())
  {
      cout<<"can not open file:"<<fn2<<endl;
      return;
  }
}
CFileCopy::~CFileCopy()
{
  file1.close();
  file2.close();
}
void CFileCopy::copyfile()
{
  char ch;
  file1.get(ch);
  while(! file1.eof())
  {
      file2.put(ch);
      file1.get(ch);
  }
}
void main()
{
  CFileCopy fcopy;
  fcopy.copyfile();
  cout<<"结果文件内容:"<<endl;
  fcopy.prtfile();
}
```

设有 test1 文件内容是:

ABCDEFGHIJKLMNOPQRSTUVWXYZ;

测试结果：

```
请输入源文件的文件名和目的文件名
源文件路径名：test1
目的文件路径名：test2
结果文件内容：
ABCDEFGHIJKLMNOPQRSTUVWXYZ；
```

题 6.3.5 参考答案：

```
#include "iostream.h"
#include "fstream.h"

class CFileCat{
  fstream file1,file2;
  char fn1[20],fn2[20];

public:
  CFileCat();          //打开对接的两个文件
  ~CFileCat();         //关闭文件
  void catfile ();     //读源文件，写目的文件
  void prtfile();      //输出文件
};

CFileCat::CFileCat()
{
  cout<<"请输入两个对接的文件名"<<endl;
  cout<<"文件 1 路径名："；
  cin>>fn1；
  cout<<"文件 2 路径名："；
  cin>>fn2；
  file1.open(fn1,ios::out|ios::in,0)；
  if(file1.fail())
  {
      cout<<"can not open file："<<fn1<<endl；
      return；
  }
  file2.open(fn2,ios::in,0)；
  if(file2.fail())
  {
      cout<<"can not open file："<<fn2<<endl；
      return；
  }
}
CFileCat::~CFileCat()
```

```
{
  file1.close();
  file2.close();

}
void CFileCat::catfile()
{
  char ch;
  file1.seekg(0,ios::end);   //把前一个文件的文件指针指向文件的尾部
  file2.get(ch);
  while(! file2.eof())
  {
      file1.put(ch);
      file2.get(ch);
  }
}

void CFileCat::prtfile()
{
  char ch;
  file1.seekp(0);
  file1.get(ch);
  while(! file1.eof())
  {
      cout<<ch;
      file1.get(ch);

  }
  cout<<endl;
}
void main()
{
  CFileCat fcat;
  fcat.catfile();
  cout<<"结果文件:"<<endl;
  fcat.prtfile();
}
```

设有 test1 文件内容是:

AAAAAAAAAA;
BBBBBBBBBB;
CCCCCCCCCC;
DDDDDDDDDD;
EEEEEEEEEE;

test2 文件内容是：

```
1111111111;
2222222222;
3333333333;
4444444444;
5555555555;
```

则程序的测试结果如下：

```
请输入两个对接的文件名
文件 1 路径名：test1
文件 2 路径名：test2
结果文件：
AAAAAAAAAA;
BBBBBBBBBB;
CCCCCCCCCC;
DDDDDDDDDD;
EEEEEEEEEE;
1111111111;
2222222222;
3333333333;
4444444444;
5555555555;
```

题 6.3.6 参考答案：

```
#include <iostream.h>
#include <fstream.h>
#include <strstrea.h>
#include <stdlib.h>
#include <iomanip.h>
class CFileLineNo{
  fstream file;
  char fn[20];

public:
  CFileLineNo();        //打开源文件
  ~CFileLineNo();       //关闭文件
  void writeln();       //读源文件,写行号
  void prtfile();       //输出文件
};

CFileLineNo::CFileLineNo()
{
  cout<<"源文件路径名:";
```

```
    cin≫fn;
    file.open(fn,ios::out|ios::in,0);
    if(file.fail())
    {
        cout≪"can not open file:"≪fn≪endl;
        return;
    }
}
CFileLineNo::~CFileLineNo()
{
    file.close();
}
void CFileLineNo::writeln()
{
    strstream filebuf;
    filebuf≪file.rdbuf();
    file.seekp(0);
    const int bsz=100;
    char buf[bsz];
    int line=0;
    while(filebuf.getline(buf,bsz))
    {
        file.setf(ios::right,ios::adjustfield);
        file≪setw(2)≪++line≪" "≪buf≪endl;
    }
}
void CFileLineNo::prtfile()
{
    char ch;
    file.seekp(0);
    file.get(ch);
    while(!file.eof())
    {
        cout≪ch;
        file.get(ch);
    }
    cout≪endl;
}
void main()
{
    CFileLineNo fln;
    fln.writeln();
    cout≪"结果文件内容:"≪endl;
    fln.prtfile();
}
```

设运行前文件 test 的内容为：

```
aaaaaaaaaaaaaaaaaaa;
bbbbbbbbbbbbbbbbbbbb;
cccccccccccccccccccc;
dddddddddddddddddddd;
eeeeeeeeeeeeeeeeeeee;
ffffffffffffffffffff;
gggggggggggggggggggg;
```

测试结果：

```
源文件路径名:test
结果文件内容:
1  aaaaaaaaaaaaaaaaaaa;
2  bbbbbbbbbbbbbbbbbbbb;
3  cccccccccccccccccccc;
4  dddddddddddddddddddd;
5  eeeeeeeeeeeeeeeeeeee;
6  ffffffffffffffffffff;
7  gggggggggggggggggggg;
8
```

说明：

在程序中用到字符串流对象 filebuf，是考虑到将结果写回原文件时，文件中的原有内容会被删除，为此将文件中内容先读出来临时存放到 textfile 对象中后，再写回去。

题 6.3.7 参考答案：

```
//compare two files
#include <iostream.h>
#include <fstream.h>
#include <process.h>
class CFileComp{
  fstream file1,file2;
  char fn1[20],fn2[20];

public:
  CFileComp();        //打开两个比较的源文件
  ~CFileComp();       //关闭文件
  void compfile ();   //比较文件
};

CFileComp::CFileComp()
{
  cout<<"文件 1 路径名:";
  cin>>fn1;
  cout<<"文件 2 路径名:";
```

```
    cin>>fn2;
    file1.open(fn1,ios::in,0);
    if(file1.fail())
    {
        cout<<"can not open file:"<<fn1<<endl;
        return;
    }
    file2.open(fn2,ios::out|ios::in,0);
    if(file2.fail())
    {
        cout<<"can not open file:"<<fn2<<endl;
        return;
    }
}
CFileComp::~CFileComp()
{
    file1.close();
    file2.close();
}
void CFileComp::compfile()
{
    const int bsz=100;
    char buf1[bsz],buf2[bsz];
    int line=0;
    int col;
    while(file1.getline(buf1,bsz) && file2.getline(buf2,bsz))
    {
        line++;col=0;
        while(buf1[col]==buf2[col])
        {
            if(buf1[col]=='\0')break;
            col++;
        }
        if(buf1[col]!=buf2[col])
        {
            cout<<"两文件开始不同处位于: line "<<line<<", col "<<col+1<<endl;
            break;
        }
    }
}

void main()
{
    CFileComp fcomp;
    fcomp.compfile();
}
```

设文件 test1 的内容为：

ABCDEFG
ABCDUVW
1234567

文件 test2 的内容为：

ABCDEFG
ABCDXYZ
1234567

测试结果：

文件 1 路径名：TEST1
文件 2 路径名：TEST2
两文件开始不同处位于：line 2 ，col 5

题 6.3.8 参考答案：

```
#include <iostream.h>
#include <fstream.h>
class ioBinary{
  fstream iof;
public:
  ioBinary();
  void wfile();
  void rfile();
};
ioBinary::ioBinary()
{
  iof.open("test",ios::binary|ios::in|ios::out,0);
  if(iof.fail())
  {
      cout<<"can not open file!"<<endl;
      return;
  }
}
void ioBinary::wfile()
{
  float fnum[10]={6.6,3.3,7.7,8.8,9.9,2.2,5.5,1.1,4.4,1.2};
  for(int i=1;i<=10;i++)
  {
      iof.write((char *)&i,sizeof(i));
      iof.write((char *)&fnum[i-1],sizeof(fnum[i-1]));
  }
```

```
}
void ioBinary::rfile()
{
  float fmin,fmax,ftemp;
  int minno,maxno,tempno;
  iof.seekg(0);
  iof.read((char*)&tempno,sizeof(tempno));
  iof.read((char*)&ftemp,sizeof(ftemp));
  minno=maxno=tempno;
  fmin=fmax=ftemp;
  for(int i=1;i<10;i++)
  {
      iof.read((char*)&tempno,sizeof(tempno));
      iof.read((char*)&ftemp,sizeof(ftemp));
      if(ftemp<fmin)
      {
         minno=tempno;
         fmin=ftemp;
      }
      if(ftemp>fmax)
      {
         maxno=tempno;
         fmax=ftemp;
      }
  }
  cout<<"The minimum no is:";
  cout<<minno<<endl;
  cout<<"The maximum no is:";
  cout<<maxno<<endl;
}

void main()
{
  ioBinary iobin;
  iobin.wfile();
  iobin.rfile();
}
```

测试结果：

```
The minimum no is:8
The maximum no is:5
```

题 6.3.9 参考答案：

```
#include <iostream.h>
#include <fstream.h>
class CStudent{
  struct SRecord{      // 声明记录结构
      char name[20];
      char sex;
      int age;
      float avescore;
  }srec;
  fstream file;
public:
  CStudent();
  ~CStudent();
  void createfile();
};
CStudent::CStudent() //构造函数建立二进制文件
{
  file.open("student.bin",ios::out|ios::binary,0);
  if(file.fail())
  {
      cout<<"can not create file:"<<endl;
      return;
  }
}
CStudent::~CStudent() //析构函数关闭文件
{
  file.close();
}
void CStudent::createfile()
{
  char yn;
  cout<<"是否输入下一记录(Y/N)?";
  cin>>yn;
  while(yn=='Y'||yn=='y')
  {
      cout<<"姓名：(用分号作为输入的结束符标志)";
      for(int i=0;i<20;i++)
      {
         cin>>srec.name[i];
         if(srec.name[i]==';')
              break;
      }
```

```
        srec.name[i]='\0';
        cout<<"性别(m/f):";
        cin>>srec.sex;
        cout<<"年龄:";
        cin>>srec.age;
        cout<<"平均成绩:";
        cin>>srec.avescore;
        file.write((char *)&srec,sizeof(srec));//向文件中写一个记录
        cout<<"是否输入下一纪录(Y/N)?";
        cin>>yn;
    }
}
void main()
{
  CStudent student;
  student.createfile();
}
```

测试结果:

```
是否输入下一记录(Y/N)? Y
姓名:(用分号作为输入的结束符标志)zhaoxiaogang;
性别(m/f): m
年龄: 18
平均成绩: 68.5
是否输入下一记录(Y/N)? Y
姓名:(用分号作为输入的结束符标志)wangming;
性别(m/f): m
年龄: 20
平均成绩: 73.8
是否输入下一记录(Y/N)? Y
姓名:(用分号作为输入的结束符标志)zhangqiang;
性别(m/f): m
年龄: 19
平均成绩: 85.
是否输入下一记录(Y/N)? Y
姓名:(用分号作为输入的结束符标志)sunlin;
性别(m/f): m
年龄: 19
平均成绩: 79.
是否输入下一记录(Y/N)? Y
姓名:(用分号作为输入的结束符标志)shifeng;
性别(m/f): f
年龄: 18
```

平均成绩：90.
是否输入下一记录(Y/N)? Y
姓名：(用分号作为输入的结束符标志)hexianxia;
性别(m/f)：f
年龄：19
平均成绩：89.
是否输入下一记录(Y/N)? N

题 6.3.10 参考答案：

```
#include <fstream.h>>
class CStudent;
  struct SRecord{                    //声明记录结构
      char name[20];
      char sex;
      int age;
      float avescore;
  }srec;
  fstream file;

public:
  CStudent();
  ~CStudent();
  void appendrecord();
};
CStudent::CStudent()            //构造函数以读写方式打开 student.bin 文件
{
  file.open("student.bin",ios::in|ios::out
            |ios::binary|ios::nocreate,0);
  if(file.fail())
  {
      cout<<"can not open file."<<endl;
      return;
  }
  file.seekg(0,ios::end);       //将文件指针指向文件末尾
}
CStudent::~CStudent()
{
  file.close();
}
void CStudent::appendrecord()
{
  char yn;
  cout<<"是否追加下一记录(Y/N)?";
```

```
    cin≫yn;
    while(yn==='Y'||yn==='y')
    {
        cout≪"姓名：(用分号作为输入的结束符标志)";
        for(int i=0;i<20;i++)
        {
            cin≫srec.name[i];
            if(srec.name[i]==';')
                break;
        }
        srec.name[i]='\0';
        cout≪"性别(m/f)：";
        cin≫srec.sex;
        cout≪"年龄：";
        cin≫srec.age;
        cout≪"平均成绩：";
        cin≫srec.avescore;
        file.write((char*)&srec,sizeof(srec));  //向文件中追加一个记录
        cout≪"是否输入下一记录(Y/N)?";
        cin≫yn;
    }
}
void main()
{
    CStudent student;
    student.appendrecord();
}
```

测试结果：

```
是否追加下一记录(Y/N)? Y
姓名：(用分号作为输入的结束符标志)lili;
性别(m/f)：f
年龄：18
平均成绩：77.8
是否追加下一记录(Y/N)? N
```

题 6.3.11 参考答案：

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>
class CStudent{
    struct SRecord{
        char name[20];
```

```
        char sex;
        int age;
        float avescore;
    }srec;
    fstream file;

  public:
    CStudent();
    ~CStudent();
    void oallrecord();
  };
  CStudent::CStudent()
  {
    file.open("student.bin",ios::in|ios::binary|ios::nocreate,0);
    if(file.fail())
    {
        cout<<"can not open file:"<<endl;
        return;
    }
  }
  CStudent::~CStudent()
  {
    file.close();
  }
  void CStudent::oallrecord()
  {
    file.read((char *)&srec,sizeof(srec));

    while(! file.eof())
    {

        cout<<"姓名:"<<srec.name<<", ";
        cout<<"性别:"<<srec.sex<<", ";
        cout<<"年龄:"<<srec.age<<", ";
        cout<<"平均成绩:"<<srec.avescore<<endl;
        file.read((char *)&srec,sizeof(srec));
    }
  }
  void main()
  {
    CStudent student;
    student.oallrecord();
  }
```

测试结果：

姓名：zhaoxiaogang，性别：m，年龄：18，平均成绩：68.5
姓名：wangming，性别：m，年龄：20，平均成绩：73.8
姓名：zhangqiang，性别：m，年龄：19，平均成绩：85
姓名：sunlin，性别：m，年龄：19，平均成绩：79
姓名：shifeng，性别：f，年龄：18，平均成绩：90
姓名：hexianxia，性别：f，年龄：19，平均成绩：69
姓名：lili，性别：f，年龄：18，平均成绩：77.8

题 6.3.12 参考答案：

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>
class CStudent{
  struct SRecord{
      char name[20];
      char sex;
      int age;
      float avescore;
  }srec;
  fstream file;

public:
  CStudent();
  ~CStudent();
  void query();
};
CStudent::CStudent()
{
  file.open("student.bin",ios::in|ios::binary|ios::nocreate,0);
  if(file.fail())
  {
      cout<<"can not open file:"<<endl;
      return;
  }
}
CStudent::~CStudent()
{
  file.close();
}
void CStudent::query()
{
  char name[20];
```

```
    bool flag=false;
    cout<<"查询的学生姓名(用分号作为输入的结束符标志):";
    for(int i=0;i<20;i++)
    {
        cin>>name[i];
        if(name[i]==';') break;
    }
    name[i]='\0';
    file.read((char*)&srec,sizeof(srec));

    while(!file.eof())
    {

        if(strcmp(srec.name,name)==0)
        {
            flag=true;
            break;
        }
        file.read((char*)&srec,sizeof(srec));
    }
    if(flag)
    {
        cout<<"姓名:"<<srec.name<<endl;
        cout<<"性别:"<<srec.sex<<endl;
        cout<<"年龄:"<<srec.age<<endl;
        cout<<"平均成绩:"<<srec.avescore<<endl;
    }
    else
        cout<<"查无此人!"<<endl;
}
void main()
{
    CStudent student;
    student.query();
}
```

测试结果:

```
查询的学生姓名(用分号作为输入的结束符标志):shifeng;
姓名:shifeng
性别:f
年龄:18
平均成绩:90
```

第7章 Visual C++程序设计

7.1 编程知识

7.1.1 Windows应用程序设计

1. Windows应用程序设计的特点

(1) 面向对象；

(2) 窗口为核心的用户界面；

(3) 以事件驱动为动力的程序运行机制；

(4) 程序代码与用户界面分别处理。

2. 有关概念

(1) 窗口

窗口是由应用程序创建的、系统显示器上的一个用于接收用户输入和显示输出的矩形区域。进一步说，窗口涉及如下一些概念：

① 应用程序只能通过窗口访问系统显示器，并且与其他应用程序共享显示器。

② 同一时间只有一个窗口是活动的。

③ Windows允许应用程序创建多个窗口，但每个应用程序至少要创建一个窗口——主窗口作为用户与应用程序的主要接口；其他的窗口用于完成相关任务，如对话框、消息框等。

④ 一旦创建了一个窗口，Windows就提供该窗口所对应的各种交互信息，并负责为应用程序管理屏幕，控制窗口的位置和显示，确保不会有两个应用程序的窗口冲突。

⑤ 一个应用程序的窗口一般由标题栏、菜单栏、工具栏、边框、客户区、滚动条等部件组成。

(2) 对象与句柄

表7.1是一些公共Windows对象及其句柄。

表7.1 公共Windows对象及其句柄

对象	句柄	对象	句柄
快捷键表	HACCEL	图标	HICON
位图	HBITMAP	菜单	HMENU
画刷	HBRUSH	调色板	HPALETTE
光标	HCURSOR	画笔	HPEN
设备描述表	HDC	区域	HRGN
文件	HFILE	窗口	HWND
字体	HFONT		

(3) 事件驱动

采用事件驱动的程序不给程序预先规定一个执行顺序，程序一旦启动，就处于等待状

态，等待某种事件的发生；只要有用户输入事件发生，它就立刻接收事件并做出响应；处理完毕又立刻返回等待下一个事件的发生。这种循环等待方式，具有面向用户性和被动性，可以给用户提供较大的干预空间，让用户以各种合理的和可能的顺序来安排程序的流程，特别适合交互性强的程序。

(4) 消息与消息循环

对程序来说，事件驱动是以消息传递的形式进行的。任何一个事件的发生，都有相应的消息进入程序，被程序接收。所谓循环等待事件的发生，也称为消息循环，有的也把事件驱动称为消息驱动。因此，消息机制是事件驱动程序设计的关键所在。图 7.1 为 Windows 消息机制的示意图。

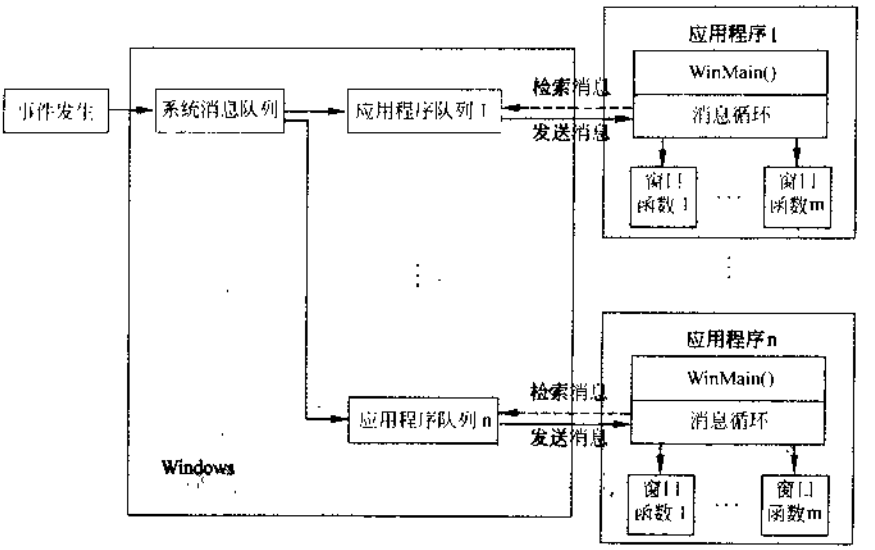

图 7.1 Windows 消息机制

图 7.2 为 Windows 应用程序的执行流程，它是图 7.1 的进一步补充。

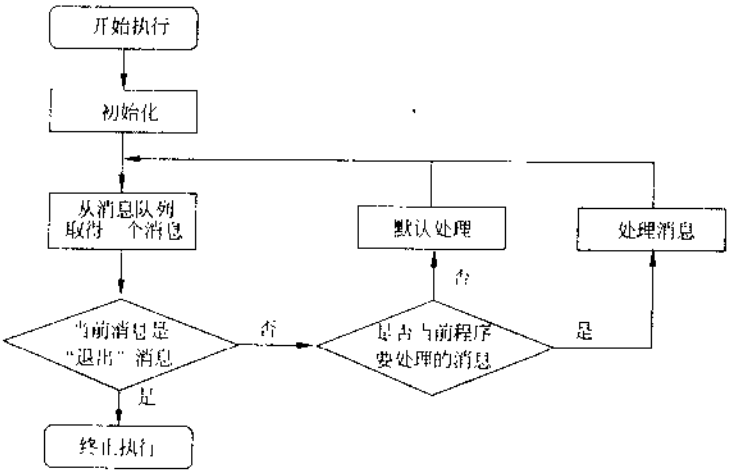

图 7.2 Windows 程序和 Windows 消息的基本流程

Windows 系统中有许许多多的消息，据统计大约有近千种消息，但从分布来说，可以分为系统消息和应用程序消息两种类型。其中系统消息可分为如下 3 类：

① 标准 Windows 消息：鼠标消息、键盘消息、窗口消息。

② 控件消息：对控件操作引起的消息。

③ 命令消息：菜单项、工具栏按钮、快捷键等用户界面发送的消息 WM_COMMAND。

每一个消息要携带许多数据，为此，在 Windows 中，把消息定义成如下的一个结构体：

```
typedef struct tagMSG{
    HWND hWnd;          // 目标窗口句柄
    UINT Message;       // 消息变量标识符，值为无符号整数(unsign int)
    WPARAM wParam;      // 伴随消息的附加信息(16 位)
    LPARAM lParam;      // 伴随消息的附加信息(32 位)
    DWORD time;         // 发送时间
    POINT pt;           // 发送时的鼠标指针位置
}MSG;
```

3. Windows 应用程序的结构框架

Windows 应用程序的基本结构具有固定性：入口点函数 WinMain 和窗口函数构成了 Windows 应用程序的基本框架。

(1) WinMain 函数

WinMain 函数是 Windows 应用程序的入口点。当 Windows 运行程序时，首先调用该程序的 WinMain 函数。因此，每一个 Windows 应用程序都必须有一个 WinMain 函数。但是 Windows 是一个多任务操作系统，每一个 Windows 应用程序都有可能并行地执行，形成该应用程序的多个实例，并用一个实例句柄惟一地标识它。

WinMain 函数用于完成一些特别的功能，其中最主要的是创建该应用程序的主窗口，为此，多数程序需要执行如下 4 部分操作：

① 注册窗口类。

② 创建窗口　　创建窗口就是以窗口类为模板，生成一个具体的窗口对象。具体是：指定窗口名、窗口标题、窗口(左上角)坐标、大小、父窗口句柄、当前应用程序实例句柄等。

③ 显示窗口。

④ 建立消息循环：从应用程序队列检索消息。

(2) 窗口函数

窗口函数主要执行两方面的操作：

① 从 Windows 接收消息。

② 检查每一条消息，进行相应的处理。

4. Windows 应用程序的工作模式

见图 7.3，以从键盘键入一个字符 x 为例，说明 Windows 应用程序的工作模式。

下面介绍这个过程。

(1) 用户利用中断将键入字符 x 送入系统队列；

(2) 系统将字符 x 送入应用程序队列；

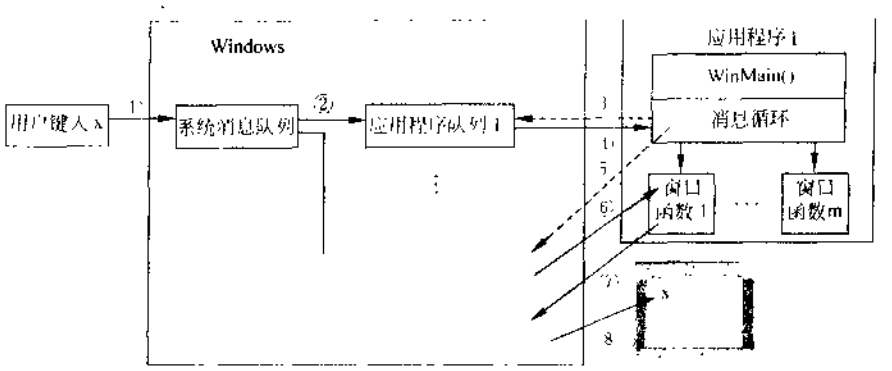

图 7.3　Windows 应用程序的工作模式

(3) 应用程序检索应用程序队列；

(4) 消息循环接收字符 x；

(5) Windows 从应用程序得到译成 ANSI 字符消息的 WM_CHAR；

(6) Windows 将 WM_CHAR 与原键盘消息发给窗口函数；

(7) 窗口函数发出请求；

(8) Windows 将字符 x 输出到对应的窗口。

5. Windows 应用程序的开发流程

图 7.4 表明 Windows 应用程序的开发流程。它也分为编辑、编译和链接 3 步，才能由源程序变为可执行的代码.EXE。

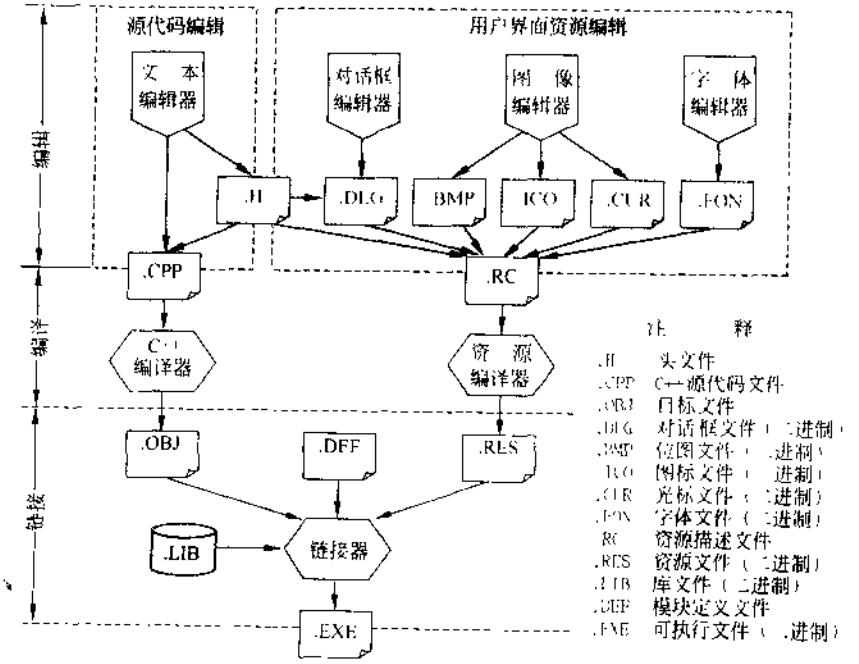

图 7.4　Windows 应用程序的开发流程

7.1.2 基于 MFC 的 Visual C++ 编程模式

1. MFC 类库层次结构

MFC(微软基本类库,Microsoft Foundation Class)不仅包含了 C++ 的基类,还包含了 WinMain 函数以及其他一些支持标准菜单、常见消息处理、窗口滚动等功能的元素和 API 函数,功能涉及整个 Windows 操作系统。

MFC 提供的类主要有:

(1) 应用程序框架结构类:应用程序和线程支持类、命令流程类、文档类、视图(体系结构)类、框架窗口类、文档模板类等。

(2) 窗口类:框架窗口类、视图(窗口)类、对话框类、控制类、控制条类等。

(3) 图形和打印类:输出类、绘图工具类等。

(4) 简单数据类型类:字符串类、位置类、时间类及其他类等。

(5) 集合类:模板集合类、数组类、列表类、映射类等。

(6) 文件和数据库类:文件 I/O 类、DAO 类、ODBC 类等。

(7) OLE 支持类:OLE 包容器类、OLE 服务器类、OLE 拖放和数据传输类、OLE 公用对话框类、OLE 自动化类、OLE 控制类、OLE 相关类、Active 类等。

(8) Internet 和网络类:ISAPI 类、Windows Socket 类、Win32 Internet 类等。

(9) 调试和异常处理类:调试支持类、异常处理类等。

(10) 其他各种辅助类。

除了各种辅助类外,MFC 中的类都是由 CObject 派生而来。在图 7.5 中,只画出了几个主要的类。

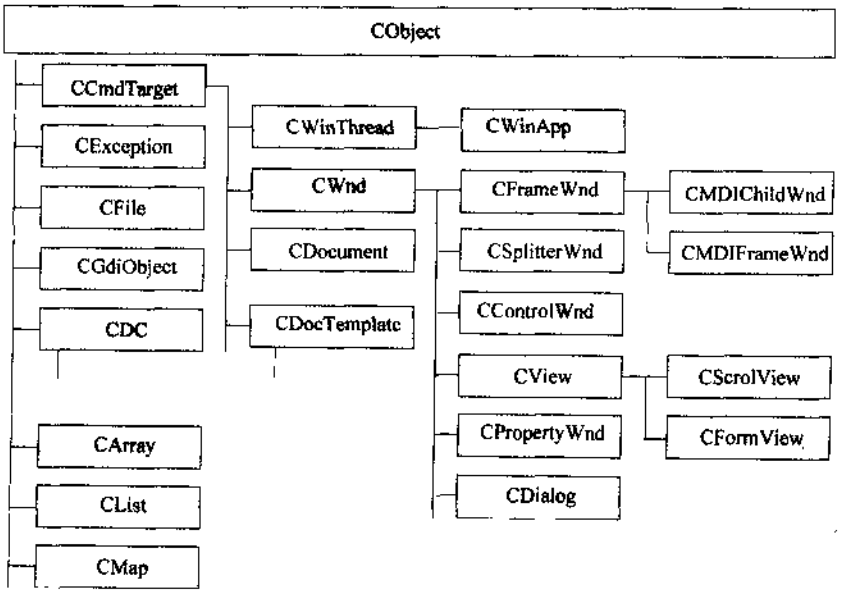

图 7.5 MFC 类库层次结构

2. 文档-视图结构

文档和视图是 MFC 的两类对象：文档(Document)对象用来维护、管理数据，包括数据的读取、存储与修改；视图(View)对象用来接收并显示数据，将这些数据交给文档类来处理。也就是说，视图类负责显示和编辑数据，文档类负责定义和处理数据。于是，“完整”的程序代码主要部分被分散到视图和文档两个类。MFC 将数据的文档与视图分离，会使程序的结构更清晰，更易于维护，给程序开发带来很大方便。

AppWizard 可以建立 3 种类型的应用程序结构：

(1) MDI(multiple document interface)：多文档界面，允许同时打开多个文档，多用于像 Word、Excel 那样的规模比较大的应用程序。

(2) SDI(single document interface)：单文档界面，一次只允许打开一个文档。

(3) Dialog based：基于对话的应用程序，不支持文档/视图结构，仅显示一个简单的对话框，主要用于编写小程序。

图 7.6 和图 7.7 分别是 MDI 和 SDI 类型应用程序的例子。

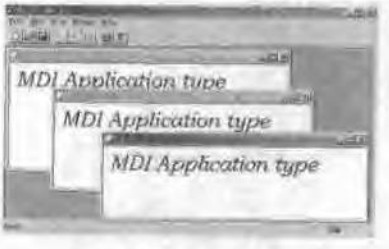

图 7.6 多文档界面的外观

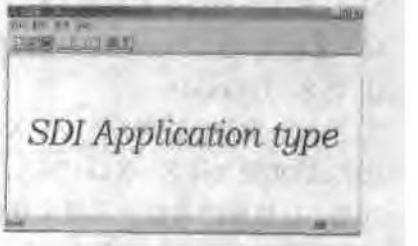

图 7.7 单文档界面应用程序的外观

3. Visual C++ 应用程序的逻辑构件——对象

Visual C++ 是一种用于进行 Windows 程序设计的可视化程序开发工具。基于用户界面的实现，不管是简单程序，还是复杂程序，都有 4 个主要的组成部分。

(1) 文档对象：用于存放应用程序的数据和文件。

(2) 视图对象：管理视图窗口，显示用户数据，接收用户对数据的鼠标、键盘操作，并传送给文档类对象。文档类对象必须通过视图类对象才能实现与用户的交互。

(3) 应用程序对象：控制应用程序的所有对象，包括文档、视图和边框窗口，并完成应用程序的初始化工作和程序退出的清除工作。

(4) 主窗口对象：主要用于管理主程序的窗口，显示标题栏、工具栏、状态栏，处理窗口操作消息。

使用 MFC AppWizard 生成 MFC 应用程序时，将派生一些类，主要的程序任务分布在这些类中实现。一般说来：

① 一个对话框应用程序由一个应用程序类和一个对话框类构成；

② 一个 SDI 应用程序由应用程序类、框架类、文档类和视图类构成；

③ 一个 MDI 应用程序由应用程序类、主框架类、子框架类、文档类和视图类构成。

与此同时，MFC AppWizard 为每个类生成各自的源文件，包括各类的定义头文件(.h)

和类的实现源文件(.cpp)

下面,以假定创建的名为 Xxxx 的 MDI 应用程序为例,分别介绍这几个类。

(1) 应用程序类(由 CWinApp 类派生)

默认的名字:

① 类名:CXxxxApp

② 类的头文件名:Xxxx.h

③ 类的实现文件名:Xxxx.cpp

功能:

① 控制应用程序的所有对象;

② 应用程序的初始化;

③ 创建主窗口;

④ 处理消息循环,从 Windows 操作系统中获取信息分发给程序窗口;

⑤ 退出应用程序;

⑥ 关闭应用

MFC 采用应用程序框架类 CWinApp 封装了这些共同代码。

(2) 文档类(由 CDocument 类派生)

① 类名:CXxxxDoc

② 类的头文件(定义该类)名:XxxxDoc.h

③ 类的实现文件名:XxxxDoc.cpp

功能:存放程序数据并实现文件保存和装载。

(3) 视图类(由 CView 类派生)

默认的名字:

① 类名:CXxxxView

② 类的头文件名:XxxxView.h

③ 类的实现文件名:XxxxView.cpp

功能:指定用户与文档间的交互形式。

(4) 主框架类(由 CMDIFrameWnd 类派生)

默认的名字:

① 类名:CMainFrame

② 类的头文件名:MainFrm.h

③ 类的实现文件名:MainFrm.cpp

功能:管理应用程序窗口,显示标题栏、菜单栏、工具栏、状态栏、控制菜单和控制按钮,包容子框架窗口。

(5) 子框架类(由 CMDIChildWnd 类派生)

默认的名字:

① 类名:CChildFrame

② 类的头文件名:ChildFrm.h

③ 类的实现文件名:ChildFrm.cpp

功能:管理应用程序窗口,无自己的菜单栏、工具栏、状态栏,但可与主框架类共享。

在上述5个类中，文档类、视图类、主框架类和应用程序类是最基本的。它们通过调用其他类的公共成员函数和发送消息进行通信，协同工作，交换数据，实现程序的功能。

除了主要类的源文件之外，MFC AppWizard还生成一些为建立应用程序所需的其他文件。

在MFC中用指针来标识对象。为了跨越MFC与Windows，MFC维护着一个查询表集合——句柄映像，用以在Windows对象与C++/MFC对象间的一一对应关系。

每个句柄都只对创建对象的系统组件有意义。MFC创建的句柄映像有两种类型：永久性的和临时性的。永久性的句柄映像应用于通过MFC创建的Windows API对象。对于不是通过MFC创建的Windows API对象，则使用临时性的句柄映像。

4. Visual C++应用程序的存储结构——文件

从物理上来看，Visual C++程序是由一些文件组成的。4个主要组成部分由不同的文件描述。一般说来，每一个主要的类都由两类文件支持：一类的后缀为.h，主要用于该类的定义（派生关系以及组成成员）等；一类的后缀为.cpp，主要用于描述该类的实现（写成员函数的代码）等。例如，主窗口类CmainFrame的两个支持文件是MainFrm.h和MainFrm.cpp。当然，组成程序的文件还要包括资源等。因而，AppWizard的成果之一是生成一系列文件：

（1）自述文件ReadMe.txt：该文件简要地介绍了AppWizard创建的其他新文件的内容和作用，并反映了对可选项所做的选择。

（2）项目文件：

PRJNAME.dsp：项目文件，用于管理项目结构，可用命令"Project | Setup…"修改配置。

PRJNAME.clw：ClassWizard文件，包含用ClassWizard编辑现有类或增加新类的信息。

（3）应用程序源文件和头文件：

PRJNAME.h：应用程序的应用程序类的头文件。

PRJNAME.cpp：应用程序的应用程序类的实现文件，包含应用类的具体实现代码和消息映射。

MainFrame.h：应用程序框架窗口类的头文件。

MainFrame.cpp：应用程序框架窗口类的实现文件。

ChildFrm.h：应用程序子框架窗口类的头文件。

ChildFrm.cpp：应用程序子框架窗口类的实现文件。

PRJNADoc.h：应用程序文档类的头文件。

PRJNADoc.cpp：应用程序文档类的实现文件，实现文档的初始化、保存、装入、调试诊断。

PRJNAView.h：应用程序视图类的头文件。

PRJNAView.cpp：应用程序视图的实现文件，实现文档数据的显示、打印。

（4）资源文件：

AppWizard在创建程序框架时能为其提供最基本的资源配置，并建立下面的资源

文件。

PRJNAME.rc：项目的资源文件。

Resource.h：资源头文件，包含了标准 MFC 的资源——标准的菜单、加速键和字符串表。

res\ PRJNAME.ico：应用程序的图标文件。

res\ PRJNDoc.ico：MDI 应用程序的子窗口图标文件。

res\TOOLBAR.bmp：位图文件，指定了支持工具条时，用于创建级联图像。

res\ PRJNAME.rc2：AppStudio 尚未建立和编辑的资源文件。

(5) 预定义的文件：STDAFX.h 和 STDAFX.cpp。

(6) 上下文敏感的帮助文件：

MakeHelp.bat：建立应用程序的帮助文件。

hlp\AfxCore.rtf：基于文档的应用程序的模板帮助文件。

hlp\AfxPrint.rtf：用于描述打印命令和打印对话框。

hlp\ PRJNAME.hpj：帮助项目文件，告诉编译器.rtf 文件以及帮助文件所需的位图、光标和图标。

*.bmp：生成帮助文件的位图文件。

(7) 按可选项增加的 AppWizard 文件。

对于不同的问题要求，要涉及更多的对象和文件。因而，程序的开发就成为一项复杂的工程项目了。程序开发的实际工作就是写组成项目的各个文件。不过，由于 Visual C++ 的开发工具能自动生成许多代码，大大减轻了程序开发的工作量。

5. 文档模板

MFC 用文档模板(Document Template)类的对象来存储程序文档、视图和框架窗口有关的信息，提供 MFC 用来将文档、视图和框架结合在一起的框架结构，以协调框架窗口、文档对象和视图对象之间的关系。简单地说，文档模板为应用程序创建新的文档和视图，只要 MFC 创建一个新的文档和视图，就会使用文档、视图和框架类型的类信息。如果要在应用程序中使用它们的不同组合，就必须为每一种组合创建独立的模板。

MFC 类库提供两种文档模板：

(1) CSingleDocTemplate：用于 SDI 应用程序，每次只能创建和管理一个文档。

(2) CMultiDocTemplate：用于 MDI 应用程序，每次可以创建和管理多个文档。

6. 应用程序的控制

一个 Visual C++ 应用程序的执行过程受事件驱动的控制。图 7.8 是其流程控制的示意图。

现对图 7.8 说明如下：

(1) 在 Xxx.cpp 中定义了一个全局的 CXxxApp 类的对象 theApp，从而定义了应用程序 CXxxApp 的一个实例。在 AfxWinMain()接收控制之前，首先要构造一个 theApp。CXxxApp 的构造函数是个空函数，什么也不做，于是编译器将调用类 CXxxApp 的基类 CWinApp 的构造函数。

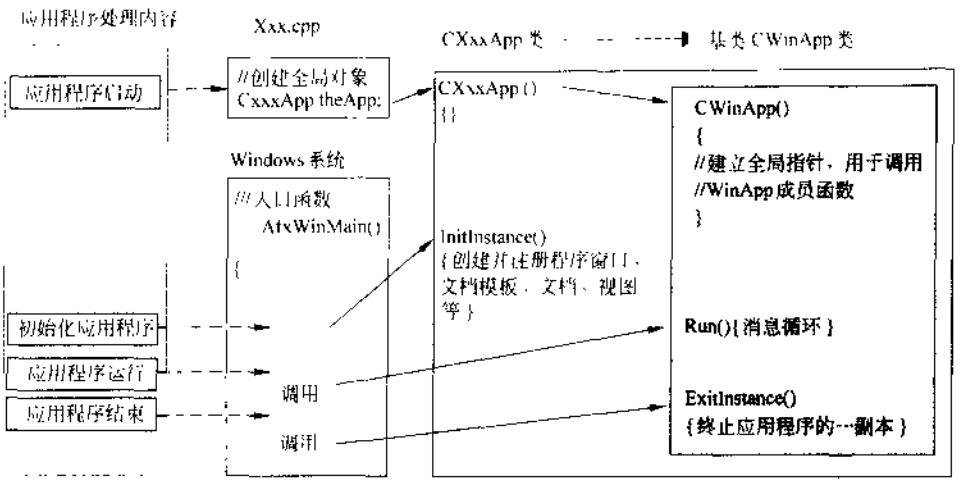

图 7.8　应用程序的流程控制

(2) CWinApp 的构造函数将 CWinApp 类对象 theApp 的地址保存在一个全局指针(pCXxxApp)中，以便以后调用 CWinApp 类对象的成员函数。

(3) AfxWinMain()开始接收控制。

初始化：AfxWinMain()调用 CXxxApp 类对象的成员函数 InitInstance()，初始化应用程序，构造文档模板，产生最初的文档、视图和主框架窗口等。

运行应用程序：AfxWinMain()调用 CXxxApp 类对象的成员函数 Run()，处理消息循环。

结束应用程序：AfxWinMain()调用 CXxxApp 类对象的成员函数 ExitInstance()。

7.2　例题示范

例 7.2.1　基于 Windows API 编写一个窗口应用程序。该程序在窗口的标题栏显示"Windows API 函数程序设计示例"标题，在窗口内输出一行红色文字"欢迎学习、使用 Windows API 编程"。

解：应用 Windows API 编程是可视化编程的基础，它不同于利用 MFC 编程。利用 MFC 编程时，部分框架的代码是由系统直接生成的，而利用 Windows API 函数进行编程，所有的代码需要用户自己完成，虽然编写程序比较繁杂，但能从最根本、最底层上了解可视化编程的整个体系。因此，要完全掌握可视化编程，学习 Windows API 编程是必不可少的环节。该窗口应用程序主要用到以下 API 函数，其函数原型如下：

```
HWND WINAPI CreateWindow (DWORD dwExStyle,
                          LPCSTR lpClassName,
                          LPCSTR lpWindowName,
                          DWORD dwStyle,
                          int X,
                          int Y,
```

```
                                int nWidth,
                                int nHeight,
                                HWND hWndParent ,
                                 HMENU hMenu,
                                 HINSTANCE hInstance,
                                 LPVOID lpParam);
BOOL WINAPI GetMessage (LPMSG lpMsg,
                                 HWND hWnd ,
                                 UINT wMsgFilterMin,
                                 UINT wMsgFilterMax);
BOOL WINAPI TranslateMessage(CONST MSG * lpMsg);
LONG WINAPI DispatchMessage (CONST MSG * lpMsg);
BOOL WINAPI ShowWindow(HWND hWnd, int nCmdShow);
BOOL WINAPI UpdateWindow(HWND hWnd);
HFONT WINAPI CreateFont (int, int, int, int, int, DWORD,
                                 DWORD, DWORD, DWORD, DWORD,
                                 DWORD, DWORD, DWORD, LPCSTR);
BOOL WINAPI TextOutA(HDC, int, int, LPCSTR, int);
VOID WINAPI PostQuitMessage(int nExitCode);
HGDIOBJ WINAPI SelectObject(HDC, HGDIOBJ);
BOOL WINAPI GetTextMetrics (HDC, LPTEXTMETRICA);
BOOL WINAPI DeleteObject(HGDIOBJ);
BOOL WINAPI EndPaint( HWND hWnd, CONST PAINTSTRUCT * lpPaint);
VOID WINAPI PostQuitMessage(int nExitCode);
LRESULT WINAPID WindowProc ( HWND hWnd, UINT Msg,
                                 WPARAM wParam, LPARAM lParam);
ATOM WINAPI RegisterClass (CONST WNDCLASSEXA * );
HICON WINAPI LoadIcon(HINSTANCE hInstance, LPCSTR lpIconName);
HCURSOR WINAPI LoadCursor (HINSTANCE hInstance,LPCSTR lpCursorName);
```

WinMain 函数是所有 Windows 函数的入口，类似 C 语言的 main 函数，其功能是完成一系列的定义和初始化工作，并产生消息循环。WinMain 函数和 WndProc 函数构成所需的基本框架，窗口 WndProc 函数负责接收和处理由 WinMain 函数发送的消息。

WinMain 函数主要由以下 3 个部分组成：

(1) 定义并注册窗口类；

(2) 建立窗口；

(3) 产生消息循环。

本程序的 WinMain 函数设计如下：

```
int WINAPI WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPSTR     lpCmdLine,
                int       nCmdShow)
{
```

```
  MSG msg;                                    //声明消息结构变量
  MyRegisterClass(hInstance);
                                              // 初始化
  if (! InitInstance (hInstance, nCmdShow))
  {
      return FALSE;
  }
                                              // 产生消息循环
  while (GetMessage(&msg, NULL, 0, 0))
  {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
  }
  return msg.wParam;
}
```

WinMain 函数和 WndProc 函数构成所需的基本框架，窗口 WndProc 函数负责接收和处理由 WinMain 函数发送的消息。

```
LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
  HDC hdc;                                    //声明设备环境句柄
  HFONT hF;                                   //声明字体句柄
  TEXTMETRIC tm;                              //声明包含字体信息的结构体变量
  PAINTSTRUCT ps;                             //声明包含字体信息的结构体变量
char show[]="欢迎学习、使用 Windows API 编程";  //设置输出文字
  char cFont[7];                              //定义包含字体的字符串变量
  int x=100,y=100;                            //设置输出坐标

  switch (message)                            //处理消息
  {
      case WM_PAINT:                          //处理绘图消息
         hdc=BeginPaint(hWnd,&ps);            //获取设备环境句柄
         SetTextColor(hdc,RGB(255,0,0));
                                              //设置字体颜色为红色
         strcpy(cFont,"黑体");                //设置字体
         hF=CreateFont(                       //获得字体句柄
             40,                              //字体高度
             20,                              //字体宽度
             0,                               //文本水平
             0,                               //字体斜度为 0
             500,                             //字体粗度
```

```
            0,                                  //字体不倾斜
            0,                                  //无下划线
            0,                                  //无中划线
            GB2312_CHARSET,                     //字符集
            OUT_DEFAULT_PRECIS,                 //输出精度默认
            CLIP_DEFAULT_PRECIS,                //建材精度默认
            DEFAULT_QUALITY,                    //输出质量默认
            DEFAULT_PITCH|FF_DONTCARE,
                                                //间距默认
            cFont);                             //字体名称
        SelectObject(hdc,hF);                   //选入字体
        GetTextMetrics(hdc,&tm);                //获得字体信息
        TextOut(hdc,x,y,&show[0],strlen(show));
                                                //输出文字
        DeleteObject(hF);                       //删除字体
        EndPaint(hWnd,&ps);                     //删除设备环境句柄
        break;
    case WM_DESTROY:                            //结束
        PostQuitMessage(0);                     //发送一个脱离消息
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

步骤 1：建立一个 Win32 应用程序工程。

(1) 运行 VC++ 6.0。在开发平台中单击"File"菜单中的"New"选项。弹出图 7.9 所示的"New"对话框。

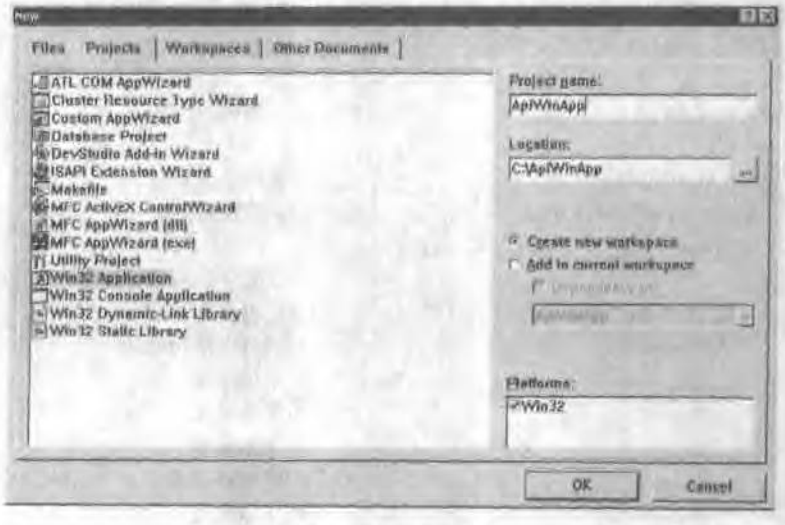

图 7.9 "New"对话框

(2) 在对话框中选择“Projects”选项卡，且在窗口中列出的一系列项目类型中选择“Win32 Application”选项。

(3) 在 Project name：编辑框中输入项目名称。

(4) 再单击对话框中的“OK”命令按钮，弹出图 7.10 所示的对话框。在对话框中选择“A simple win32 application”选项，最后单击对话框中的“Finish”命令按钮，结束项目空间的建立。

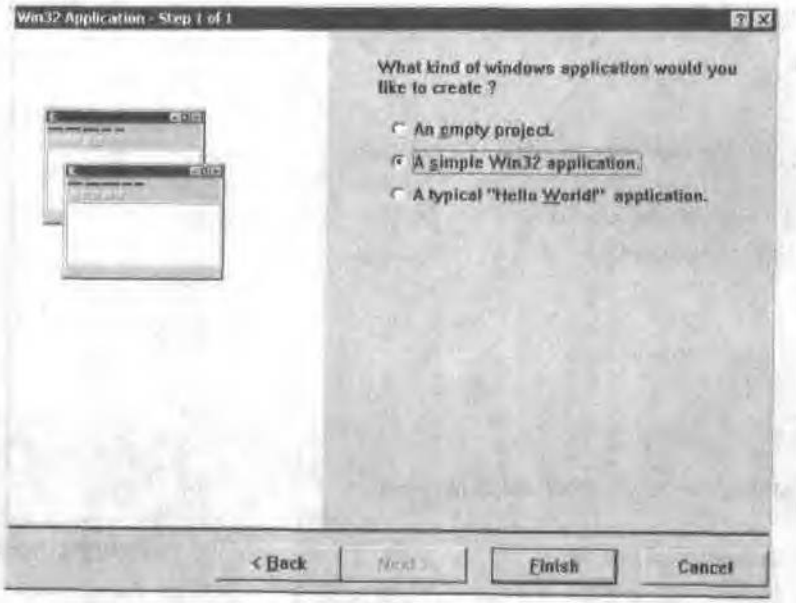

图 7.10 Step 1 对话框

步骤 2：打开 ApiWinApp.cpp 文件添加程序代码，具体的源程序代码如下：

```
#include "windows.h"

//设置窗口类名称
TCHAR szWindowClass[]="ApiWinApp";

//函数超前声明
UINT                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance,
            HINSTANCE hPrevInstance,
            LPSTR     lpCmdLine,
            int       nCmdShow)
{
```

```
    MSG msg;                                                    //声明消息结构变量

    MyRegisterClass(hInstance);

                                                                // 初始化
    if (! InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

                                                                // 处理消息循环
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}


                                                                //注册窗口类
UINT MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;                                            //声明窗口类结构变量

    wcex.cbSize = sizeof(WNDCLASSEX);                           //窗口类尺寸

    wcex.style          = CS_HREDRAW | CS_VREDRAW;              //窗口风格
    wcex.lpfnWndProc    = (WNDPROC)WndProc;                     //窗口消息处理函数
    wcex.cbClsExtra     = 0;                                    //窗口类扩展无
    wcex.cbWndExtra     = 0;                                    //窗口实例扩展无
    wcex.hInstance      = hInstance;                            //当前实例句柄
wcex.hIcon          = LoadIcon(wcex.hInstance, NULL);
                                                                //图标
    wcex.hCursor        =LoadCursor(NULL,IDC_ARROW);            //光标
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);             //背景颜色
    wcex.lpszMenuName   = NULL;                                 //菜单
    wcex.lpszClassName  = szWindowClass;                        //窗口类名称
    wcex.hIconSm        = LoadIcon(wcex.hInstance, NULL);
                                                                //图标 Win 95 以上版本应用
    return RegisterClassEx(&wcex);
}


                                                                //初始化函数建立主窗口
```

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;                                    //声明窗口句柄

  hWnd = CreateWindow(                          //获取窗口句柄
                szWindowClass,                  //窗口类名称
                "Windows API 函数程序设计示例",
                                                //标题栏名称
                WS_OVERLAPPEDWINDOW,            //窗口样式
                CW_USEDEFAULT,                  //窗口左上角坐标
                0,
                CW_USEDEFAULT,                  //窗口高度和宽度默认
                0,
                NULL,                           //无父窗口
                NULL,                           //无菜单
                hInstance,                      //当前应用实例句柄
                NULL);                          //指向传递给窗口的参数的指针

  if (! hWnd)                                   //创建失败返回
  {
     return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);                   //显示窗口
  UpdateWindow(hWnd);                           //刷新用户区

  return TRUE;
}

                                                //主窗口消息处理函数
LRESULT WINAPI
WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
HDC hdc;                                        //声明设备环境句柄
HFONT hF;                                       //声明字体句柄
TEXTMETRIC tm;                                  //声明包含字体信息的结构体变量
PAINTSTRUCT ps;                                 //声明包含字体信息的结构体变量
char show[]="欢迎学习、使用 Windows API 编程";
                                                //设置输出文字
char cFont[7];                                  //定义包含字体的字符串变量
int x=100,y=100;                                //设置输出坐标

switch (message)                                //处理消息
```

```
{
        case WM_PAINT:                              //处理绘图消息
            hdc=BeginPaint(hWnd,&ps);               //获取设备环境句柄
            SetTextColor(hdc,RGB(255,0,0));         //设置字体颜色为红色
            strcpy(cFont,"黑体");                   //设置字体
            hF=CreateFont(                          //获得字体句柄
                40,                                 //字体高度
                20,                                 //字体宽度
                0,                                  //文本水平
                0,                                  //字体斜度为0
                500,                                //字体粗度
                0,                                  //字体不倾斜
                0,                                  //无下划线
                0,                                  //无中划线
                GB2312_CHARSET,                     //字符集
                OUT_DEFAULT_PRECIS,                 //输出精度默认
                CLIP_DEFAULT_PRECIS,                //建材精度默认
                DEFAULT_QUALITY,                    //输出质量默认
                DEFAULT_PITCH|FF_DONTCARE,          //间距默认
                cFont);                             //字体名称
            SelectObject(hdc,hF);                   //选入字体
            GetTextMetrics(hdc,&tm);                //获得字体信息
            TextOut(hdc,x,y,&show[0],strlen(show));//输出文字
            DeleteObject(hF);                       //删除字体
            EndPaint(hWnd,&ps);                     //删除设备环境句柄
            break;
        case WM_DESTROY:                            //结束
            PostQuitMessage(0);                     //发送一个脱离消息
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

步骤3：编译运行该程序，其结果如图7.11所示。

例7.2.2 基于MFC类库编写一个单文档SDI应用程序，该程序在窗口中输出一行文字"欢迎学习、掌握MFC AppWizard编程技术"。

解：由于采用Windows API编程，程序的编写必须从头做起，所需的所有的代码均需要用户自己完成。Visual C++为了减轻程序员的工作量，提供了AppWizard应用程序向导功能。AppWizard应用程序向导为程序员提供了一个基于MFC的应用程序框架。程序员只要在此基础上添加实现特定功能的代码即可。

本题主要目的是学习利用MFC AppWizard建立一个文档/视图结构的程序。通过对

图 7.11

该程序的学习初步掌握利用 MFC AppWizard 建立一个文档/视图结构的程序的工作步骤。

步骤 1：

(1) 运行 VC++6.0。在开发平台中单击"File"菜单中的"New"选项。弹出图 7.12 所示"New"对话框。

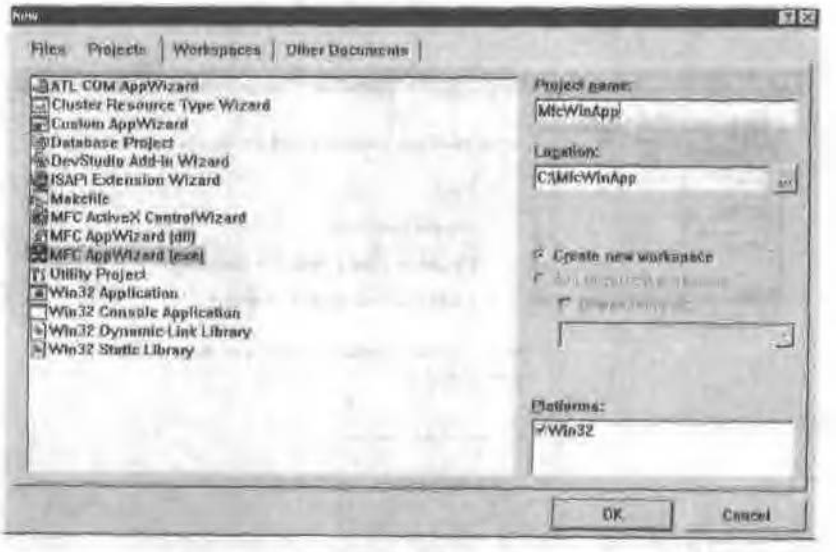

图 7.12 "New"对话框

(2) 在对话框中选择"Projects"选项卡，且在窗口中列出的一系列项目类型中选择"MFC AppWizard(exe)"选项。

(3) 在 Project name：编辑框中输入工程名称，选择"OK"按钮弹出"MFC AppWizard-Step 1"对话框，如图 7.13 所示。

步骤 2：在"MFC AppWizard-Step 1"对话框中设置应用程序的类型为 Single document，选择"Next"按钮，弹出"MFC AppWizard-Step 2 of 6"对话框，如图 7.14 所示。

步骤 3：在"MFC AppWizard-Step 2 of 6"对话框中选择"Next"按钮，弹出"MFC

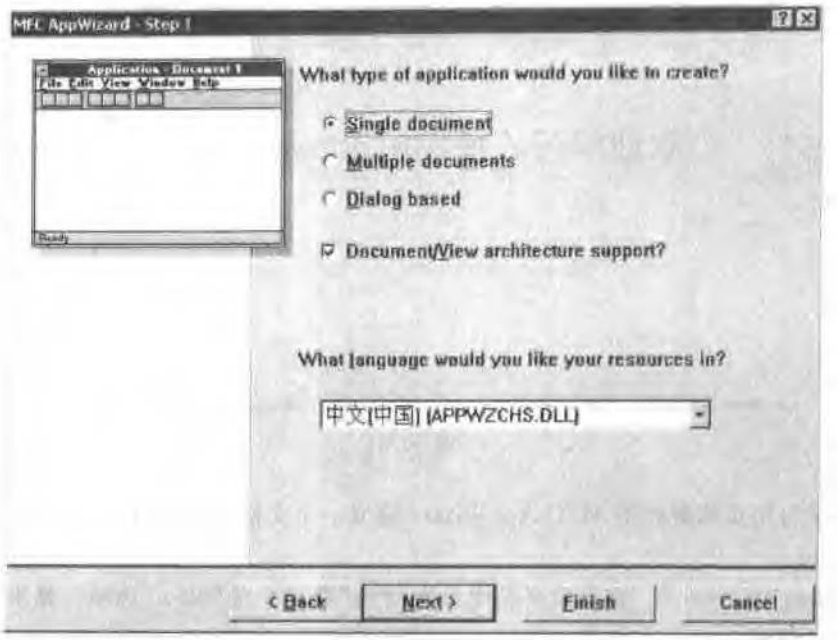

图 7.13 "Step 1"对话框

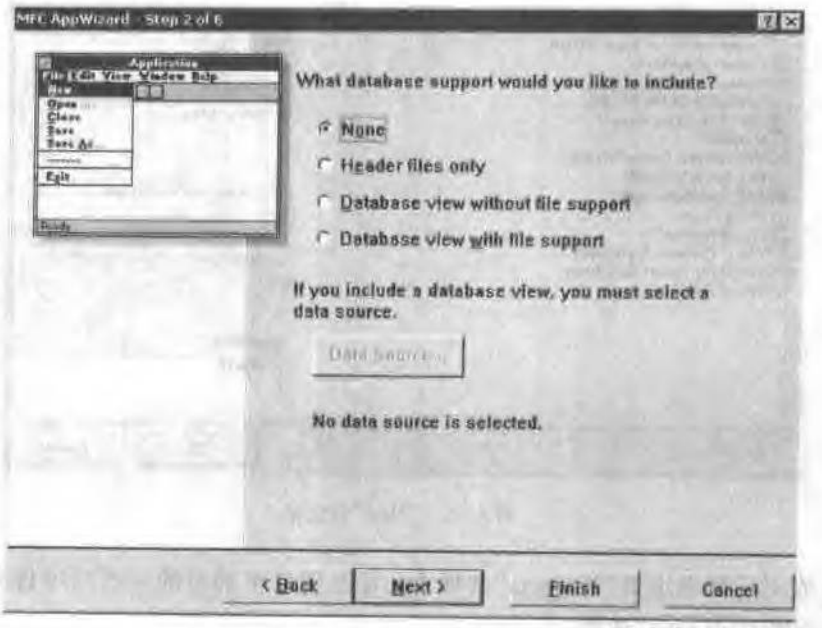

图 7.14 "Step 2"对话框

AppWizard-Step 3 of 6"对话框，如图 7.15 所示。

步骤 4：在"MFC AppWizard-Step 3 of 6"对话框中选择"Next"按钮，弹出"MFC AppWizard-Step 4 of 6"对话框，如图 7.16 所示。

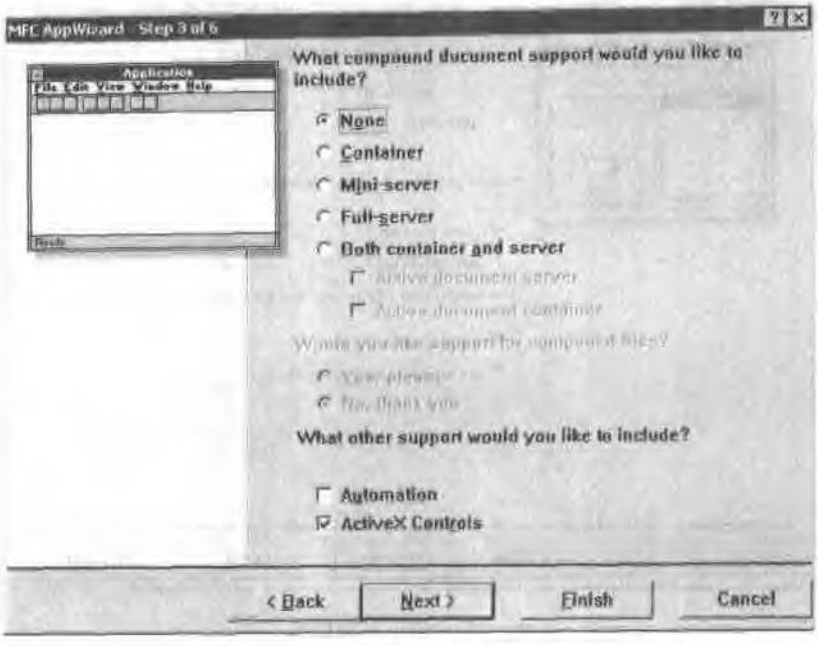

图 7.15 "Step 3"对话框

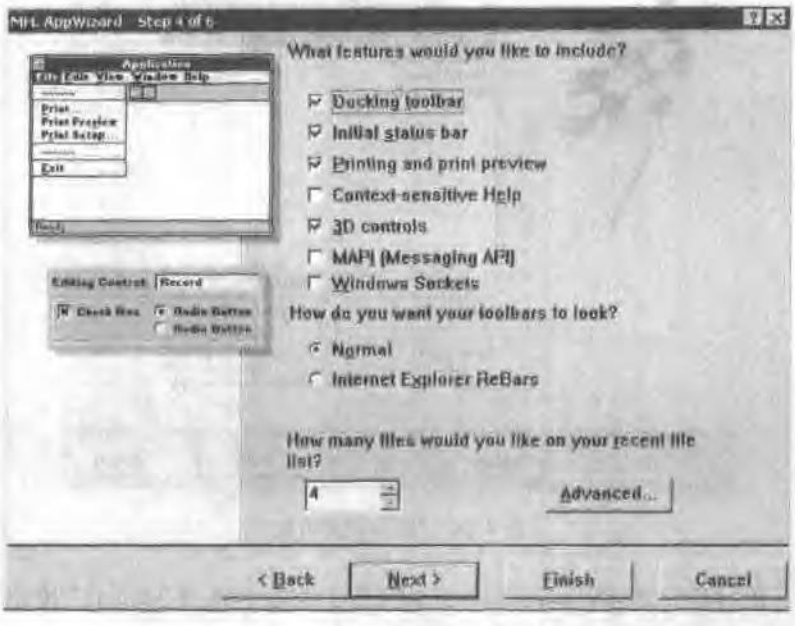

图 7.16 "Step 4"对话框

步骤 5：在"MFC AppWizard-Step 4 of 6"对话框中选择"Next"按钮，弹出"MFC AppWizard-Step 5 of 6"对话框，如图 7.17 所示。

步骤 6：在"MFC AppWizard-Step 5 of 6"对话框中选择"Next"按钮，弹出"MFC AppWizard-Step 6 of 6"对话框，如图 7.18 所示。

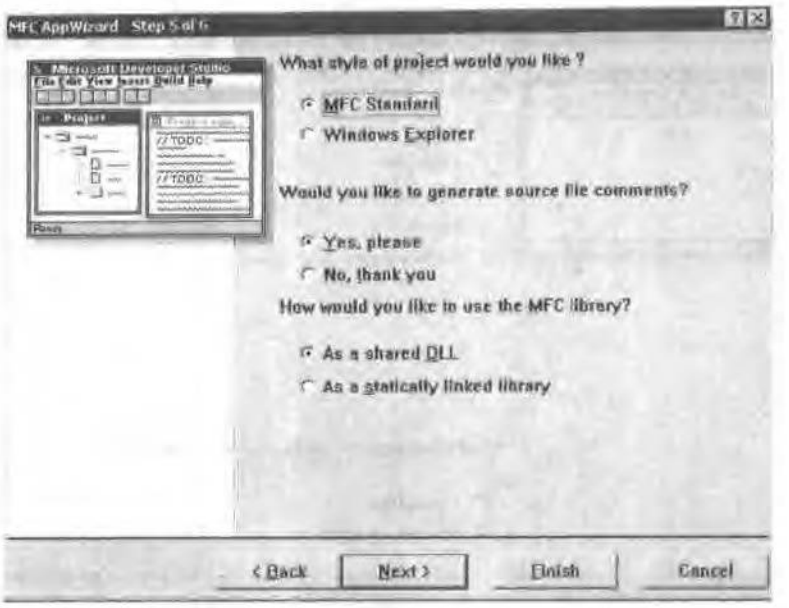

图 7.17 "Step 5"对话框

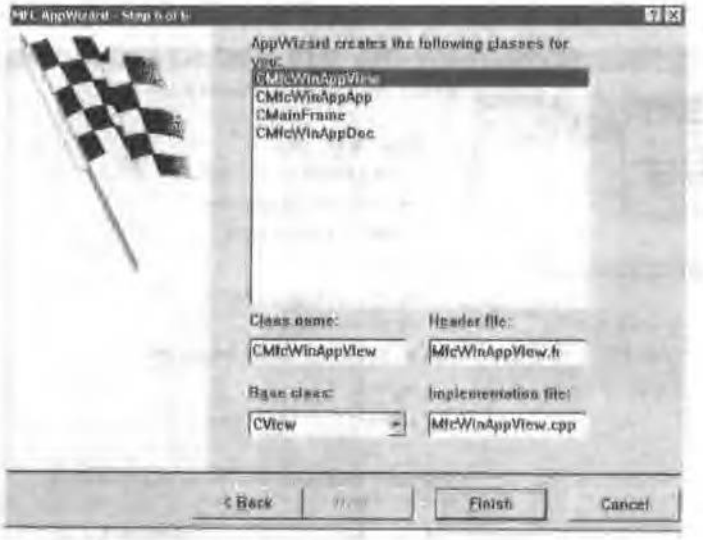

图 7.18 "Step 6"对话框

步骤 7：在"MFC AppWizard-Step 6 of 6"对话框中选择"Finish"按钮弹出"New Project Information"对话框，如图 7.19 所示。

该对话框中列出了关于建立的应用程序工程文件的相关信息，最后选择该对话框中的 OK 按钮，完成主窗口框架程序的建立。

步骤 8：打开 Workspace 工作空间，单击"ClassView"选项卡，在该选项卡中选择视图类 CmfcWinAppView 中的 OnDraw 函数，如图 7.20 所示。在该函数中添加如下代码：

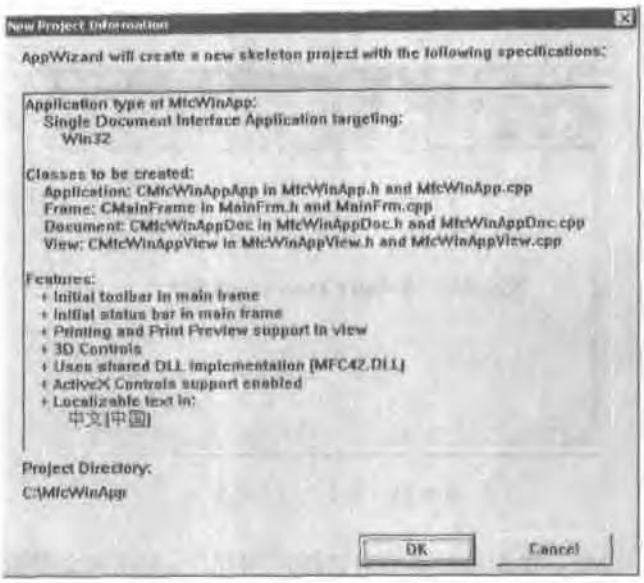

图 7.19 "New Project Information"对话框

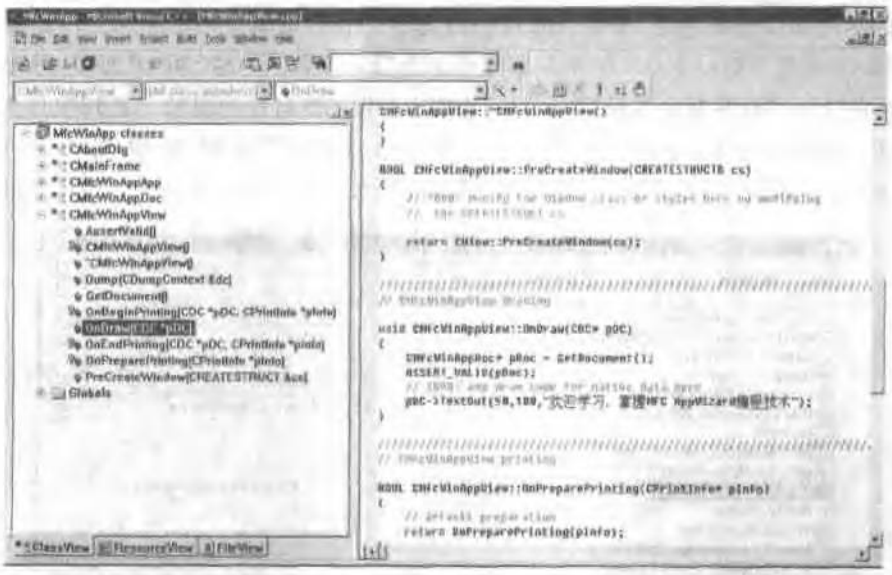

图 7.20 选择 OnDraw 函数

```
void CMfcWinAppView::OnDraw(CDC * pDC)
{
  CMfcWinAppDoc * pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  // TODO: add draw code for native data here
  pDC->TextOut(50,100,"欢迎学习,掌握 MFC AppWizard 编程技术");
}
```

步骤 9：编译运行该程序，其结果如图 7.21 所示。

图 7.21 例 7.2.2 运行结果

例 7.2.3 基于 MFC 类库编写一个对话框应用程序，实现两个整数的加、减、乘、除算术运算。

解：本题主要目的是学习利用 MFC AppWizard 建立一个对话框结构的程序，通过对该程序的学习初步掌握利用 MFC AppWizard 建立一个对话框应用程序的工作步骤。

步骤 1：运行 VC++ 6.0。在开发平台中单击"File"菜单中的"New"选项，弹出"New"对话框。在"New"对话框中选择"Projects"选项卡，且在窗口中列出的一系列项目类型中选择"MFC AppWizard (exe)"选项。在"Project name:"编辑框中输入工程名称"MfcDlgApp"，如图 7.22 所示。选择"OK"按钮，进入"MFC AppWizard-Step 1"对话框。

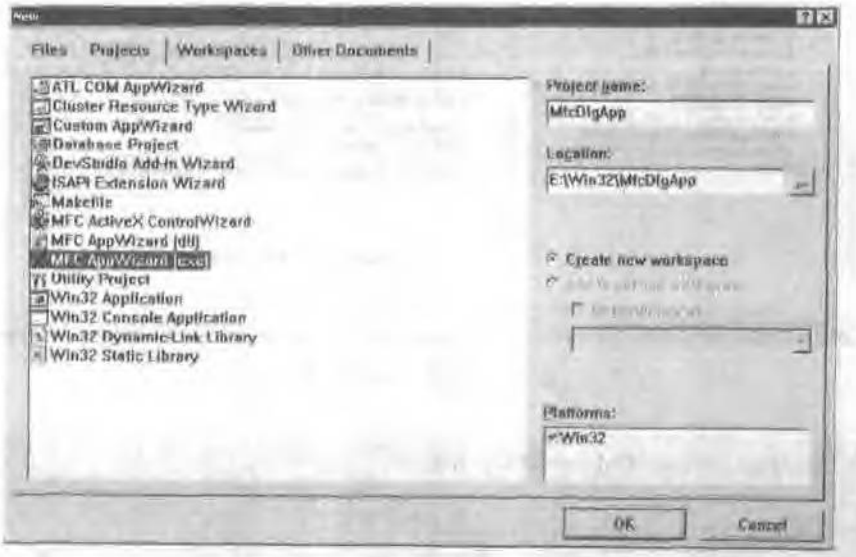

图 7.22 "New"对话框

步骤 2：在"MFC AppWizard-Step 1"对话框中设置应用程序的类型为 Dialog based，如图 7.23 所示。选择"Next"按钮，进入"MFC AppWizard-Step 2 of 4"对话框。

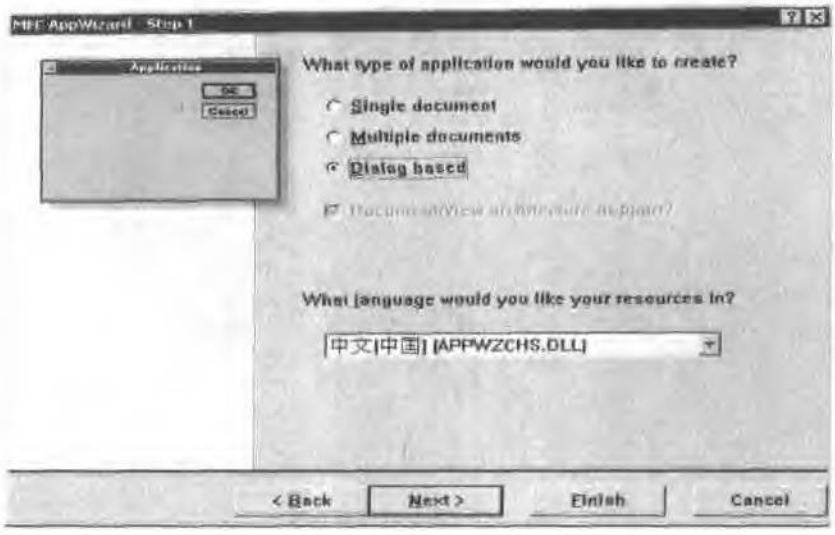

图 7.23 "Step 1"对话框

步骤 3：在依次弹出的"AppWizard-Step 2 of 4"和"AppWizard-Step 3 of 4"对话框中选择"Next"按钮进入"MFC AppWizard-Step 4 of 4"对话框，如图 7.24 所示。

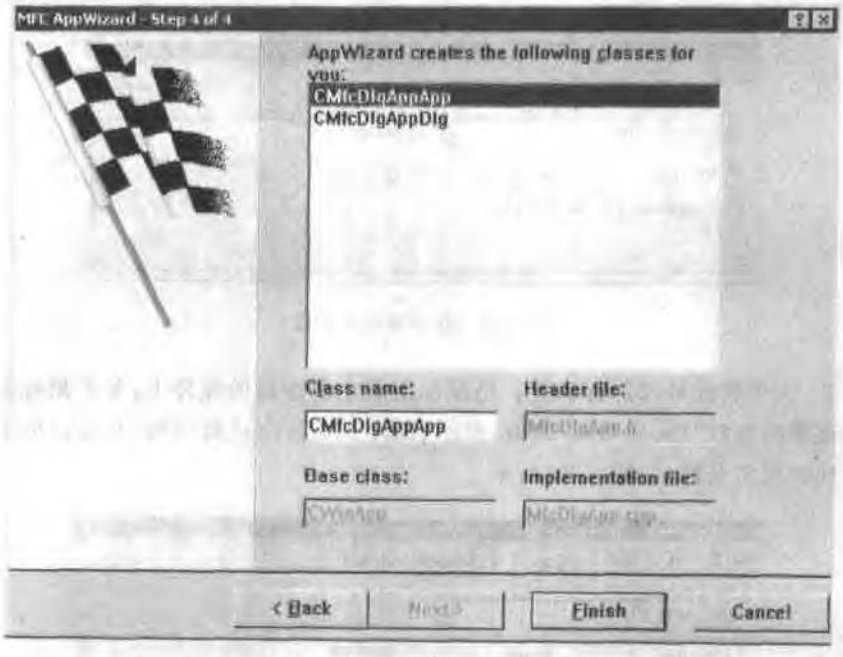

图 7.24 "Step 4"对话框

步骤 4：在"MFC AppWizard-Step 4 of 4"对话框中选择"Finish"按钮，再弹出的"New Project Information"对话框中选择"OK"按钮，完成对话框应用程序工程的建立。

步骤 5：打开对话框资源窗口，在窗口中添加控件，如图 7.25 所示。

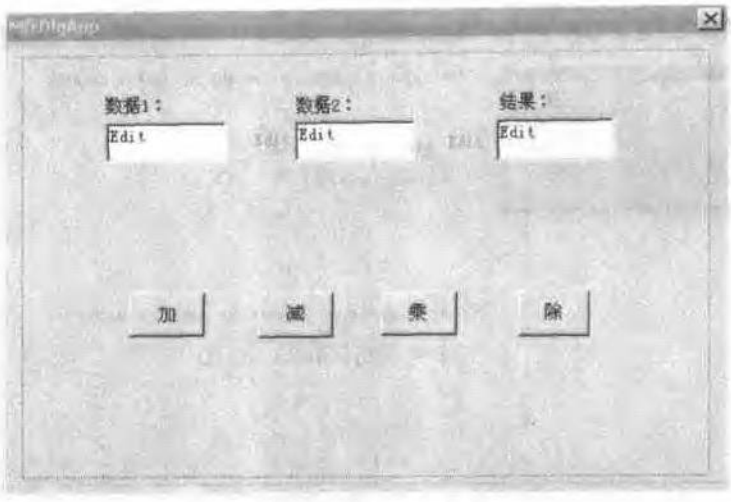

图 7.25　添加控件对话框

步骤 6：为按钮控件添加名称和标识符。把鼠标定位在所编辑的控件上，单击鼠标右键，在弹出的编辑菜单中选择“Properties”选项，弹出“Push Button Properties”对话框，在对话框中的“ID”编辑框中添加标识符名称，再在“Caption”编辑框中添加控件名称，如图 7.26 所示。

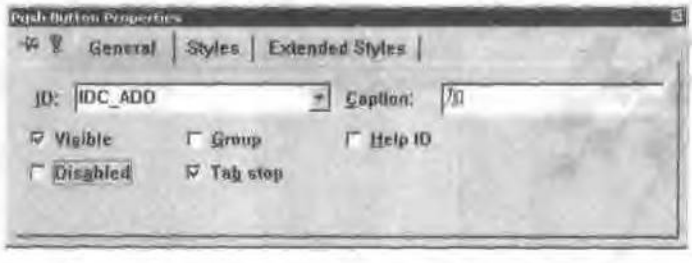

图 7.26　按钮属性对话框

步骤 7：为编辑控件添加标识符。把鼠标定位在所编辑的控件上，单击鼠标右键，在弹出的编辑菜单中选择“Properties”选项，弹出“Edit Properties”对话框，在对话框中的 ID 编辑框中添加标识符名称，如图 7.27 所示。

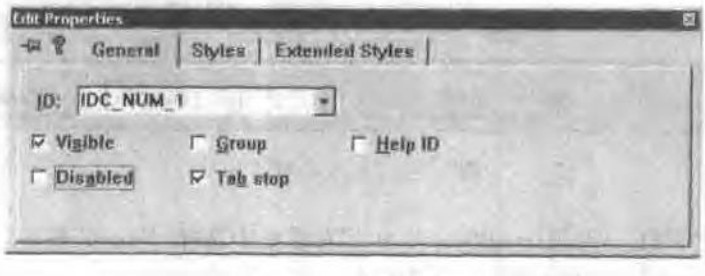

图 7.27　编辑属性对话框

步骤 8：为编辑控件标识符追加变量名。选择“ClassWizard”菜单项，在弹出的“MFC ClassWizard”对话框中选择“Member Variables”选项卡，在 Control Ids 标识符列表中选择追加变量的标识符，如图 7.28 所示，再单击“Add Variable”按钮，弹出“Add Member Variable”对话框。在“Add Member Variable”对话框中添加变量名称和选择类型后单击按钮，完成对一个标识符追加变量名的操作，如图 7.29 所示，重复此项操作直至所有标识符的变量名追加完毕为止。

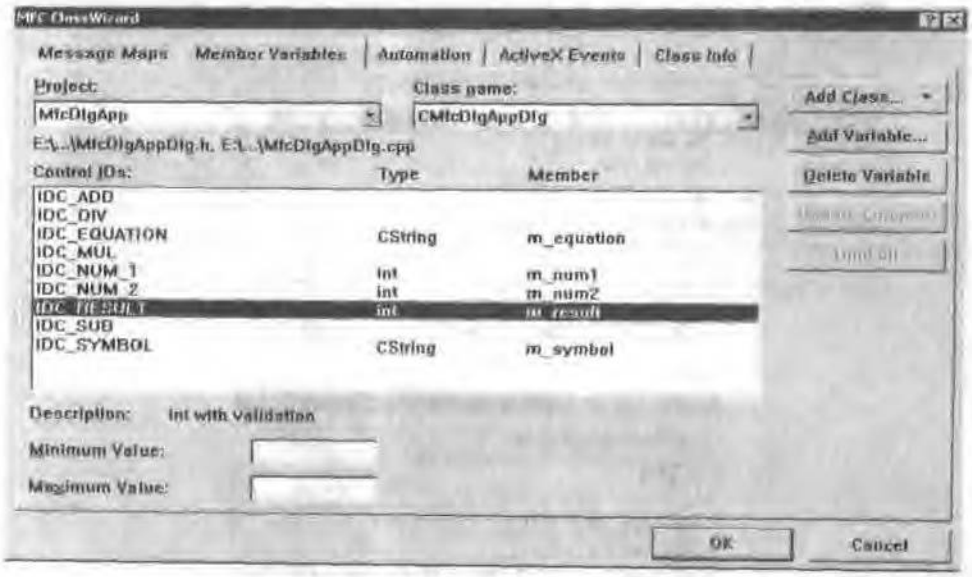

图 7.28 “MFC ClassWizard”工具对话框

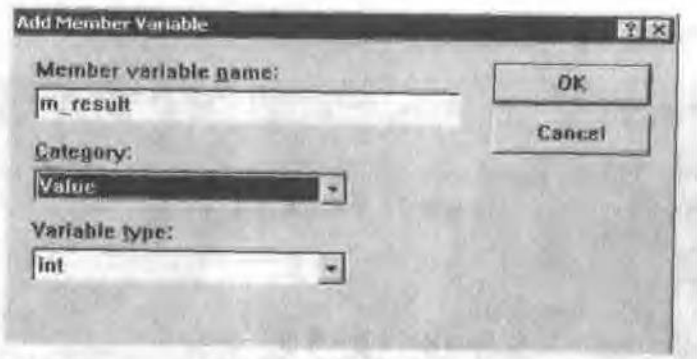

图 7.29 添加成员变量对话框

步骤 9：为按钮控件添加消息映射函数，选择“ClassWizard”菜单项，在弹出的“MFC ClassWizard”对话框中选择“Message Maps”选项卡，在 Object IDs 标识符列表中选择添加函数的标识符，在 Message 列表中选择“BN_CLICK”，如图 7.30 所示，再单击“Add Function”按钮，弹出“Add Member Function”对话框。在“Add Member Function”对话框中单击“OK”按钮，完成对一个按钮控件添加消息映射函数的操作，如图 7.31 所示，重复此项操作，直至所有按钮控件消息映射函数添加完毕为止。

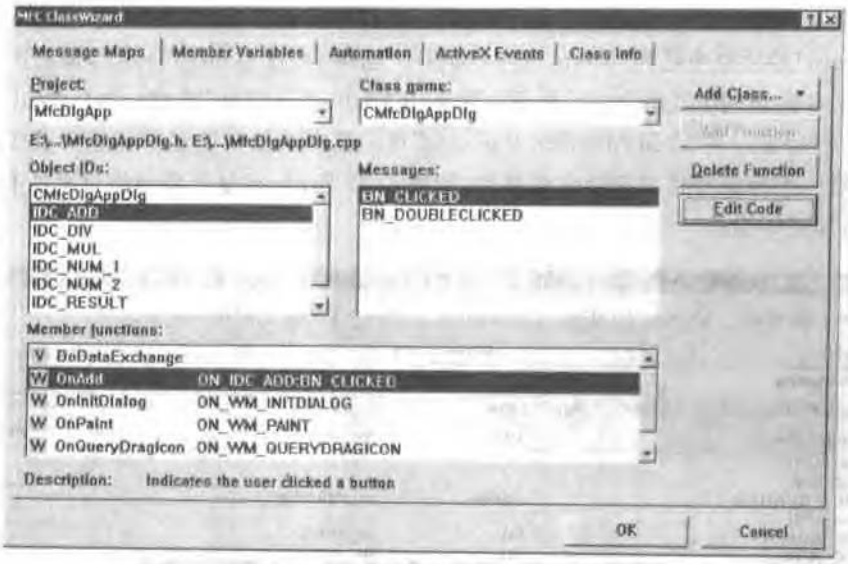

图 7.30 "MFC ClassWizard"工具对话框

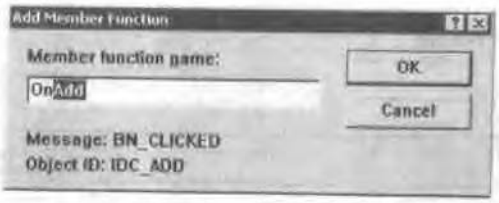

图 7.31 添加成员函数对话框

步骤 10：为消息映射函数添加如下代码。

```
void CMfcDlgAppDlg::OnAdd()
{
    // TODO: Add your control notification handler code here
    UpdateData();          //获取编辑框录入的数据
    m_symbol="+";
    m_equation="=";
    m_result=m_num1+m_num2;
    UpdateData(FALSE);  //在编辑框中显示数据
}

void CMfcDlgAppDlg::OnSub()
{
    // TODO: Add your control notification handler code here
    UpdateData();
    m_symbol="-";
    m_equation="=";
    m_result=m_num1-m_num2;
```

```
    UpdateData(FALSE);
}

void CMfcDlgAppDlg::OnMul()
{
    // TODO: Add your control notification handler code here
    UpdateData();
    m_symbol="*";
    m_equation="=";
    m_result=m_num1 * m_num2;
    UpdateData(FALSE);
}

void CMfcDlgAppDlg::OnDiv()
{
    // TODO: Add your control notification handler code here
    UpdateData();
    m_symbol="÷";
    m_equation="=";
    if(m_num2)
      m_result=m_num1/m_num2;
    UpdateData(FALSE);
}
```

步骤 11：编译运行该程序，其结果如图 7.32 所示。

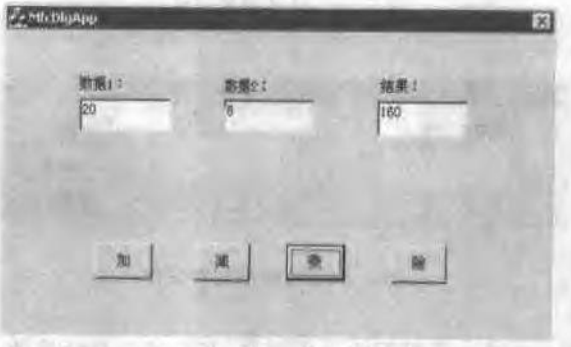

图 7.32 例 7.2.3 运行结果

7.3 自我测试

7.3.1 自测习题

题 7.3.1 基于 Windows API 编写一个窗口应用程序，该程序在窗口的标题栏显示“Windows API 函数程序设计显示大、小写英文字母”标题，在窗口内输出一行小写英文字母，另起一行输出大写英文字母。

题 7.3.2 基于 MFC 类库编写一个应用程序，用来实现鼠标的响应，创建鼠标左键双击和右键单击的消息映射，并用对话框显示出来，当双击左键时显示“鼠标左键双击已经响应！”，单击右键时显示“鼠标右键单击已经响应！”。

题 7.3.3 基于 MFC 类库编写一个应用程序，用来捕捉鼠标的移动轨迹，并在屏幕上显示出来。

题 7.3.4 在视图窗口的相邻几行按字体从小到大输出字符串“欢迎使用 VC++！”。程序要求 SDI 类型，且视图类的基类是 Cview 类。

题 7.3.5 编写一个程序，要求在窗口中利用鼠标画多个矩形，当按下鼠标左键移动鼠标时显示矩形的虚线，松开鼠标时显示实线，并且能够保存到文件中。

题 7.3.6 创建一个应用程序，在程序运行时在视图区右击，弹出一个快捷菜单。

题 7.3.7 编写一个应用程序，能在状态栏中显示鼠标的位置。

题 7.3.8 创建一个可伸缩的对话框，单击扩展按钮后对话框如图 7.33 所示，单击收缩按钮后对话框如图 7.34 所示。

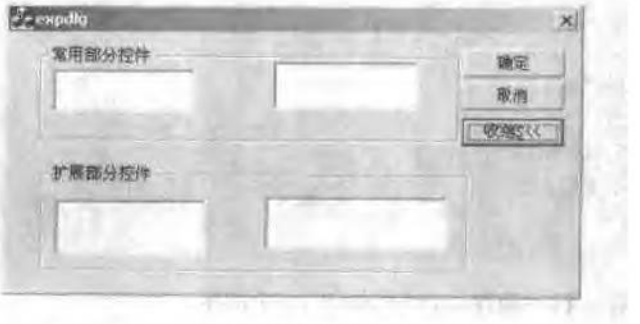

图 7.33 可伸缩的对话框

图 7.34 收缩后的对话框

题 7.3.9 创建一个单文档界面的应用程序，在它的工具条按钮下面显示按钮文字。

题 7.3.10 基于 MFC 类库编写一个具有一个自定义活动工具条的应用程序，工具条由 A、B、C 3 个按钮组成，分别单击按钮 A、B、C，则在主窗口 3 个不同行位置书写从小到大 3 行相同文字“Visual C++ 6.0 编程实践”。

7.3.2 参考答案

题 7.3.1 参考答案：

解：本题主要是利用 Windows API 直接编写一个窗口应用程序的例子。

步骤 1：建立一个 Win32 应用程序工程，工程名为“WinApiApp”。

步骤 2：打开 WinApiApp.cpp 文件添加程序代码，具体的源程序代码如下：

```
#include "windows.h"

//设置窗口类名称
TCHAR szWindowClass[]="WinApiApp";


//函数超前声明
UINT                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR     lpCmdLine,
                   int       nCmdShow)
{
    MSG msg;                                        //声明消息结构变量

    MyRegisterClass(hInstance);

    // 初始化
    if (! InitInstance (hInstance, nCmdShow))
    {
     return FALSE;
    }

    // 处理消息循环
    while (GetMessage(&msg, NULL, 0, 0))
    {
     TranslateMessage(&msg);
     DispatchMessage(&msg);
    }

    return msg.wParam;
}
//注册窗口类
UINT MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;                                //声明窗口类结构变量


    wcex.cbSize = sizeof(WNDCLASSEX);               //窗口类尺寸
```

```
    wcex.style   = CS_HREDRAW | CS_VREDRAW;          //窗口风格
    wcex.lpfnWndProc= (WNDPROC)WndProc;              //窗口消息处理函数
    wcex.cbClsExtra= 0;                              //窗口类扩展无
    wcex.cbWndExtra= 0;                              //窗口实例扩展无
    wcex.hInstance= hInstance;                       //当前实例句柄
    wcex.hIcon   = LoadIcon(wcex.hInstance, NULL);   //图标
    wcex.hCursor= LoadCursor(NULL,IDC_ARROW);        //光标
    wcex.hbrBackground   = (HBRUSH)(COLOR_WINDOW+1);  //背景颜色
    wcex.lpszMenuName   = NULL;                      //菜单
    wcex.lpszClassName   = szWindowClass;            //窗口类名称
    wcex.hIconSm=LoadIcon(wcex.hInstance,NULL);      //图标 win 95 以上版本应用
    return RegisterClassEx(&wcex);
}

//初始化函数建立主窗口
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;                                         //声明窗口句柄

  hWnd = CreateWindow(                               //获取窗口句柄
                szWindowClass,                       //窗口类名称
                "Windows API 函数程序设计显示大、小写英文字母",
                                                     //标题栏名称
                WS_OVERLAPPEDWINDOW,                 //窗口样式
                CW_USEDEFAULT,                       //窗口左上角坐标
                0,
                CW_USEDEFAULT,                       //窗口高度和宽度默认
                0,
                NULL,                                //无父窗口
                NULL,                                //无菜单
                hInstance,                           //当前应用实例句柄
                NULL);                               //指向传递给窗口的参数的指针

  if (! hWnd)                                        //创建失败返回
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);                        //显示窗口
  UpdateWindow(hWnd);                                //刷新用户区

  return TRUE;
}
```

```
                                                        //主窗口消息处理函数
LRESULT WINAPI
WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  HDC hdc;                                              //声明设备环境句柄
  HFONT hF;                                             //声明字体句柄
  TEXTMETRIC tm;                                        //声明含字体信息结构体变量
  PAINTSTRUCT ps;                                       //声明含字体信息结构体变量
  char lowercase[]="abcdefghijklmnopqrstuvwxyz";        //设置输出文字
  char capital[]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";          //设置输出文字
  char cFont[7];                                        //定义包含字体的字符串变量
  int x=100,y=100;                                      //设置输出坐标

switch (message)                                        //处理消息
{
      case WM_PAINT:                                    //处理绘图消息
         hdc=BeginPaint(hWnd,&ps);                      //获取设备环境句柄
         SetTextColor(hdc,RGB(255,0,0));                //设置字体颜色为红色
         strcpy(cFont,"黑体");                          //设置字体
         hF=CreateFont(                                 //获得字体句柄
              40,                                       //字体高度
              20,                                       //字体宽度
              0,                                        //文本水平
              0,                                        //字体斜度为 0
              500,                                      //字体粗度
              0,                                        //字体不倾斜
              0,                                        //无下划线
              0,                                        //无中划线
              GB2312_CHARSET,                           //字符集
              OUT_DEFAULT_PRECIS,                       //输出精度默认
              CLIP_DEFAULT_PRECIS,                      //建材精度默认
              DEFAULT_QUALITY,                          //输出质量默认
              DEFAULT_PITCH|FF_DONTCARE,                //间距默认
              cFont);                                   //字体名称
         SelectObject(hdc,hF);                          //选入字体
         GetTextMetrics(hdc,&tm);                       //获得字体信息
         TextOut(hdc,x,y,&lowercase[0],strlen(lowercase));
                                                        //输出文字
         TextOut(hdc,x,y+100,&capital[0],strlen(capital));
                                                        //输出文字
         DeleteObject(hF);                              //删除字体
         EndPaint(hWnd,&ps);                            //删除设备环境句柄
         break;
      case WM_DESTROY:                                  //结束
         PostQuitMessage(0);                            //发送一个脱离消息
```

```
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

步骤 3：编译运行该程序，其结果如图 7.35 所示。

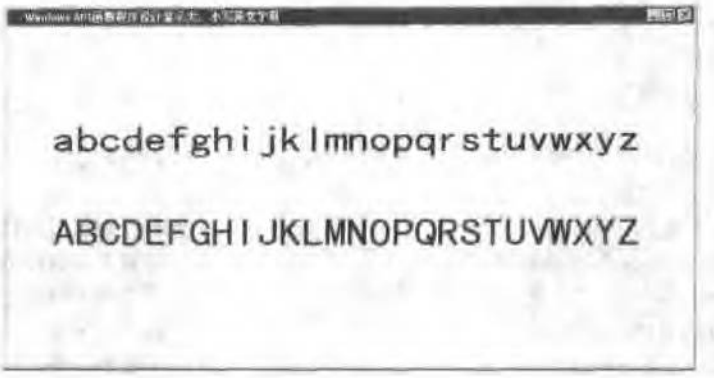

图 7.35　题 7.3.1 运行结果

题 7.3.2 参考答案：

解：本题是处理鼠标单击消息 WM_LBUTTONDBLCLK 和 WM_RBUTTOND DOWN 的一个例子。

步骤 1：运用 MFC AppWizard 创建一个工程名为“MouseResp”单文档应用程序。

步骤 2：利用 ClassWizard 工具在 CmouseRespView 视图类中按如下方法添加鼠标事件消息映射。

打开 ClassWizard 工具，在 Class name 中选择视图类 CmouseRespView，再在 Message 中选择 WM_LBUTTONDBLCLK，然后单击“Add Function”按钮。按此法在 Message 中选择 WM_RBUTTONDDOWN，然后单击“Add Function”按钮。单击“OK”按钮，完成添加鼠标事件消息映射操作，如图 7.36 所示。

步骤 3：在鼠标事件消息映射函数中添加代码

```
void CMouseRespView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
MessageBox("鼠标左键双击已经响应!");
CView::OnLButtonDblClk(nFlags, point);
}
void CMouseRespView::OnRButtonDown(UINT nFlags, CPoint point)
```

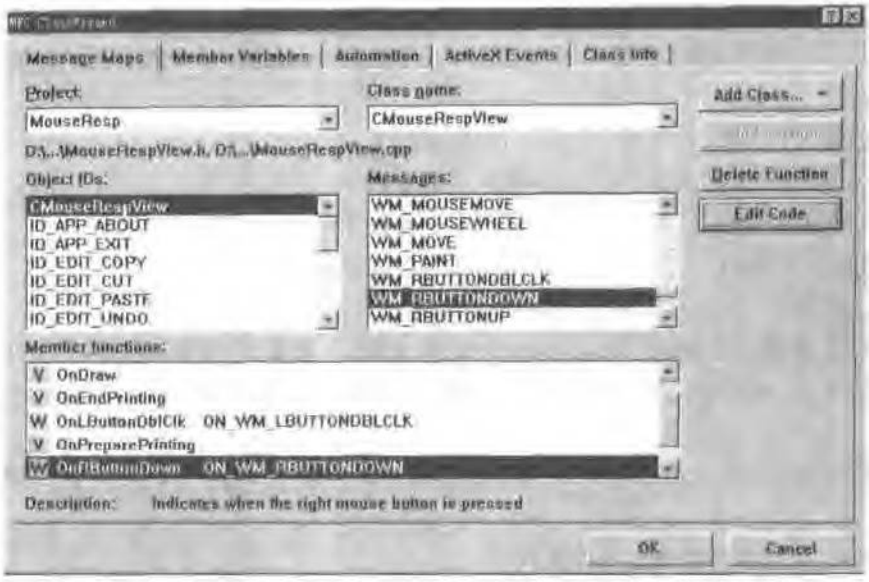

图 7.36 "MFC ClassWizard"工具对话框

```
{
// TODO: Add your message handler code here and/or call default

    MessageBox("鼠标右键单击已经响应!");

    CView::OnRButtonDown(nFlags, point);
}
```

步骤 4：编译并运行程序，双击鼠标左键或单击鼠标右键后，程序运行窗口如图 7.37 所示。

图 7.37 题 7.3.2 运行结果

题 7.3.3 参考答案：

解：本题是处理鼠标移动消息 WM_MOUSEMOVE 的一个例子。

鼠标移动消息 WM_MOUSEMOVE 的消息映像宏处理函数为

afx_msg void OnMouseMove(UINT nFlag,CPoint point);

point 是 Cpoint 类的对象，记录鼠标当前的 x、y 坐标；nFlag 是一个无符号型的整数，它可以取下列值：

MK_CONTROL　　//Ctrl 键被按下
MK_LBUTTON　　//鼠标左键被按下
MK_MBUTTON　　//鼠标中键被按下
MK_RBUTTON　　//鼠标右键被按下
MK_SHIFT　　//Shift 键被按下

步骤 1：新建一个 MFC Appwizard(exe)的工程，工程名为 mousetrack，单击完成。

步骤 2：在 MFC Appwizard-step1 中设置应用程序类型为单个文档，使用的语言资源为中文。

步骤 3：在 MFC Appwizard-step2-6 中都设为默认值。至此，一个应用程序的框架就算完成了。

步骤 4：生成窗口的消息处理函数。

首先，选择"View ClassWizard"选项，打开"MFC ClassWizard"对话框，选择"Message Maps"卡。在"Class name"下拉列表框中选择"CMousetrackView"，在"Object IDs"列表框中选择"CMousetrackView"，在"Messages"列表框中选择"WM_MOUSEMOVE"，单击"Add Function"按钮就生成了处理鼠标移动消息的函数。单击"Edit Code"在函数"OnMouseMove"中添加代码。程序如下所示：

```
void
CMousetrackView::OnMouseMove(UINT nFlags, CPoint point)
{
  // TODO: Add your message handler code here and/or call default
  if(nFlags==MK_LBUTTON)              //如果鼠标左键被按下，则在鼠标当前位置画一个点
  {
      CClientDC dc(this);             //创建一个设备对象 dc
      dc.SetPixel(point,RGB(255,0,0));  //在当前位置输出一个红点
  }
  // my code end here
  CView::OnMouseMove(nFlags, point);
}
```

步骤 5：编译并运行程序。在程序运行时，在窗口中按下鼠标左键就会在鼠标的当前位置画一个红点。若鼠标在窗口中移动时一直按着左键，就会画出鼠标的轨迹。程序测试结果如图 7.38 所示。

题 7.3.4 参考答案：

解：该题主要是验证字体的创建和使用的方法。

本题涉及 Cfont 类。要使用 Cfont 类对象，首先要构造一个 Cfont 对象，再使用 Cfont 类的成员函数 CreateFont、CreateFontIndirect、CreatePointFont 或 CreatePoint FontIndirect，把该对象与 Windows 字体相连，这样就可以使用该对象的成员函数来操纵字体。

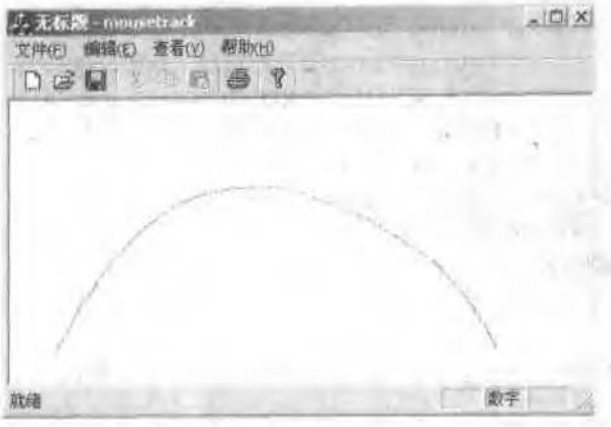

图 7.38 题 7.3.3 运行结果

本例用 CreateFont()函数来创建字体，其函数原型如下：

```
BOOL CFont::CreateFont(int nHeight,        //单位度量的字体高度
int nwidth,                                //逻辑单位度量的字体的平均宽度
        int nEscapement,                   //字体标注行逆时针方向的旋转角度
        int nOrientation,                  //字符的基线与 x 轴的夹角
        int nWeight,                       //字体的粗细程度
        Byte bItalic,                      //确定字体是否为斜体
        BYTE bUnderline,                   //确定字体是否加下划线
        BYTE cStrickout,                   //确定字体是否被穿透
        BYTE nCharSet,                     //指出字体的字符集
        BYTE nOutPrecision,                //指定所要求的裁剪精度
        BYTE nClipPrecision,               //指定所要求的裁剪精度
        BYTE nQuality,                     //指出输出字体的质量
        BYTE nPitAndFamily,                //字体的间距和所属的族
        LPCSTR lpszFacename);              //指向 ASZIIZ 字符串的指针
```

步骤 1：利用 MFC AppWizard(exe)创建应用程序 TestFont。

步骤 2：在 testfontView.h 头文件的 CTestfontView 类中添加 protected 类型的成员函数

```
void WriteText(CDC * pDc, int nX, int nY, int nSize, int nAnge)
```

步骤 3：在 testfontView.cpp 文件中添加 WriteText()函数。WriteText()函数的具体实现如下：

```
void
CTestfontView::WriteText(CDC * pDC,int nX,int nY,int nSize,int nAngle)
{
    CFont * pOldFont;
    CFont * pNewFont=new CFont;
```

```
    pNewFont->CreateFont(nSize,0,nAngle,0,400,FALSE,FALSE,
                         0,ANSI_CHARSET,OUT_DEFAULT_PRECIS,
                         CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
                         DEFAULT_PITCH&FF_SWISS, "Aerial");
        pOldFont=(CFont *)pDC->SelectObject(pNewFont);
        pDC->TextOut(nX,nY,"欢迎使用 VC++ !");
        pDC->SelectObject(pOldFont);
        delete pNewFont;
}
```

步骤 4：在 testfontView.cpp 文件的 OnDraw()函数中添加代码如下：

```
void
CTestfontView::OnDraw(CDC * pDC)
{
      CTestfontDoc * pDoc = GetDocument();
      ASSERT_VALID(pDoc);
      // TODO: add draw code for native data here
      int i;
      CRect rect;
GetClientRect(&rect);
CBrush * pOldBrush=(CBrush *)pDC->SelectStockObject(WHITE_BRUSH);
pDC->Rectangle(rect);
pDC->SelectObject(pOldBrush);
pDC->SetViewportOrg(250,100);
pDC->SetTextAlign(TA_CENTER);
for(i=0;i<5;i++)
{
    WriteText(pDC,0,30*i-50,16+4*i,0);
}
return;
//my code end here
}
```

步骤 5：编译并运行程序。本程序运行的图形界面如图 7.39 所示。

题 7.3.5 参考答案：

解：在本实例中需要设计一个 CRectangle 类，使用集合类来保存所画的多个矩形对象，在视图中画图，重载序列化函数 Serialize()。

步骤 1：新建项目 polygon，在 AppWizard 的第一步选择 Single document。在第四步中有一个"Advanced"按钮，单击该按钮，弹出"Advanced Option"对话框，其中的 File Extension 表示该应用程序创建的文档所用的文件扩展名，在这里输入 rect。表示该程序创建的文档的后缀为 rect。

步骤 2：编写源代码

(1) 创建 Rectangle.h 头文件。

新建一个头文件，取名为 Rectangle.h，将其添加到项目中去。实现源代码如下：

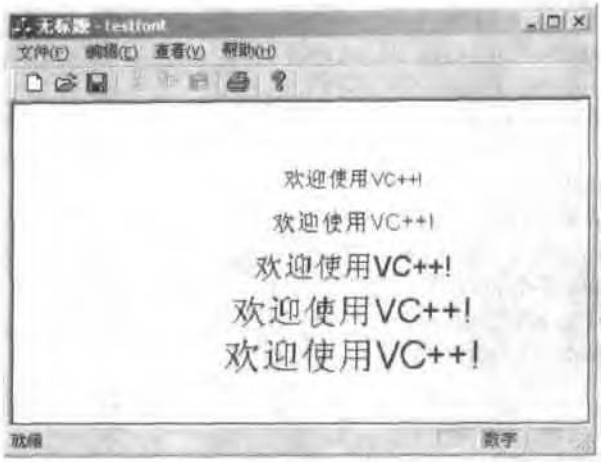

图 7.39 题 7.3.4 运行结果

```
//Rectangle.h
#ifndef RECTANGLE_H
#define RECTANGLE_H

#define cMaxFrameWidth 100
#define cMinFrameWidth 1

class CRectangle:public CObject{
private:
        int m_nLeft;
        int m_nTop;
        int m_nRight;
        int m_nBottom;
        COLORREF m_FrameColor;          //矩形的边框颜色
        COLORREF m_InnerColor;          //矩形的内部颜色
        int m_nFrameWidth;              //边框宽度
        BOOL m_bOnlyFrame;              //内部是否填充
        BOOL m_bFrameOnTop;             //是否先填充后画边框

public:
        CRectangle(CRect * pRect,COLORREF frameColor,COLORREF innerColor,
        int frameWidth=1,BOOL onlyFrame=TRUE,BOOL frameOnTop=TRUE);
        CRectangle();
        ~CRectangle();
        void SetRect(int nLeft,int nTop,int nRight,int nBottom);
        void SetRect(CRect * pRect);
        void GetRect(CRect * pRect);
        void SetFrameColor(int nR,int nG,int nB);
        void SetFramecolor(COLORREF color);
```

```
        COLORREF GetFrameColor();
        void SetInnerColor(int nR,int nG,int nB);
        void SetInnerColor(COLORREF color);
        COLORREF GetInnerColor();
        void SetFrameWidth(int nWidth);
        int GetFrameWidth();
        BOOL IsHasInner();
        void SetInner(BOOL bHasInner);
        BOOL IsFrameOnTop();
        void FrameTop(BOOL bTop);
        void Draw(CDC * pDC);

protected:
        virtual void Serialize(CArchive &ar);
        DECLARE_SERIAL(CRectangle)
};
#endif
```

(2) 编写类的实现文件 Rectangle.cpp,实现代码如下:

```
//Rectangle.cpp
#include "stdafx.h"
#include "polygon.h"
#include "Rectangle.h"

IMPLEMENT_SERIAL(CRectangle,CObject,1)
CRectangle::Crectangle
        (CRect * pRect,COLORREF frameColor,COLORREF innerColor,
         int frameWidth,BOOL onlyFrame,BOOL frameOnTop)
{
        m_nLeft=min(pRect->left,pRect->right);
        m_nRight=max(pRect->left,pRect->right);
        m_nTop=min(pRect->top,pRect->bottom);
        m_nBottom=max(pRect->top,pRect->bottom);
        m_FrameColor=frameColor;
        m_InnerColor=innerColor;
        m_nFrameWidth=frameWidth;
        m_bOnlyFrame=onlyFrame;
        m_bFrameOnTop=frameOnTop;
}

CRectangle::CRectangle()
{

}
```

```
CRectangle::~CRectangle()
{

}

void
CRectangle::SetRect(int nLeft,int nTop,int nRight,int nBottom)
{
        m_nLeft = min(nLeft,nRight);
        m_nRight = max(nLeft,nRight);
        m_nTop = min(nTop,nBottom);
        m_nBottom = max(nTop,nBottom);
}

void
CRectangle::SetRect(CRect * pRect)
{
        m_nLeft = min(pRect->left,pRect->right);
        m_nRight = max(pRect->left,pRect->right);
        m_nTop = min(pRect->top,pRect->bottom);
        m_nBottom = max(pRect->top,pRect->bottom);
}

void
CRectangle::GetRect(CRect * pRect)
{
        pRect->left = m_nLeft;
        pRect->right = m_nRight;
        pRect->top = m_nTop;
        pRect->bottom = m_nBottom;
}

void
CRectangle::SetFrameColor(int nR,int nG,int nB)
{
        if(nR>255)nR=255;
        if(nR<0)nR=0;
        if(nG>255)nG=255;
        if(nG<0)nG=0;
        if(nB>255)nB=255;
        if(nB<0)nB=0;
    m_FrameColor=RGB(nR,nG,nB);
}
```

```
COLORREF
CRectangle::GetFrameColor()
{
        return m_FrameColor;
}

void
CRectangle::SetInnerColor(int nR,int nG,int nB)
{
        if(nR>255)nR=255;
        if(nR<0)nR=0;
        if(nG>255)nG=255;
        if(nG<0)nG=0;
        if(nB>255)nB=255;
        if(nB<0)nB=0;
        m_InnerColor=RGB(nR,nG,nB);
}

void
CRectangle::SetInnerColor(COLORREF color)
{
        m_InnerColor=color;
}

COLORREF
CRectangle::GetInnerColor()
{
        return m_InnerColor;
}

void
CRectangle::SetFrameWidth(int nWidth)
{
        if(nWidth>cMaxFrameWidth)
            nWidth=cMaxFrameWidth;
        if(nWidth<cMinFrameWidth)
            nWidth=cMinFrameWidth;
        m_nFrameWidth=nWidth;
}

int
CRectangle::GetFrameWidth()
{
        return m_nFrameWidth;
}
```

```
BOOL
CRectangle::IsHasInner()
{
        return (! m_bOnlyFrame);
}

void
CRectangle::SetInner(BOOL bHasInner)
{
        m_bOnlyFrame=! bHasInner;
}

BOOL
CRectangle::IsFrameOnTop()
{
        return m_bFrameOnTop;
}

void
CRectangle::FrameTop(BOOL bTop)
{
        m_bFrameOnTop=bTop;
}

void
CRectangle::Serialize(CArchive &ar)   // 重载序列化函数,将矩形对象的参数存盘
{
        if(ar.IsStoring()){
            ar<<m_nLeft<<m_nTop<<m_nRight<<m_nBottom;
            ar<<m_nFrameWidth;
            ar<<m_FrameColor<<m_InnerColor;
            ar<<m_bOnlyFrame<<m_bFrameOnTop;
        }
        else{
            ar>>m_nLeft>>m_nTop>>m_nRight>>m_nBottom;
            ar>>m_nFrameWidth;
            ar>>m_FrameColor>>m_InnerColor;
            ar>>m_bOnlyFrame>>m_bFrameOnTop;
        }
}

void
CRectangle::Draw(CDC * pDC)
{
        CPen rectPen(PS_SOLID,m_nFrameWidth,m_FrameColor);
```

```
        CPen * pOldPen;
        CBrush innerBrush(m_InnerColor);
        CRect rect(m_nLeft,m_nTop,m_nRight,m_nBottom);
        pOldPen=pDC->SelectObject(&rectPen);
        if(m_bOnlyFrame){
            pDC->MoveTo(m_nLeft,m_nTop);
            pDC->LineTo(m_nRight,m_nTop);
            pDC->LineTo(m_nRight,m_nBottom);
            pDC->LineTo(m_nLeft,m_nBottom);
        pDC->LineTo(m_nLeft,m_nTop);
    }
    else{
        if(m_bFrameOnTop){
            pDC->FillRect(&rect,&innerBrush);
            pDC->MoveTo(m_nLeft,m_nTop);
            pDC->LineTo(m_nRight,m_nTop);
            pDC->LineTo(m_nRight,m_nBottom);
            pDC->LineTo(m_nLeft,m_nBottom);
            pDC->LineTo(m_nLeft,m_nTop);
        }
        else{
            pDC->FillRect(&rect,&innerBrush);
            pDC->MoveTo(m_nLeft,m_nTop);
            pDC->LineTo(m_nRight,m_nTop);
            pDC->LineTo(m_nRight,m_nBottom);
            pDC->LineTo(m_nLeft,m_nBottom);
            pDC->LineTo(m_nLeft,m_nTop);
        }
    }
    pDC->SelectObject(pOldPen);
}
```

(3) 在 polygonDoc.h 头文件中添加如下代码：

```
private:
    CObList m_RectangleList;
public:
    CObList * GetRectList();
protected:
    virtual void DeleteContents();
```

在 polygonDoc.cpp 中添加以上两个函数的实现代码如下：

```
CObList * CPolygonDoc::GetRectList()
{
    return (&m_RectangleList);
}
```

```
void
CPolygonDoc::DeleteContents()
{
#ifdef _DEBUG
Dump(afxDump);
#endif
  while(m_RectangleList.GetHeadPosition())
  {
      delete m_RectangleList.RemoveHead();
  }
}
```

其中，GetRectList()能从文档中获取指向链表数据的指针，通过遍历链表就可以访问各矩形。DelectContents()是Cdocument类的一个虚函数，它能删除文档的内容，当文档被关闭时，应用程序会自动调用该函数。

(4) 在polygonView.cpp中添加头文件rectangle.h，重载视图类的OnDraw()函数如下：

```
void
CPolygonView::OnDraw(CDC* pDC)
{
      CPolygonDoc* pDoc = GetDocument();
      ASSERT_VALID(pDoc);
      // TODO: add draw code for native data here
      //my code start here
      CObList *pList=pDoc->GetRectList();          //得到指向链表的指针
      POSITION pos=pList->GetHeadPosition();       //指向链表的第一个元素
      while(pos!=NULL){
         CRectangle *pRectangle=(CRectangle *)(pList->GetNext(pos));
      pRectangle->Draw(pDC);
  }
  //my code end here
}
```

(5) 为了能使程序随着鼠标的拖动画出矩形，需在视图类中添加能够响应鼠标消息的函数。用ClassWizard为视图类的WM_LBUTTONDOWN、WM_MOUSEMOVE、WM_LBUTTONUP加入消息处理函数OnLButtonDown()、OnMouseMove()、OnLButtonUp()。程序代码如下：

```
CRect m_Rect;  //全局变量
void
CPolygonView::OnLButtonDown(UINT nFlags, CPoint point)
{
            // TODO: Add your message handler code here and/or call default
            //my code start here
     SetCapture();
```

```
        m_Rect.top=point.y;
        m_Rect.left=point.x;
        m_Rect.bottom=point.y;
        m_Rect.right=point.x;
        //my code end here

    CView::OnLButtonDown(nFlags, point);
}

void
CPolygonView::OnMouseMove(UINT nFlags, CPoint point)
{
        // TODO: Add your message handler code here and/or call default
        //my code start here
        if((nFlags&MK_LBUTTON)==MK_LBUTTON){
        if(GetCapture()!=this) return;
        CClientDC dc(this);
        CPen newPen(PS_DASH,1,RGB(0,0,255));
        CPen *pOldPen=dc.SelectObject(&newPen);
        int nOldMode=dc.SetROP2(R2_NOTXORPEN);      //设置绘图模式为反相画
        dc.MoveTo(m_Rect.left,m_Rect.top);
        dc.LineTo(m_Rect.right,m_Rect.top);
        dc.LineTo(m_Rect.right,m_Rect.bottom);
        dc.LineTo(m_Rect.left,m_Rect.bottom);
        dc.LineTo(m_Rect.left,m_Rect.top);                  //擦除前次所画的矩形

        m_Rect.right=point.x;
        m_Rect.bottom=point.y;

        dc.MoveTo(m_Rect.left,m_Rect.top);
        dc.LineTo(m_Rect.right,m_Rect.top);
        dc.LineTo(m_Rect.right,m_Rect.bottom);
        dc.LineTo(m_Rect.left,m_Rect.bottom);
        dc.LineTo(m_Rect.left,m_Rect.top);                  //在当前鼠标位置画矩形

        dc.SelectObject(pOldPen);
        dc.SetROP2(nOldMode);
    }
    //my code end here

    CView::OnMouseMove(nFlags, point);
}

void CPolygonView::OnLButtonUp(UINT nFlags, CPoint point)
{
```

```
    // TODO: Add your message handler code here and/or call default
    //my code start here
    if(GetCapture()! =this) return;
    CClientDC dc(this);
    CPen newPen(PS_DASH,1,RGB(0,0,255));
    CPen * pOldPen=dc.SelectObject(&newPen);
    int nOldMode=dc.SetROP2(R2_XORPEN);
    dc.MoveTo(m_Rect.left,m_Rect.top);
    dc.LineTo(m_Rect.right,m_Rect.top);
    dc.LineTo(m_Rect.right,m_Rect.bottom);
    dc.LineTo(m_Rect.left,m_Rect.bottom);
    dc.LineTo(m_Rect.left,m_Rect.top);  //在当前鼠标位置画矩形

    dc.SelectObject(pOldPen);
    dc.SetROP2(nOldMode);

    CPen framePen(PS_SOLID,1,RGB(0,0,255));
    pOldPen=dc.SelectObject(&framePen);
    m_Rect.right=point.x;
    m_Rect.bottom=point.y;

    dc.MoveTo(m_Rect.left,m_Rect.top);
    dc.LineTo(m_Rect.right,m_Rect.top);
    dc.LineTo(m_Rect.right,m_Rect.bottom);
    dc.LineTo(m_Rect.left,m_Rect.bottom);
    dc.LineTo(m_Rect.left,m_Rect.top);
    dc.SelectObject(pOldPen);

    if(m_Rect.left! =m_Rect.right&&m_Rect.top! =m_Rect.bottom){
      CRectangle * pRectangle=new CRectangle(&m_Rect,
          RGB(0,0,255),RGB(255,255,255),1,TRUE,TRUE);
      CPolygonDoc * pDoc=GetDocument();
      CObList * pList=pDoc->GetRectList();
      pList->AddTail(pRectangle);     //将最新矩形添加至链表
      pDoc->UpdateAllViews(this,NULL,NULL);
      pDoc->SetModifiedFlag(TRUE); //关闭程序时提醒用户保存

      ReleaseCapture();
    }
    //my code end here

    CView::OnLButtonUp(nFlags, point);
}
```

步骤3：编译、运行源程序后在窗口中按住鼠标左键，移动鼠标即可画出矩形。单击“关闭”按钮时，程序会提醒是否要保存文档。程序运行窗口如图7.10所示。

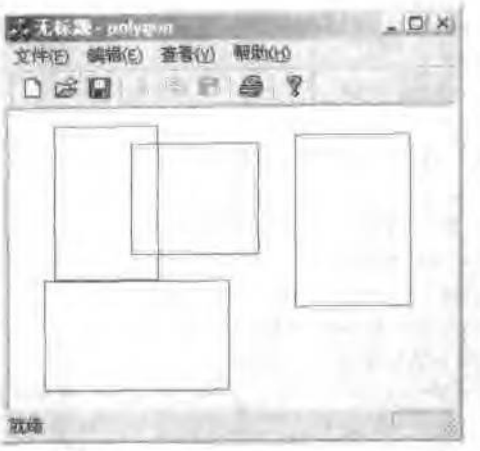

图 7.40 题 7.3.5 运行结果

题 7.3.6 参考答案：

解：本实例是使用 Visual C++ 的 Pop-Up Menu 组件来生成快捷菜单的。

步骤 1：用 AppWizard 创建一个工程 popmenu。

步骤 2：用 Pop-Up Menu 组件添加快捷菜单。

从 VC++ 主菜单中选择"Project\Add To Project\Components and Controls..."菜单项，弹出"Components and Controls Gallery"对话框。双击"Visual C++ Components"文件夹，在双击"Pop-Up Menu"组件，弹出 Pop-Up Menu 对话框，在 Add Pop-up menu to:后的下拉列表中选择 CPopmenuView 类，单击 OK 按钮，回到"Components and Controls Gallery"对话框后，再单击"OK"按钮，结束添加快捷菜单。此时，"Pop-up Menu"组件已在 CPopmenuView 类中添加了 void CPopmenuView::OnContextMenu(CWnd *, CPoint point)函数，程序如下：

```
void
CPopmenuView::OnContextMenu(CWnd *, CPoint point)
{

  // CG: This block was added by the Pop-up Menu component
  {
      if (point.x == -1 && point.y == -1){
          //keystroke invocation
          CRect rect;
          GetClientRect(rect);
          ClientToScreen(rect);

          point = rect.TopLeft();
          point.Offset(5, 5);
      }
```

```
    CMenu menu;
    VERIFY(menu.LoadMenu(CG_IDR_POPUP_POPMENU_VIEW));

    CMenu * pPopup = menu.GetSubMenu(0);
    ASSERT(pPopup != NULL);
    CWnd * pWndPopupOwner = this;

    while (pWndPopupOwner->GetStyle() & WS_CHILD)
        pWndPopupOwner = pWndPopupOwner->GetParent();

    pPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON,
                           point.x, point.y,pWndPopupOwner);
  }
}
```

步骤 3：为快捷菜单添加菜单项。

在“资源编辑器”中打开“MENU”资源文件夹，双击“CG_IDR_POPUP_POPMENU_VIEW”菜单资源，打开“菜单编辑器”。添加“点我”菜单项，ID 设置为 ID_APP_POPMENU。运行程序后在视图区右击弹出快捷菜单。

步骤 4：为菜单项添加代码。

在“点我”菜单项单击右键，选择建立类向导，弹出“Adding a class”对话框，选择 Select an existing class 单击“OK”。在弹出的“select class”对话框中选择“CPopmenuView”类。在“Object Ids”中选择“ID_APP_POPMENU”，在 messages 后双击“COMMAND”，单击“edit code”按钮，添加如下代码：

```
void
CPopmenuView::OnAppPopmenu()
{
    // TODO: Add your command handler code here
    //my code start here
    AfxMessageBox("这是一个快捷菜单的例子!");
    //my code end here

}
```

这样，在快捷菜单中选择“点我”时会弹出一个消息框，如图 7.41 所示。

题 7.3.7 参考答案：

解：本实例是状态栏应用的一个实例。

步骤 1：用 AppWizard 创建应用程序 StatusBar。

步骤 2：利用 ClassWizard 在 CStatusbarView 中添加鼠标消息的映射函数。

打开“MFC ClassWizard”对话框，在 Class name 下拉列表框中选择“CstatusbarView”类，在 Object

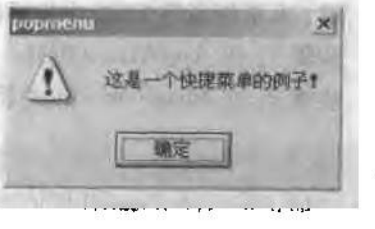

图 7.41 题 7.3.6 运行结果

IDs 下选择“CstatusbarView”，在 messages 下双击“WM_MOUSEMOVE”，单击“Edit Code”按钮，添加如下代码：

```
void
CStatusbarView::OnMouseMove(UINT nFlags, CPoint point)
{
  // TODO: Add your message handler code here and/or call default
  // my code start here
  //获得状态栏的指针
  CstatusBar * pStatus=(CStatusBar *)AfxGetApp()->m_pMainWnd->
          GetDescendantWindow(ID_VIEW_STATUS_BAR);
  if(pStatus)
  {
      char tbuf[40];
      sprintf(tbuf,"(%3d,%3d)",point.x,point.y);
      //在状态栏的第一个窗格中输出当前鼠标的位置
      pStatus->SetPaneText(0,tbuf);
  }

  CView::OnMouseMove(nFlags, point);
}
```

步骤 3：编译并运行程序，程序界面如图 7.42 所示，在下角的数字即为鼠标的当前位置。

图 7.42　图 7.3.7 运行结果

题 7.3.8 参考答案：

解：本实例利用 MFC 中提供的函数 SetButtonText()来加入按钮文字。

步骤 1：用 AppWizard 创建一个单文档界面的应用程序。

步骤 2：在 MainFrm.cpp 中将 OnCreate()函数修改如下：

```
int
CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
```

```
if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
    return -1;

if (! m_wndToolBar.CreateEx
   (this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP|
   CBRS_GRIPPER| CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC)
                || ! m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
{
    TRACE0("Failed to create toolbar\n");
    return -1;      // fail to create
}

if (! m_wndStatusBar.Create(this) ||
    ! m_wndStatusBar.SetIndicators(indicators,
     sizeof(indicators)/sizeof(UINT)))
{
     TRACE0("Failed to create status bar\n");
     return -1;      // fail to create
}
//my code start here
for(int i=0;i<m_wndToolBar.GetCount();i++)
{
    UINT id=m_wndToolBar.GetItemID(i);
    CString str;
    if(! str.LoadString(id)) //将与按钮"ID"相同的字符串资源调入
    continue;                //如果没有相对应的字符串,则跳过
    int j=str.Find("\n");
    //在串中寻找\n分隔符,其前是状态条上显示的文字,后是工具提示文字
    if(j<0)
        continue;
    //没找到则跳过
    str=str.Right(str.GetLength()-j-1);
     //取出分隔符右边的字符串
    m_wndToolBar.SetButtonText(i,str);
    //使用此字符串作为按钮文字
}
CRect rect;
m_wndToolBar.GetItemRect(0,&rect);
     //取第一个按钮的大小
m_wndToolBar.SetSizes(rect.Size(),CSize(16,15));
     //将所有的按钮都设为与第一个按钮一样大
//my code end here
// TODO: Delete these three lines if you don't want the toolbar to
// be dockable
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
```

```
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}
```

步骤 3：编译并运行程序。修改代码前的程序界面如图 7.43 所示。

修改代码后的程序界面如图 7.44 所示。

图 7.43 修改前的程序界面

图 7.44 修改后的程序界面

题 7.3.9 参考答案：

解：本题的关键是使用 CWnd 的成员函数 SetWindowPos()。扩展时用它将对话框的宽度和高度设为扩展后的大小，收缩时再设为原大小。

步骤 1：用 AppWizard(exe)生成一个名为“expdlg”的对话框，添加一个“扩展”按钮，在常用部分控件和扩展部分控件之间放置一个去掉文字的静态文本控件。

步骤 2：为“扩展”按钮添加处理单击消息的函数 OnExpand()，代码如下：

```
void
CExpdlgDlg::OnExpand()
{
    // TODO: Add your control notification handler code here
    //my code start here
    CString text;

    CWnd * pButton=GetDlgItem(IDC_EXPAND);
    ASSERT(pButton);
    pButton->GetWindowText(text);

    //用按钮文本判断进行扩展或收缩
    if(text=="扩展 &E>>")
    {
        ExpandDlg(TRUE);
        SetDlgItemText(IDC_EXPAND,"收缩 &S<<");
    }
```

```
        else
        {
            ExpandDlg(FALSE);
            SetDlgItemText(IDC_EXPAND,"扩展 &E>>");
        }
}
```

步骤 3：在 CexpdlgDlg 类中添加函数 ExpandDlg(BOOL bExpand)，当参数为 TRUE 时，扩展对话框，为 FALSE 时，收缩对话框。函数代码如下：

```
void CExpdlgDlg::ExpandDlg(BOOL bExpand)
{
        int cx,cy;
        CRect rectDlg;

        GetWindowRect(&rectDlg);
        cx=rectDlg.Width();
        cy=rectDlg.Height();        //对话框当前状态的高度

        //改变对话框的高度
        SetWindowPos(NULL,0,0,
            cx,bExpand? cy+m_nExpansion:cy-m_nExpansion,
            SWP_NOMOVE|SWP_NOZORDER);
        //处理控件的有效性
}
```

步骤 4：在 CexpdlgDlg 类中添加成员变量 m_nExpansion，并在构造函数 CexpdlgDlg()中将它初始化为零。代码如下：

```
CExpdlgDlg::CExpdlgDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CExpdlgDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CExpdlgDlg)
        // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
        //my code start here
        m_nExpansion=0;
        //my code end here
}
```

在 OnInitDialog()中添加代码如下：

```
BOOL
CExpdlgDlg::OnInitDialog()
{
```

```
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu * pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (! strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu
                    (MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);        // Set big icon
    SetIcon(m_hIcon, FALSE);       // Set small icon

    // TODO: Add extra initialization here
    //my code start here
    CRect rectDlg,rectMark;
    GetWindowRect(&rectDlg);
    CWnd * pMark=GetDlgItem(IDC_MARK);
    ASSERT(pMark);
    pMark->GetWindowRect(&rectMark);
    m_nExpansion=rectDlg.bottom-rectMark.top;
    ExpandDlg(FALSE);
    //my code end here
    return TRUE; // return TRUE unless you set the focus to a control
}
```

步骤 5：编译并运行程序。

题 7.3.10 参考答案：

解：本实例是工具栏应用的一个实例。

步骤 1：运用 AppWizard 创建一个单文档应用程序，视图类的基类选择 CView。

步骤 2：利用资源编辑器创建工具条，它包括"A"、"B"、"C" 3 个按钮，如图 7.45 所示。

步骤 3：利用"Toolbar Button Properties"对话框给按钮"A"、"B"、"C"设置 ID 值，ID 值依次是：ID_BUTTON_A、ID_BUTTON_B、ID_BUTTON_C，如图 7.46 所示。

步骤 4：在 CmainFrame 主框架类中声明一个 CtoolBar 类型的成员变量 m_testToolBar。

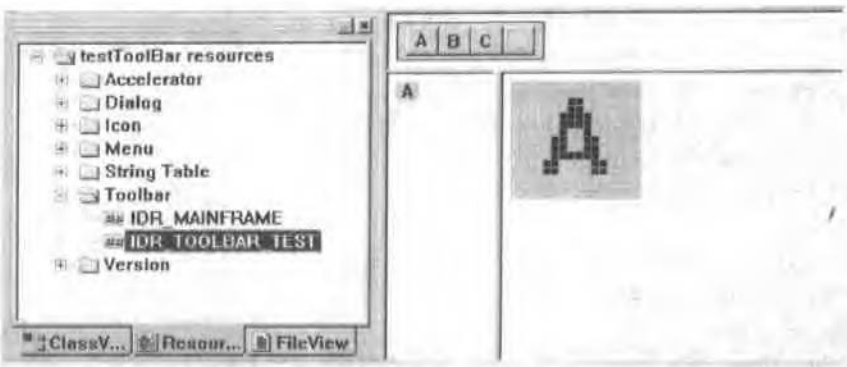

图 7.45　创建工具条资源编辑器

图 7.46　设置 ID 值对话框

步骤 5：运行 ClassWizard，给"A"、"B"、"C"按钮在 CmainFrame 主框架类中添加消息映射，且在消息映射函数中添加代码：

```
void CMainFrame::OnButtonA()
{
    // TODO: Add your command handler code here
    CDC * pDC=GetDC();
    CFont font;
    font.CreateFont(-24,                    //字体字符的逻辑高度
                    0,                      //字符平均宽度取默认值
                    0,                      //文本行角度为 0,水平
                    0,                      //字符角度为 0,正立
                    FW_NORMAL,              //正常字体
                    true,                   //倾斜
                    FALSE,                  //不加下划线
                    FALSE,                  //不加删除线
                    ANSI_CHARSET,
                    OUT_DEFAULT_PRECIS,
                    CLIP_DEFAULT_PRECIS,
                    DEFAULT_QUALITY,
```

```
                DEFAULT_PITCH|FF_MODERN,
                "Times New Roman");
        CFont *oldFont=pDC->SelectObject (&font);
        pDC->TextOut(20,80,"Visual C++ 6.0 编程实践");
        pDC->SelectObject(oldFont);
void CMainFrame::OnButtonB()
{
        // TODO: Add your command handler code here
        CDC* pDC=GetDC();
        CFont font;
        font.CreateFont(-30,                    //字体字符的逻辑高度
                0,                              //字符平均宽度取默认值
                0,                              //文本行角度为0,水平
                0,                              //字符角度为0,正立
                FW_NORMAL,                      //正常字体
                true,                           //倾斜
                FALSE,                          //不加下划线
                FALSE,                          //不加删除线
                ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS,
                DEFAULT_QUALITY,
                DEFAULT_PITCH|FF_MODERN,
                "Times New Roman");
        CFont *oldFont=pDC->SelectObject (&font);
        pDC->TextOut(20,120,"Visual C++ 6.0 编程实践");
        pDC->SelectObject(oldFont);
}
void CMainFrame::OnButtonC()
{
        // TODO: Add your command handler code here
        CDC* pDC=GetDC();
        CFont font;
        font.CreateFont(-40,                    //字体字符的逻辑高度
                0,                              //字符平均宽度取默认值
                0,                              //文本行角度为0,水平
                0,                              //字符角度为0,正立
                FW_NORMAL,                      //正常字体
                true,                           //倾斜
                FALSE,                          //不加下划线
                FALSE,                          //不加删除线
                ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS,
                DEFAULT_QUALITY,
```

```
            DEFAULT_PITCH|FF_MODERN,
            "Times New Roman");
        CFont * oldFont=pDC->SelectObject (&font);
        pDC->TextOut(20,160,"Visual C++ 6.0 编程实践");
        pDC->SelectObject(oldFont);
    }
```

步骤 6：编译并运行程序，分别点击活动工具条上的按钮“A”、“B”、“C”后，程序运行窗口如图 7.47 所示。

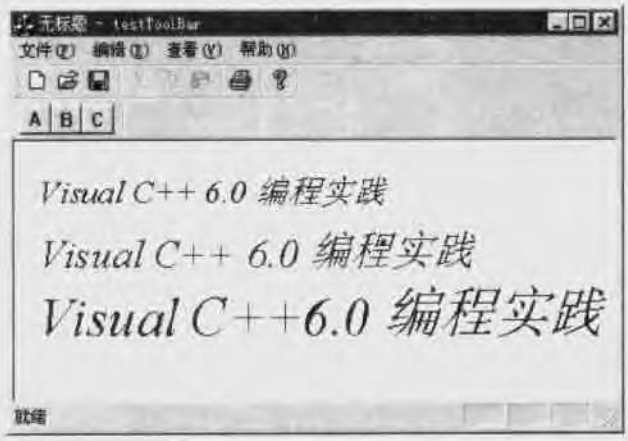

图 7.47 题 7.3.10 运行结果