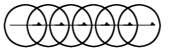
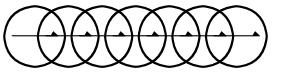


C++ 

吕凤翥　编著

電子工業出版社
Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是《C++ 语言程序设计》的学习辅导书。主要介绍了 Visual C++ 6.0 编译系统的使用，在此系统基础上进行 C++ 语言的程序调试上机指导。本书着眼于教材中的重点难点知识分析以及习题解析，并为初学者指出了在 C++ 语言学习过程中需要注意的问题。

本书适合高等理工院校学生在学习 C++语言时作为参考，以帮助自学。

图书在版编目（CIP）数据

C++语言程序设计——上机指导与习题解答/吕凤翥编著.－北京：电子工业出版社，2001.9□

高等学校教材□

ISBN 7-5053-0000-0

Ⅰ.C… Ⅱ.吕… Ⅲ.①程序设计②计算机专业－教材 Ⅳ.①TN000.00②TN000

中国版本图书馆 CIP 数据核字（2001）第 000000 号

丛 书 名：高等学校教材□

书　　名：C++语言程序设计——上机指导与习题解答□

编　　著：吕凤翥

责任编辑：束传政□

特约编辑：赫　嘉

排版制作：电子工业出版社计算机排版室□

印 刷 者：□

装 订 者：□

出版发行：电子工业出版社 URL:http://www.phei.com.cn □

北京市海淀区万寿路 173 信箱邮编 100036 □

经　　销：各地新华书店□

开　　本：787×1 092　1/16　印张：9.25　字数：236.8 千字□

版　　次：2001 年 9 月第 1 版　2001 年 9 月第 1 次印刷

书　　号：ISBN 7-5053-7029-4 / TP·4030

印　　数：0 000 册　　定价：00.00 元□

内 容 简 介

本书是《C++ 语言程序设计》的学习辅导书。主要介绍了 Visual C++ 6.0 编译系统的使用，在此系统基础上进行 C++ 语言的程序调试上机指导。本书着眼于教材中的重点难点知识分析以及习题解析，并为初学者指出了在 C++ 语言学习过程中需要注意的问题。

本书适合高等理工院校学生在学习 C++ 语言时作为参考，以帮助自学。

图书在版编目（CIP）数据

C++ 语言程序设计：上机指导与习题解答/吕凤翥编著. - 北京：电子工业出版社，2001.9
高等学校教材
ISBN 7-5053-7029-4

Ⅰ.C… Ⅱ.吕… Ⅲ.C 语言 - 程序设计 - 高等学校 - 自学参考资料 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2001）第 067146 号

丛 书 名：高等学校教材
书　　名：C++ 语言程序设计——上机指导与习题解答
编 著 者：吕凤翥
责任编辑：束传政
特约编辑：赫　嘉
排版制作：电子工业出版社计算机排版室
印 刷 者：
装 订 者：
出版发行：电子工业出版社　URL：http://www.phei.com.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销：各地新华书店
开　　本：787×1 092　1/16　印张：9.25　字数：236.8 千字
版　　次：2001 年 9 月第 1 版　2001 年 9 月第 1 次印刷
书　　号：ISBN 7-5053-7029-4 / TP·4030
印　　数：8 000 册　　定价：12.00 元

凡购买电子工业出版社的图书，如有缺页、倒页、脱页、所附磁盘或光盘有问题者，请向购买书店调换；
若书店售缺，请与本社发行部联系调换。电话　68279077

目　　录

第 1 章　C++ 程序的实现

1.1　Visual C++ 6.0 编译系统部分功能介绍

主窗口

启动 Visual C++ 6.0 **进入** Developer Studio **编译环境，如图** 1.1 **所示。**

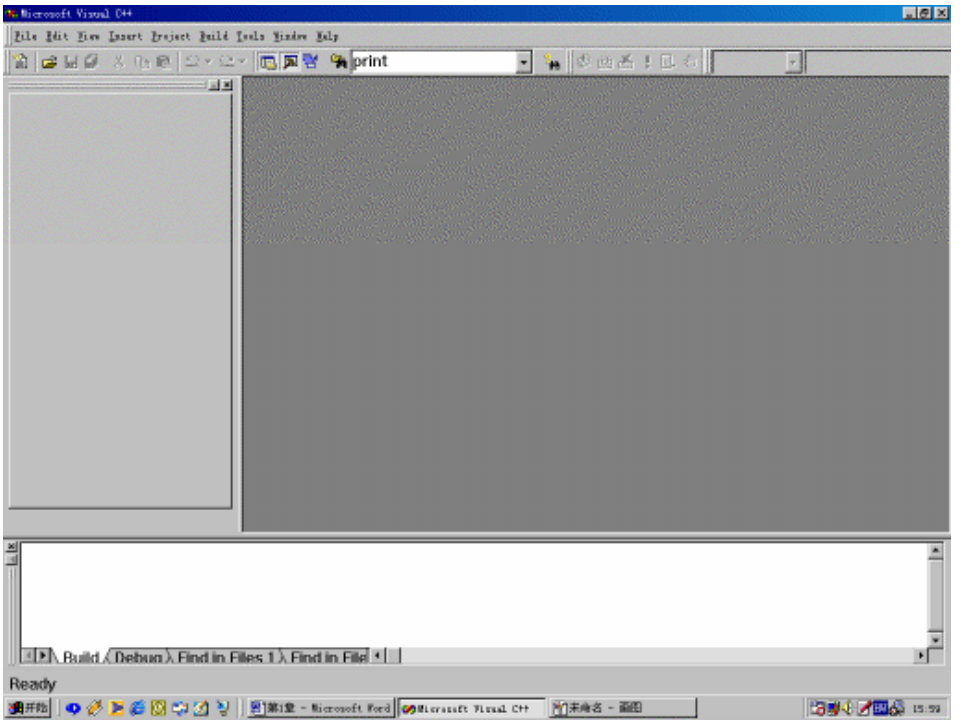


图 1.1　Visual C++ 6.0 **编译环境窗口**

主窗口由标题栏、菜单栏、工具栏、工作区窗口、源代码编辑窗口、输出窗口和状态栏组成。

屏幕窗口中最上方是标题栏，显示所打开的应用程序名。标题栏左端是控制菜单图标，单击后弹出窗口控制菜单。栏题栏右端从左至右有三个控制按钮，分别为最小化、最大化(还原)和关闭按钮，可用它们快速设置窗口的大小。

标题栏下方是菜单栏，由 9 **个菜单项组成。单击菜单项弹出下拉式菜单，可使用这些菜单项实现集成环境的各种功能。**

菜单栏下方是工具栏，它由若干个功能按钮组成，单击按钮可实现某种操作功能。

工具栏的下方有左右两个窗口，左窗口是项目工作区窗口，右窗口是源代码编辑窗口。

在项目工作窗口和源代码编辑窗口的下方有一个输出窗口，在创建项目(Build)时，用来显示项目创建过程中的错误信息。

屏幕最底部是状态栏，它可给出当前操作或所选命令的提示信息。

在上述窗口组成部分中，工作区窗口可通过单击工具栏中“Workspace”按钮隐藏或显示；输出窗口可通过单击工具栏中“Output”按钮隐藏或显示。隐藏这些窗口可以扩大源代码编辑区的大小。

工具栏

为了用户操作方便，该系统在主窗口中提供了多种工具栏，每种工具栏中有若干个按钮，每个工具栏中的按钮表示某种操作。在鼠标指向某个按钮时，将显示出该按钮的功能信息提示。

主窗口在默认情况下只显示两个常用的工具栏：Standard 和 Build 工具栏。如果要使用其他工具栏，可做下述操作：

鼠标指向工具栏的位置，单击右键，出现如图 1.2 所示的快捷菜单。该菜单中，提供了 10 个工具栏供选择使用。单击要选用的工具栏后，在该工具栏名前出现符号√，且该工具栏将出现在主窗口中。如果不想用某个工具栏，也可在工具栏位置单击右键，在出现的快捷菜单中选择不想用的某工具栏，该工具栏名前符号√消失，该工具栏将从主菜单中消失。

图 1.2　工具栏快捷菜单

下面介绍几个与实现 C++语言源程序相关的工具栏所含工具项的功能和用法，其余的工具栏不再介绍。

一、Standard 工具栏

该工具栏中共有 15 个工具项按钮。如图 1.3 所示。

图 1.3　Standard 工具栏

自左至右各按钮的功能介绍如下：

- New Text File　创建新的文本文件
- Open　打开已有文档
- Save　保存当前文档内容
- Save All　保存所有打开的文档
- Cut　将选定的文档内容从文档中删除，并将之复制到剪贴板中
- Copy　将选定的文档内容复制到剪贴板中
- Paste　在当前插入点处粘贴剪贴板中的内容
- Undo　取消最近一次编辑操作

・Redo　恢复前一次取消的编辑操作
・Workspace　显示或隐藏工作区窗口
・Output　显示或隐藏输出窗口
・Windows list　管理当前打开的窗口
・Find in Files　在多个文件中搜索字符串
・Find　激活查找工具
・Search　搜索联机文档

二、Build **工具栏**

该工具栏中有 8 **个工具项按钮，如图** 1.4 **所示。**

图 1.4　Build **工具栏**

这些工具项按钮用来对已建好的应用文件或项目进行编译、连接和运行。自左至右，各工具项按钮的功能介绍如下：

・Select Active Project　**选择当前活动项目**
・Select Active Configuration　**选择活动的配置**

Visual C++**提供两种活动配置：**Win32 Release **和** Win32 Debug**。前者是基于** Win32 **平台的发行版，后者是基于** Win32 **平台的调试版。**

・Compile　**编译文件**
・Build　**创建项目**
・Stop Build　**停止创建项目**
・Execute Program　**执行程序**
・Go　**启动或继续程序的执行**
・Insert/Remove Breakpoint　**插入或删除断点**

三、Build MiniBar **工具栏**

图 1.5　Build MiniBar **工具栏**

该工具栏中包含 6 **个工具项按钮，如图** 1.5 **所示。该工具栏** 6 **个工具项按钮依次为：**

・Compile
・Build
・Stop Build
・Execute program
・Go
・Insert/Remove Breakpoint

它们的功能与工具栏 Build **中的相应按钮功能相同。**

四、Debug **工具栏**

该工具栏中含有 16 **个工具项按钮，如图** 1.6 **所示。**

这些工具项按钮可用来调试已编译的 C++源文件及项目，查找所存在的问题。它们只有处于调试运行状态时才有效。

图 1.6　Debug 工具栏

第一行自左至右，各个工具项按钮的功能为：

- Restart　重新启动程序，并处于调试状态
- Stop Debugging　停止调试运行的程序
- Break Execution　中断程序的执行
- Apply Code Change　使用改变代码进行调试
- Show Next Statement　显示下一条要执行的语句
- Step Into　单步调试，进入被调函数内
- Step Over　单步调试，跳过被调函数
- Step Out　从被调函数中跳出，执行下一条语句
- Run to Cursor　运行到当前光标处

第二行自左至右分别为：

- Quick Watch　快速查看当前的调试状态
- Watch　打开一个独立窗口，用来显示用户要查看的变量值和类型。当用户输入变量名时，调试程序自动显示变量的值和类型
- Variables　打开一个独立窗口

该窗口内有 3 个标签，分别用来显示当前语句和上一条语句所用的变量、正在执行函数的局部变量以及 this 指针所指的对象的信息。

- Registers　打开一个独立窗口，显示 CPU 各个寄存器的状态
- Memory　打开一个独立窗口，显示内存的当前状态
- Call Stack　打开一个独立窗口，显示当前语句调用的所有函数，当前函数在顶部
- Disassembly　打开一个独立窗口显示反汇编代码

菜单栏

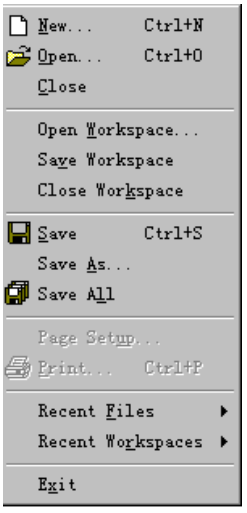


图 1.7　File 菜单

Visual C++ 6.0 主窗口的菜单栏中包含有下面 9 个主菜单项：File，Edit，View，Insert，Proiect，Build，Tools，Window 和 Help。

下面简单介绍这些菜单项的功能。

一、File 菜单

打开 File 菜单，出现如图 1.7 所示的下拉菜单项，共有 14 个选项。

该菜单的各个命令选项主要功能是用来对文件进行创建、打开、关闭、保存和打印等操作。常用命令介绍如下：

- New 命令

选择该命令，出现如图 1.8 所示的对话框。

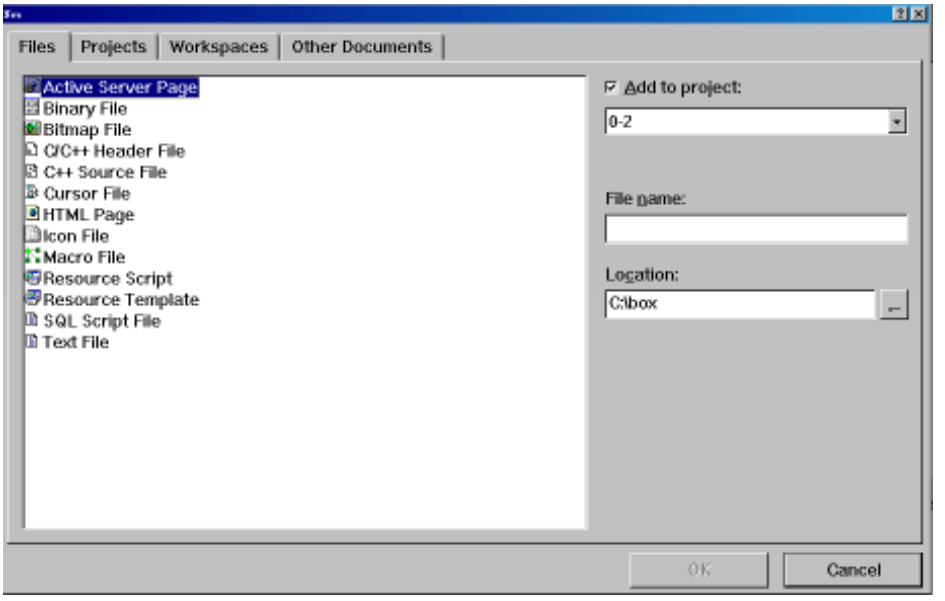


图 1.8　New **命令对话框**

该对话框是用来创建文件、项目、工作区以及其他文档的。它有 4 **个标签：**Files, Projects, Workspace **和** Other Documents**。**

① Files **标签，显示出可创建的文件类型，包括有：**

Active Server Page(**服务器页文件**)

Binary File(**二进制文件**)

Bitmap File(**位图文件**)

C/C++ Header File(C/C++ **头文件**)

C++ Source File(C++**源程序文件**)

Cursor File(**光标文件**)

HTML Page(HTML **页文件**)

Icon File(**图标文件**)

Macro File(**宏文件**)

Resource Script(**资源脚本文件**)

Resource Template(**资源模板文件**)

SQL Script File(SQL **脚本文件**)

Text File(**文本文件**)

选择某种文件类型后，在对话框内右边的 File name **文本框内键入要创建的文件名。**

② Projects **标签，显示出各种可供选择的项目类型，如图** 1.9 **所示。**

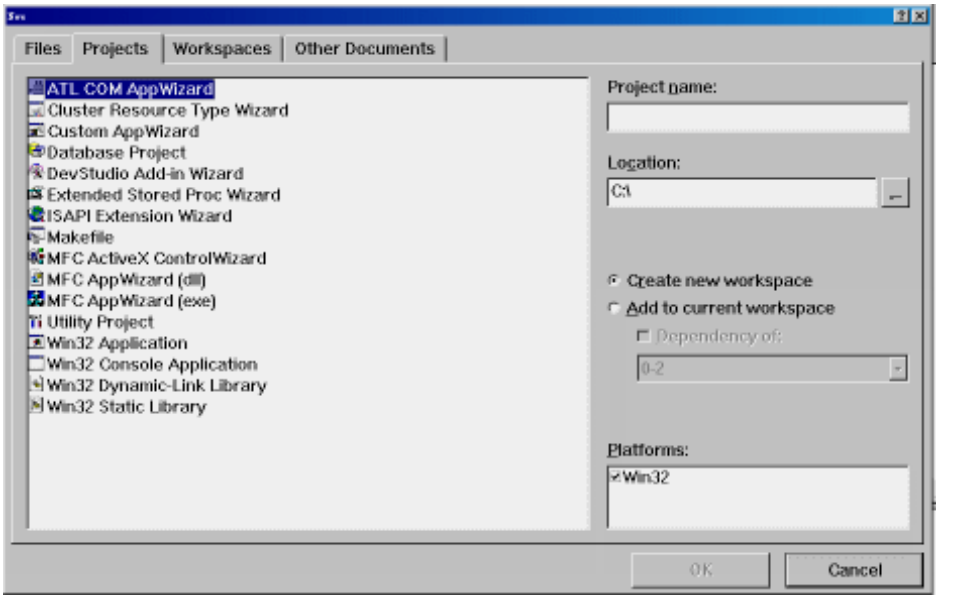


图 1.9　Projects　**标签窗口**

可供选择的项目类型包含有：

ATL COM AppWizard(ATL **应用程序创建向导**)

Cluster Resource Type Wizard(**簇资源类型创建向导**)

Custom AppWizard(**自定义的应用程序创建向导**)

Database Project(**数据库项目**)

DevStudio Add-in Wizard

Extended Stored Proc Wizard

ISAPI Extension Wizard

Makefile(C/C++**生成文件**)

MFC ActiveX ContorlWizard(MFC Activex **控制程序创建向导**)

MFC AppWizard(exe)(MFC **可执行程序创建向导**)

MFC AppWizard(dll)(MFC **动态链接库创建向导**)

New Database Wizard(**新数据库创建向导**)

Utility Project(**单元项目**)

Win32 Application(Win32 **应用程序**)

Win32 Console Application(Win32 **控制台应用程序**)

Win32 Dynamic-Link Library(Win32 **动态链接库**)

Win32 Static Library(Win32 **静态库**)

选择某种项目类型后，在该对话框内右边的 Project name **文本框中输入项目名。在** Location **框内输入或修改项目所在路径。**

该对话框的 Workspaces **标签和** Other Document **标签可以创建各种类型的工作区文件和文档，在这里就不做详细介绍了。**

・Open **命令**

选择该命令后，弹出“打开”对话框，如图 1.10 所示。

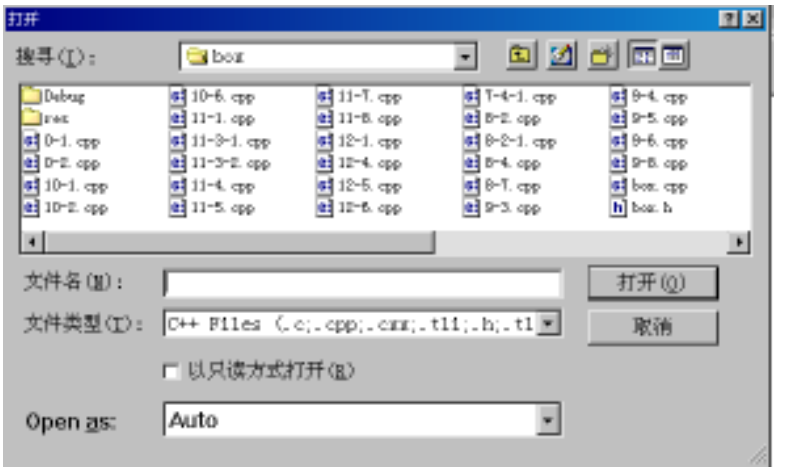


图 1.10 “打开”对话框

该对话框可用来打开 C++源文件、项目文件和其他文件。具体操作方法如下：

先在“搜寻”栏中选定要打开的文件的路径，再通过“文件类型”栏指定要打开文件的类型。此时，在文件名列表框中会出现所要选的文件名。双击文件名，或单击文件名再单击“打开”按钮，两种方式都可以打开所选的文件。

・Close 命令

该命令用来关闭在活动窗口中打开的文件。若该文件修改后尚未保存，系统会提示用户是否应该保存该文件。

・Open Workspace 命令

选择该命令将弹出“Open Workspace”对话框，用来打开该工作区的文件，也可打开其他文件。

・Close Workspace 命令

该命令用来关闭当前工作区的文件，选择该命令后，弹出一个对话框，提示用户是关闭所有文件(选择“是”)还是保留这些文件(选择“否”)。

・Save 命令

该命令用于保留当前窗口中的文件内容，并存放到原文件名中。如果该文件是未命名的新文件，则系统提示“Save As”对话框。

・Save As 命令

该命令用来将已打开的文件保存到一个新的文件名中。选择该命令，出现“Save As”对话框，用户可将新的文件名输入到该对话框中的文件名文本框内。

・Save All 命令

该命令用来保存当前窗口中所有被打开的文件的内容。如果某个文件尚未命名，系统将会提示用户先输入文件名。

・Page Setup 命令

该命令用来设置和格式化打印结果。选择该命令后，出现“Page Setup”对话框，使用该对话框为打印文档设置标题、脚注以及边距等。

・Recent Files 命令

该命令用来显示打开过的最近 4 个文件，单击该文件名可以将该文件打开。

・Exit 命令

该命令用来退出 Visual C++ 6.0 编译系统。在退出系统前，应将打开的文件保存。

二、Edit 菜单

该菜单的功能是对文档进行编辑和搜索，例如，包含撤消/恢复命令 Redo/Undo，剪贴命令 Cut/Copy/Paste 等。此外，常用命令还有：

・Find 命令

该命令的功能是用来在当前打开的文件中查找指定的字符串。选择该命令后，出现“Find”对话框，如图 1.11 所示。

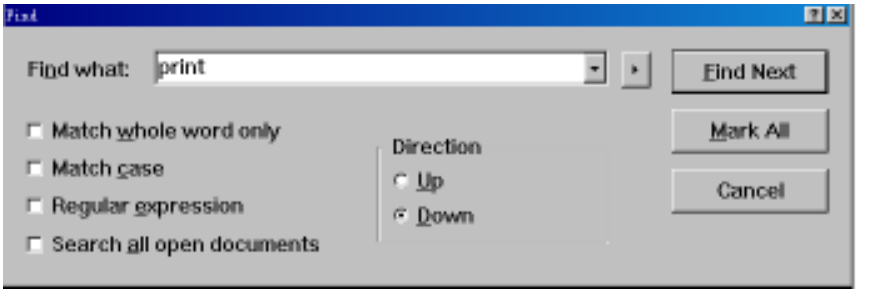


图 1.11　Edit 菜单中的“Find”命令对话框

在该对话框的 Find What 文本框中输入要查找的字符串，在 Direction 框内，选择查找方向：向前(Up)和向后(Down)。再在 4 个复选框内根据需要进行选择。

Match whole word only(仅匹配整个单词)

Match case(区分大小写字母)

Regular expression(正则表达式)

Search all open Documents(搜索所有打开的文档)

在使用正则表达式(Regular expression)查找文件中的匹配文本时，可使用特殊的字符序列去匹配文件中的某种文本模式，常用的查找模式有下述几种：

*：匹配多个字符。例如，a*可匹配的有 a1，a12，abc 等。

•：匹配一个字符。例如，a•可匹配的有 ab，a1，a5 等。

^：匹配以指定字符串开头的一行。例如，print^，表示所有的以 print 开头的行都可匹配。

+：匹配以指定字符串结尾的字符串。例如，+X 可匹配 StartX，EndX 等。

・Go To 命令

该命令用来指定如何将光标移到当前活动窗口的指定位置。选择该命令后弹出“Go To”对话框。

・Bookmarks 命令

该命令用来设置、命名、删除和读取书签。

・Breakpoint 命令

该命令是用来设置、删除和查看断点的。断点可分为位置断点、数据断点、消息断点和条件断点。断点将告诉调试器何时何处中断程序的执行，以便检查程序代码、变量和寄存器

的值。

・List Members 命令

该命令用来列出当前光标处对象类属的成员。可以通过双击相应成员名，将该成员添加到光标处。

・Parameter Info 命令

该命令用来显示当前光标处函数的参数信息，可在书写函数调用时提供参考，保证调用函数的实参与形参的一致性，从而减少错误。

三、View 菜单

该菜单包含有调试信息和控制屏幕显示方式的命令项，并提供从这里访问“ MFC ClassWizad ”的方法。常用命令项有：

・ClassWizard 命令

该命令用来显示“ MFC ClassWizard ”对话框，使用它可以进行 MFC 的常规操作。

・Full Screen 命令

该命令用来使源代码编辑区扩大到全屏幕。

・Workspace 命令

该命令用于显示项目工作区窗口。

・Output 命令

该命令用于显示数据输出窗口，在编译时该窗口将会显示出编译信息，包括出错信息。

・Debug Window 命令

选择该命令出现级联菜单，在级联菜单中列出了调试窗口的若干操作。

四、Insert 菜单

该菜单可以创建新类、资源、窗体，并将它们插入到文档中，也可以将文件作为文本插入到文档中，还可以添加新的 ATL 对象到项目中。常用的命令有：

・Resource Copy 命令

该命令用来复制选定的资源。

・File As Text 命令

该命令用来选择插入到文档中的文件。

・New ATL Object 命令

该命令用来启动 ATL Object Wizard，将添加新的对象到项目中。

五、Project 菜单

该菜单用于对项目和工作区的管理。可以选择指定项目为工作区中的当前(活动)项目，也可以将文件、文件夹等添加到指定的项目中去，还可以编辑和修改项目间的依赖关系。常用的命令有：

・Set Active Project 命令

该命令用来选择当前活动项目。

・Add To Project 命令

该命令用来将新文件或已有文件或者部件及控制加到指定的项目中去。

・Insert Project into Workspace 命令

该命令用来将项目插入到工作区中。

六、Build 菜单

该菜单包括用于编译、连接和运行应用程序的命令。常用命令有：

・Compile 命令

该命令用于编译显示在源代码编辑窗口中的源文件。在编译过程检查源文件中是否有语法错。如果发现错误(warning 或 error)，则将错误信息显示在输出窗口中。使用鼠标双击某行错误信息时，将在源代码编辑窗口中用粗箭头指向出错的代码行，方便用户修改。

・Build 命令

该命令用来创建当前文件项目。该命令实际上包含了对源文件或项目的编译和连接，最终生成可执行文件。如果被创建的文件或项目已被编译，则该命令将用来连接，生成可执行文件。在编译或连接中检查出语法错误时，将出错信息显示在输出窗口中。用户修改后，再进行创建，直到生成可执行文件为止。

・Rebuild All 命令

该命令用来对所有文件进行重新编译、连接，包含已编译过的文件。因此，此项操作耗时稍长。

・Execute 命令

该命令用来运行已生成好的可执行文件，并将运行结果显示到相应的环境中(例如 MS-DOS，Windows 98 或 Windows NT 等)。

・Start Debug 命令

选择该命令出现级联菜单，选取该菜单项便可启动调试器。这时，将用 Debug 菜单项代替 Build 菜单项。

・Go 命令

该命令用在调试过程中，从当前语句启动或者继续运行。

・Restart 命令

该命令将系统重新装载程序到内存中，并且将放弃所有变量的当前值。

・Stop Debugging 命令

该命令将中断当前调试过程，并返回到原来的编辑状态。

・Step into 命令

该命令用来设置单步执行程序。当程序执行到某一函数调用语句时，进入该函数体，并从第一行语句开始单步执行。

・Step over 命令

该命令也是单步操作命令，不同的是当程序执行到某一函数调用语句时，不进入该函数体内，直接执行该调用语句，然后停在该调用语句后面的语句。

・Step out 命令

该命令用来在单步执行时从某个函数体内跳出，调试该函数调用语句后面的语句。该命令是与 Step into 命令配合使用，先用 Step into 命令将单步执行植入某函数体内，发现不需要对该函数体内进行单步调试时，使用该命令跳出该函数体。

・Quick Watch 命令

选择该命令，将弹出“Quick Watch”对话框，通过该对话框可以查看和修改变量和表达式，或将变量和表达式添加到 Watch 窗口中。

七、Tools 菜单

该菜单中的命令用来浏览用户程序中定义的符号、定制菜单与工具栏，激活常用的工具或更改选项和变量的设置。

八、Window 菜单

该菜单的命令用来进行有关窗口的操作，常用的命令有：

・New Window 命令

该命令为当前项目文件打开一个新窗口。

・Split 命令

该命令用于将活动窗口进行分区。

・Close 命令

该命令用来关闭当前打开的窗口。

・Close All 命令

该命令用来关闭所有的窗口。

・Next 命令

该命令用来显示下一个窗口。

・Previous 命令

该命令用来显示前一个窗口。

九、Help 菜单

Help 用来获得大量的帮助信息。读者应该养成遇到问题查“Help”的习惯。

项目工作区

一、项目(Project)

项目是一些相互关联的源文件的集合，这些源文件组成一个程序，它们被编译、连接后生成一个可执行文件。

创建一个项目后，可以添加任何其他目录的文件到该项目中。添加文件到项目中并不改变文件的位置，而项目只是记录文件的名字和位置。

二、项目工作区(Project Workspace)

在 Visual C++ 中，文件、项目和项目配置是由项目工作区组织起来的。项目工作区的内容和设置通过项目工作区文件(.dsw)来描述，在建立一个项目工作区文件的同时，还生成项目文件(.dsp)和工作区选项文件(.opt)，用来保存工作区的设置。

三、项目工作区窗口

项目工作区窗口用来查看和修改项目中的所有元素。该窗口的底部提供了三种面板：类面板(Class View)**、资源面板**(Resource View)**和文件面板**(Files View)**。它们的功能如下所述：**

·类面板

该面板在项目工作区窗口中显示该项目中所有类及其成员函数。单击“+”号，打开树形结构的每一项，显示出某类的成员函数和数据成员。双击某一项，则在右边的源代码编辑窗口中显示该成员的源代码。如果该源代码已被显示，则使用光标进行指示。

·文件面板

该面板在项目工作区窗口中显示项目中的所有文件及其相互关系。单击“+”号时，依次打开树形结构的每一页，并显示出所有的资源文件、头文件和源代码文件。双击某一项时，则会在右边的源代码编辑窗口内打开该文件，显示其源代码。

·资源面板

该面板在项目工作区窗口中显示项目中的所有资源。单击“+”号时，依次打开树形结构的某一项，并显示出所有资源，包括字符串表、对话框图符及其版本信息。双击某一项时，则在右边的源代码编辑窗口内显示该资源的图形编辑窗口，可直接在该窗口内增添资源或修改资源特性。

1.2 C++ 单文件应用程序的实现

本节介绍使用 Visual C++ 6.0 **实现** C++**单文件应用程序的方法。**

一、编辑

单击主窗口菜单栏中的 File **菜单项，弹出如图** 1.7 **所示的下拉式菜单。**

单击下拉式菜单中的选项 New**，弹出“**New**”对话框(如图** 1.8 **所示)，该对话框中有** 4 **个标签。选择** Files **标签后，弹出的对话框中列出了** 13 **个选项。在编辑单个** C++**源文件时，双击** C++ Source File **选项，返回到** Microsoft Visual C++**的主窗口。这时可在源代码编辑窗口中输入源代码程序。例如，输入下述源程序：**

```
#include<iostream.h>
void main( )
{
    cout<<?This is a C++ source File.\n?;
}
```

按上述格式将该源代码程序输入到机器后，适当进行编辑，然后将其文件内容存入磁盘。存盘方法如下：

单击菜单栏中 File **菜单项，在其下拉菜单中选择** Save **选项或者** Save As **选项，弹出“保存为”对话框。选择存放文件的路径后，在“文件名”框内输入文件名(默认的扩展名为** cpp**)后按回车键，或按“保存”按钮即可。**

二、编译和连接

选择菜单栏中的 Build 菜单项，在弹出的下拉式菜单中单击 Compile xxx.cpp 选项，将对该源代码程序进行编译。如果发现错误，则将其错误信息显示在下面的输出窗口中，该信息将指出错误性质、出现位置及错误原因等内容。双击某条错误信息时，将会有一个提示箭头指向源代码编辑窗口中出错行的右边。用户可根据错误性质进行修改，然后再重新编译，直到没有显示错误信息为止。

单击下拉菜单项中的 Build xxx.exe 选项，将对编译好的程序进行连接。如果该程序尚未编译，它将会先编译，之后再进行连接。在连接过程中，如果发现错误，仍在输出窗口中显示其错误信息，用户需修改程序后再连接，直到没有错误为止，最后生成可执行文件。

三、运行

源代码程序经过编译连接后，生成可执行文件，此时便可运行。

单击 Build 菜单项的下拉式菜单中 Execute xxx.exe 选项，生成的可执行文件将被运行，运行后的输出结果被显示在 DOS 窗口的屏幕上。查看输出结果后，按任意键，将返回到 Visual C++的主窗口。

1.3 C++ 多文件应用程序的实现

C++程序允许由多个文件组成，多文件程序的实现要创建项目文件。本节介绍其具体的操作步骤。

一、编辑程序中的多个文件

编辑程序中多个文件的方法与前面讲述的编辑单文件的方法相同。

先单击 File 菜单项的下拉式菜单中的 New 选项，出现“ New ”对话框。在该对话框中，选择 Files 标签，在该标签的列表清单中双击 C++ Source File 选项，弹出 Visual C++ 6.0 的主窗口，在源代码编辑窗口中输入其中一个文件，接着再输入该程序的其他文件，并单击 File 菜单项的下拉式菜单中 Save 或 Save As 选项，分别将输入的文件按指定的文件名存好。

例如，下面给出一个由两个文件组成的程序例子。

第一个文件的文件名为 f1.cpp，具体内容如下：

```
#include<iostream.h>
int add(int,int);
void main( )
{
    int a,b;
    a=7;
    b=8;
    cout<<add(a,b) <<endl;
}
```

第二个文件的文件名为 f2.cpp**，其内容如下：**

```
int add(int x,int y)
{
    return x+y;
}
```

将上述两个文件按上述方法编辑好，并按指定的文件名存入指定的文件夹中。

二、创建项目文件

创建一个空的项目文件，用来存放该程序的上述两个文件。创建一个空的项目文件的方法如下：

先单击 File **菜单项的下拉式菜单中的** New **选项，出现“** New **”对话框，选择该对话框中的** Projects **标签。在该标签的对话框中，单击** Win32 Console Application **选项，这时在右边的** Platforms **框内出现** Win32**。**

接着，在该标签的对话框的右侧 Project name **文本框内输入一个项目文件名，例如，输入** bbb**，然后回车。此时，在** Location **文本框内生成一个路径名，该名可以修改。另外，在新建项目文件时，还应选择** Create new workspace **单选按钮，再单击对话框的** OK **按钮。这时，屏幕上出现“** Win 32 Console Application—Step1 of 1 **”对话框，如图** 1.12 **所示。**

该对话框上方出现提示信息：“ What kind of Console Application do you want to create**？(请选择你所要创建的控制台应用程序的类型)。”这时选择“** An empty project **”选项。此外，还有** 3 **个供选择的选项。单击该对话框下方的** Finish **按钮。**

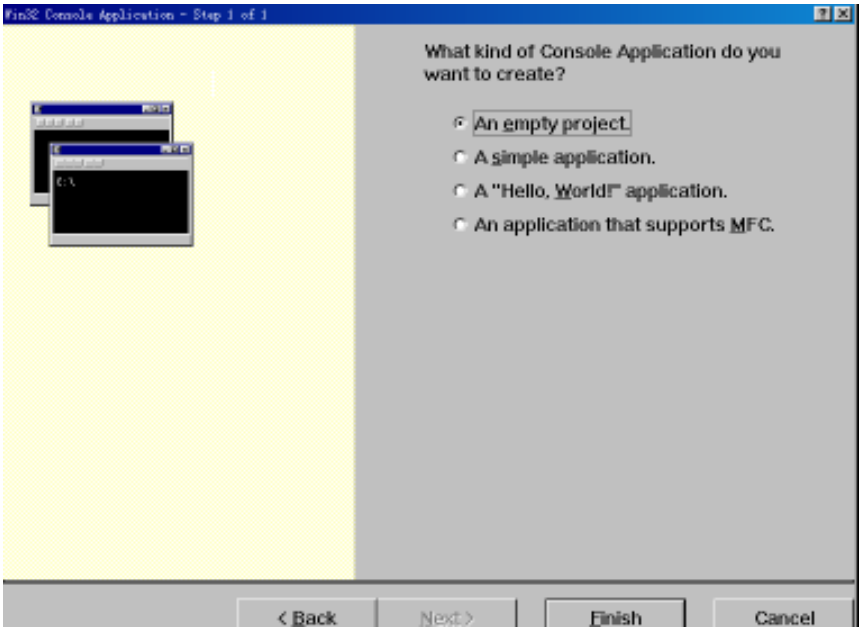


图 1.12 **“** Win32 Console Application—Step1 of 1 **”对话框**

这时，屏幕上出现“ New Project Information **”对话框，该对话框告诉用户所创建的控制**

台应用程序新框架项目的特性。单击该对话框下方的 OK 按钮，返回到 Visual C++ 6.0 主窗口。项目文件 bbb 创建结束。

三、将多个文件添加到项目中去

空的项目文件 bbb 创建好后，将事先编辑好的 f1.cpp 和 f2.cpp 文件添加到项目文件中。具体操作如下：

首先，在 Visual C++ 6.0 主窗口中，选择菜单栏中 Project 菜单项，在出现的下拉式菜单中单击 Add To Projects 选项，在弹出的级联菜单中单击 Files 选项，弹出“Insert Files into Project”对话框，如图 1.13 所示。

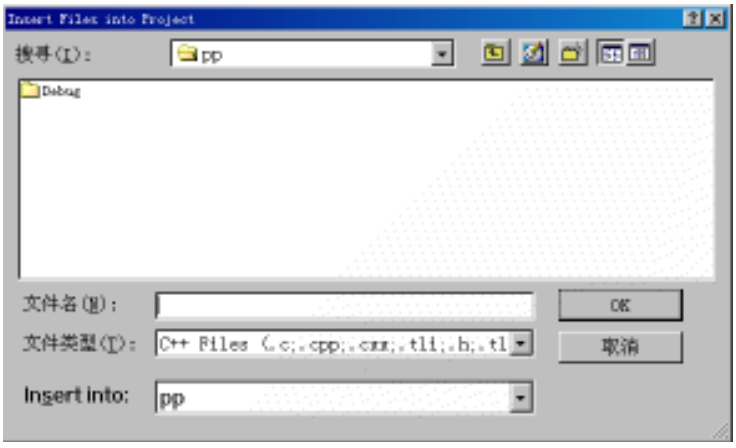


图 1.13 “Insert Files into Project”对话框

在该对话框中，先确定项目文件 bbb，显示在“搜寻”框内。打开 f1.cpp 和 f2.cpp 所在的文件夹，将这两个文件选择到该对话框的文件名框内。然后，单击 OK 按钮，则完成添加文件的任务。此时，项目文件 bbb 中将包含有 f1.cpp 和 f2.cpp。

四、编译连接项目文件

选择菜单栏中 Build 菜单项的下拉式菜单中的 Build bbb.exe 选项，按顺序编译项目中的各个文件。如果发现错误，仍将其错误信息显示在输出窗口中，并停止编译。修改其错误后，继续单击 Build bbb.exe 选项，则重新编译。第一个文件编译好后，再编译第二个文件，直到所有文件都编译好后，再进行连接。连接无错时，生成可执行文件 bbb.exe。

五、运行项目文件

单击 Build 菜单项的下拉式菜单中的 Execute bbb.exe 选项，将执行该项目文件，并将输出结果显示在 DOS 窗口中。

第 2 章　上机练习指导

2.1　第 1 章上机练习指导

上机目的要求

1. **熟悉** Visual C++ 6.0 **编译系统的常用功能。**
2. **学会使用** Visual C++ 6.0 **编译系统实现简单的** C++**程序。**

上机练习题

1. **熟悉** Visual C++ 6.0 **编译系统的常用功能。**

(1) **启动该编译系统，熟悉屏幕上显示的** Microsoft Visual C++ 6.0 **主窗口的组成部分及简单操作。**

① **标题栏**　② **菜单栏**　③ **工具栏**

④ **项目工作区窗口**　⑤ **源代码编辑区窗口**　⑥ **输出窗口**

⑦ **状态栏**

(2) **主窗口的常用操作。**

① **显示各菜单项的下拉式菜单，熟悉所包含的菜单项。**

② **右键单击工具栏，观察弹出的快捷菜单中各选项的内容，即选择某些选项，观察出现的工具栏。**

③ **使用菜单和工具栏，练习显示/隐藏项目工作区窗口和输出窗口的操作。**

2. **使用该编译系统，实现单文件程序的操作。**

以教材中例 1.1 **程序为例，学会使用该编译系统对单文件程序的操作方法。具体操作步骤如下：**

(1) **编辑源代码程序**

选择菜单栏中的 File **菜单，在下拉式菜单项中选择** New**，在“** New **”对话框中选择** C++ Source File **选项，出现主窗口。在主窗口的源代码编辑窗口中，将例** 1.1 **的源代码程序逐条输入，编辑后存入磁盘。**

(2) **源程序存盘和打开**

存盘方法如下：

选择菜单栏中的 File **菜单项，在该菜单项的下拉式菜单中选择** Save **或** Save As **选项，弹出“另存为”对话框。在该对话框中，先选好存放该文件的文件类，再输入文件名，按回车键或单击“保存”按钮即可。**

打开一个已被存盘的源程序的方法如下：

选择菜单栏中的 File **菜单项，在该菜单项的下拉式菜单中选择** Open **选项，弹出“打开”对话框。在该对话框中，先选择要打开的文件被存放的文件夹，再在类型为** cpp **的文件列表中查找要打开的文件，找到后或者双击该文件，或者单击该文件后再单击“打开”按钮。**

(3) 编译 C++单文件程序

打开将要编译的文件作为当前文件。选择菜单栏中的 Build 菜单项，在该菜单项的下拉式菜单中选择 Compile xxx.cpp 选项，该文件将被编译。在输出窗口中将显示出编译的过程和错误信息。

对编译中发现的错误应该进行修改，修改后再继续编译，直到没有错误为止。编译过程中出错是无法生成可执行文件的。

(4) 连接 C++单文件程序

单文件源程序被编译好后，可选择 Build 菜单的下拉式菜单中的 Build xxx.exe 选项，此时被编译好的目标程序将被连接。在连接过程中系统检查是否有错误，如发现错误，则停止连接，并将错误信息显示在输出窗口中。用户应该按错误信息中的提示及时修改，修改后再作连接，直到没有任何错误为止。此时系统将生成该程序的可执行文件。

用户也可以对编辑好的文件直接选择 Build xxx.exe 选项，系统发现该文件尚未编译时，对该文件先进行编译，然后再作连接，直至生成可执行文件。

(5) 运行 C++单文件程序

编译连接好的可执行文件，可选择 Build 菜单项的下拉式菜单中的 Execute xxx.exe 选项进行运行。运行后将结果显示在 MS-DOS 的窗口中，查看后按任意键即可关闭该窗口。

练习题

按照上述方法，将例 1.2 题、本章练习题中 1.4 题内 3 个小题都按要求上机操作一遍。

3. 使用该编译系统实现多文件程序的操作。

以教材中例 1.3 为例，学会使用该编译系统对多文件程序的操作方法。

(1) 编辑该程序的多个文件

按照前面讲述的单文件程序的编辑方法，将多文件程序中的每个文件进行编辑，并按指定的名字存盘。例如，将例 1.3 中的两个文件编辑完毕后，分别以 file1.cpp 和 file2.cpp 的名字存放在 C:\aaa 中。

如果一个程序包含有更多的文件，则按此方法，将每一个文件编辑好后存入盘中。

(2) 创建一个空的项目文件

多文件程序需要用项目文件进行编译连接和运行，创建空的项目文件的方法如下：

选择菜单栏中 File 菜单项的下拉式菜单中的 New 选项，出现“New”对话框。在该对话框中选择 Project 标签。在该标签所显示出的列表清单中选择 Win32 Console Application 选项，接着在该对话框右侧的 Project name 文本框内输入创建项目文件的名字。例如，输入 bbb，按回车键，此时，在下面的 Location 文本框内将显示出一个路径，名为 C:\bbb。在新建项目文件时，应选择 Create new workspace 单选按钮。单击 OK 按钮，退出“New”对话框，屏幕上出现“Win32 Console App1ication-Step 1 of 1”对话框，该对话框提示用户选择所要创建的控制台应用程序的类型。当选择了“An empty project”单选按钮后，单击 Finish 按钮，将退出该对话框。接着会出现“New Project Information”对话框，该对话框中显示出所创建的控制台应用程序新框架的项目特性。单击 OK 按钮，退出该对话框，返回到系统主窗口。项目文件 bbb 创建结束。

(3) 将文件添加到项目文件中

将事先编辑好的 file1.cpp 和 file2.cpp 两个文件添加到项目文件 bbb 中的方法如下：

选择菜单栏中 Project 菜单项的下拉式菜单中的“ Add To Projects ”选项，在该选项的级联菜单中选择 Files 选项，出现“ Insert Files into Project ”对话框。在该对话框中，向已打开的项目文件中添加文件，其方法如下：

先打开存放文件的文件夹 C:\aaa，这里两个文件 file1.cpp 和 file2.cpp 显示在列表框中。选择这两个文件名字，即单击第一个文件名，再按住 Shift 键单击第二个文件名字，它们都出现在“ 文件名 ”文本框中，单击 OK 按钮，即完成添入文件任务。

(4) 编译连接项目文件

选择 Build 菜单项的下拉式菜单中的 Build bbb.exe 选项后，开始编译并连接项目文件 bbb。如有错误，则应修改，直到没有错误为止，生成可执行文件 bbb.exe。

(5) 运行项目文

选择 Build 菜单项的下拉式菜单中的“ Excute bbb.exe ”选项，便可运行该项目文件。运行结果显示在 MS-DOS 窗口中，按任意键返回到该编译系统的主窗口。

4. 发现并记忆编译和连接中的错误信息，为今后调试程序积累经验。

上机调试本章练习题 1.5 题中的 3 个小题，将会发现一些编译和连接错误，根据输出窗口中显示出的错误信息，修改源程序，直到没有错误信息为止。记忆每次出现的错误信息，以备今后调试中出现类似信息时积累修改经验。例如，调试练习题 1.5 题中第 2 小题程序时，出现错误信息如下：

```
C:\aaa\ex4.cpp(4):error C2065: 'x':undeclared identifier
```

该信息告诉该文件(C:\aaa\ex4.cpp)中第 4 行出现了'x'没有被说明的错误。修改该错误，在源程序中增添说明 x 的说明语句“ int x; ”后，再编译连接，结果无错，生成可执行文件。

运行该文件，在 MS-DOS 窗口中输入 5，按回车键后，显示如下输出结果：

```
y=<<y<<
Press any key to continue
```

分析该结果它是不正确的，说明源程序中仍然有错误，检查源程序发现，输出上述错误结果是由下列语句造成的：

```
cout<<?y=<<y<<\n?;
```

将该语句修改为

```
cout<<?y=?<<y<<?\n?;
```

再编译连接并运行，输入 5 后，输出如下正确的结果：

```
y=25
```

该例子告诉我们：

① 在编译连接中，发现有错误必须改正，直到无错误，生成可执行文件为止。

② 运行可执行文件后输出的结果中还可能有错误，需要检查并修改源程序，直到输出结果正确为止。

源程序中有些错误，尤其是算法错，编译连接是检查不出来的。因此，执行完一个程序后还需对结果进行分析，检测输出结果是否正确。如果输出结果有错，还需修改源程序，再重新编译连接和运行，直到结果正确为止。

2.2 第2章上机练习指导

上机目的要求

1. 熟悉常量的各种不同的表示方法。
2. 学会对各种不同类型变量的说明方法。
3. 掌握变量赋初值与赋值的区别。
4. 学会数组的定义方法、数组元素的下标表示方法以及给数组赋初值和赋值的方法。
5. 学会对字符数组用字符串常量赋初值的方法和字符串的输出方法。

上机练习题

1. 上机调试教材例2.1中的程序，熟悉输出语句和输入语句的操作。

(1) 输入例2.1题的程序，并按教材中的数据输入，观查输出结果是否相同。

(2) 将程序中的下述三条语句注释掉，再运行程序，会发生什么现象？这说明什么问题？

```
cout<<?Enter int: ?;
cout<<?Enter double: ?;
cout<<?Enter char: ?;
```

注释掉一条语句的做法是在该语句前加上注释符//，表示该语句在编译中不起作用，相当于删掉该语句，这样做的目的在于恢复起来方便。

(3) 请试验一下，在输入的数据项之间除了用空格符外，还可以用什么字符来替代空格符呢？

2. 模仿教材中例2.2，按下列要求进行编程，并输出结果。

(1) 说明一个常整型变量M，其值为256，编程输出显示M=256。

(2) 定义一个双精度浮点型变量dd，并赋值为123.456，编程输出显示double dd=123.456。

(3) 定义一个长整型变量li，赋初值为100 000，编程输出显示M+li=100256。

3. 模仿教材中例2.3，按下列要求进行编程，并输出所要求的结果。

(1) 在main( )函数体外(前面)定义一个整型数组ia，它有5个元素，并给前3个元素相应赋初值为3，5和7。

(2) 在main( )函数体内定义一个一维的字符数组s1，并用字符串常量“abcdef”将它初始化。

(3) 在main( )函数体内再定义一个双精度浮点型数组d3，含3个元素，并给这三个元素分别赋值为12.3，45.6，78.9。

(4) 编程输出显示下列各表达式的值：

① ia[0]+ia[2]+ia[4]

② ia[1]*2−ia[3]

③ s1[2]+2

④ s1+3

⑤ d3[0]+d3[1]+d3[2]

⑥ d3[1]+ia[1]

4. **编程验证下列各问题：**

(1) **练习题** 2 **中** 2.2 **题内的第** 8 **小题，验证** 4 **种给数组赋初值的方法中哪一个是错的，并指出错误原因。**

(2) **练习题** 2 **中** 2.2 **题内的第** 10 **小题，验证** 4 **种输出显示中哪种符合要求，其余** 3 **种不符合要求的原因是什么？**

5. **上机运行练习题** 2 **中** 2.4 **题内第** 1 **小题的程序，并验证与事先分析的输出结果是否一样。**

将该程序中的下列输出语句:

```
cout<<a<<';'<<b<<';'<<d<<'\n';
```

改写为

```
cout<<a<<b<<d<<endl;
```

输出结果如何？从这个结果中受到了什么启发？

6. **上机运行练习题** 2 **中** 2.4 **题内第** 3 **小题的程序，并验证分析结果与输出结果是否一致。**

将程序中下列语句：

```
int ab[4][2]={{2,3},{4,5},{6,7},{8,9}};
```

改写为

```
int ab[4][2]={2,3,4,5,6,7,8,9};
```

运行后输出结果如何？

再将上述语句改为

```
int ab[4][3]={{2,3},{4,5},{6,7},{8,9}};
```

运行后结果是否有变化？

在程序执行下述输出语句，结果如何？

```
cout<<ab[1][1]+ab[0][2]<< ','<<ab[3][2]*ab[3][1]<<endl;
```

通过该题练习给二维数组赋初值的方法。

7. **上机运行练习题** 2 **中** 2.4 **题内第** 4 **小题的程序，并验证上机输出结果与自己分析的结果是否相同。**

该程序主要练习下列各知识点：

① 给一维字符数组赋初值有两种方法。

② 给二维字符数组赋初值的方法。

③ 一维字符数组各元素的输出显示方法。

④ 一维字符数组中字符串的输出显示方法。

⑤ 二维字符数组中存放的若干个字符串的输出显示方法。

⑥ 输出一个字符常量的两种方法。

上述各知识点在该程序中都有输出语句与之对应。

下面将本题的源程序分别作如下变动，分析每次变动后的执行情况。

(1) **将程序中下列语句：**

```
char s2[ ]=?abcde?;
```

改写为

```
char s2[5]= ?abcde?;
```

编译后会出现什么问题？为什么？

(2) 在程序中增加如下语句：

```
cout<<s3[0][0] <<s3[1][1] <<s3[3][3] <<endl;
```

分析并验证输出结果是什么？

(3) 将程序中下列语句：

```
cout<<c-1<<char(c-3)<<c<<endl;
```

改写为

```
cout<<c-1<<c-3<<c<<endl;
```

运行后输出结果如何？

(4) 通过该程序分析回答下列问题。

对一个字符常量来讲，什么情况下输出字符符号？什么情况下输出该字符的 ASCII 码值？

8. 上机运行练习题中 2.4 题的第 5 小题的程序，并验证运行结果与分析结果是否一致。

上机试一下关于 double 型数值的两种输出格式，一种是用指数形式，另一种是用小数点形式，一般地讲，何时用指数形式？何时用小数点形式？

将程序中下列语句：

```
dd=1.5e6;
```

改写为

```
dd=1.5e5;
```

这时，是用什么形式输出显示 dd 值？

再将程序中下列语句：

```
dd=.5e-2;
```

改写为

```
dd=.5e-4;
```

这时，输出显示 dd 值是用什么形式？

2.3 第 3 章上机练习指导

上机目的要求

1. 熟悉 C++语言中各种表达式的值和类型的计算方法。
2. 为了准确地计算出各种表达式的值，需要搞清各种运算符的功能、优先级和结合性。
3. 为了正确地确定各种表达式的类型，需要了解各种变量类型的转换。

上机练习题

1. 上机运行本章例 3.1 的程序，并验证输出结果与书中分析的结果是否一致。

将源程序中下列语句：

```
b=123+3.2e2-3.6/6-6/8;
```

修改为

```
b=123+3.2e2-3.6/6-6.0/8;
```

这时，b 值有何变化？为什么？

再将程序中下列语句：

```
a=c--++d;
```

修改为

```
a=--c-d++;
```

这时，计算上述表达式后，a,c,d 值各为多少？与原来的值相同吗？为什么？

2. 上机调试本章例 3.4 的程序，并将其输出结果与分析结果进行比较。

该程序中出现了 4 个逻辑表达式。逻辑表达式有其特定的运算规则，通过上机调试总结出其计算规律。

(1) 由多个逻辑与&&组成的逻辑表达式，计算时从左至右计算各操作数的值，遇到 0 值时，便停止对后面操作数的计算，确认该逻辑表达式的值为 0。

例如，本程序中下列语句：

```
!a&&b++&&c
```

(2) 由多个逻辑或||组成的逻辑表达式，计算顺序从左至右，当遇到某个操作数的值为非 0 时，则停止对后面操作数的计算，确认该逻辑表达式的值为 1。

例如，本程序中下列语句：

```
a||--b||c--；
```

(3) 对于由逻辑与、逻辑或和逻辑非组成的表达式，由于这三个逻辑运算符的优先级不同，逻辑非高于逻辑与，逻辑与又高于逻辑或，于是可将这种表达式看成为由逻辑或组成的表达式；而逻辑与和逻辑非作为操作数的表达式，按从左至右顺序，依次计算逻辑或的各个操作数，遇到其值为非 0 时，便停止计算，确认其值为 1。

例如，本程序中下列语句

```
a-8&&--b||c||b++;
```

可视为

```
(a-8&&--b)||c||b++;
```

又如，

```
--a||b&&c||++b;
```

可视为

```
--a||(b&&c)||++b;
```

在本程序中，加入下列语句，分析输出结果，并上机验证。

```
a=b=c=1;
!b&&a++||c+||!a&&b++;
cout<<a<<', '<<b<<', '<<c<<endl;
8||b++&&c--&&2||!b--;
cout<<a<<', '<<b<<', '<<c<<endl;
```

3. 上机调试本章例 3.5 的程序，并验证其输出结果。

该程序主要练习条件表达式的计算方法，特别是多个三目运算符组成的条件表达式计算时，按自右至左的结合性进行，而条件表达式的类型取决于冒号前后两个操作数中类型高的类型。上机调试时，要注意这些问题。

将该例程序中下列语句

```
k=i-j ? i+j:i+j ? i:j;
```

改写为

```
k=i+j ? i-j:i+j ? i++:j-- ? i:j;
cout<<i<<', '<<j<<', '<<k<<endl;
```

先分析输出结果，再上机验证。

4. **上机验证练习题中第** 3.4 **题的** 1，2，3 **题，与所分析的结果进行比较。**

5. **将练习题中第** 3.5 **题按下列方法编程，验证** 6 **个表达式的值和** a,b **值。**

要求：6 **个表达式中各个变量值间不相互影响。**

```
#include<iostream.h>
void main( )
{
    int a(8),b(4);
    int c=!a&&++b;
    cout<<c<<','<<a<<','<<b<<endl;
    a=8; b=4;
    c=b || a-4&&a/b;
    cout<<c<<','<<a<<','<<b<<endl;
    a=8 ; b=4 ;
    c=(a=2,b=3,a>b ? a++:b++);
    cout<<c<<','<<a<<','<<b<<endl;
    a=8; b=4;
    c=(++b,a=10,a+6);
    cout<<c<<','<<a<<','<<b<<endl;
    …
}
```

另外两个表达式的求值，请读者自己加上。

6. **练习编程。**

(1) **将练习题** 3.6 **中第** 1 **题编程后上机调试。**

下面给出一些主要语句，将它们写成 C **语言程序，即在一个文件中只有一个** main( )**的形式。**

```
float x,y;                  //定义两个变量 x 和 y
cin>>x>>y;                  //从键盘上输入两个 float 型数值，存放到变量 x 和 y 中
if (x<=y)
    cout<<x<<endl;    //该 if 语句将判断，若 x≤y，则输出 x 值，否则输出 y 值
else
    cout<<y<<endl;
```

于是将在屏幕上显示出 x **和** y **中较小的值。**

(2) **将练习题** 3.6 **的第**(3)**题编程后上机调试。**

提示：

```
double c,f;              //c 和 f 用来表示摄氏温度值和华氏温度值，它们是 double 类型的变量
c=(f-32)*5/9;            //这是公式 C=(F-32)*5/9 的写法
cout<<c<<endl;           //输出显示由华氏温度转成摄氏温度的值
```

如果将程序中给定的计算公式写成如下格式，请上机调试一下，看结果如何：

```
c=5/9*(f-32);
```

这时，计算出的 c 值为 0，请想一想为什么？

如果再将该公式写成如下格式，上机调试又如何？

```
c=5.0/9*(f-32);
```

2.4 第 4 章上机练习指导

上机目的要求

1. **练习各种语句的定义格式、功能和使用方法，特别是条件语句、开关语句和三种循环语句的用法。**
2. **熟悉预处理命令的用法。**
3. **学会使用学过的语句，练习编写简单的程序。**

上机练习题

1. **上机调试本章例 4.1 的程序，按下列要求输入数据后，分析并验证输出结果。**

(1) Input x,y: 2.6　3.4 ↵

(2) Input x,y: 7.8　1.8 ↵

(3) Input x,y: 2.5　2.5 ↵

上机验证，将程序中的条件语句换成下列语句是否可以？

```
if(x>=y)
    if(x>y)
        cont<<"x>y\n";
    else
        cout<<"x=y\n";
else
        cout<<"x<y\n";
```

请读者再使用一种 if 语句实现上述功能，并上机验证其正确性。

2. **上机调试本章例 4.2 程序后得到输出结果，并与分析结果进行比较。再回答程序中的 else 短语与哪个 if 短语配对？是靠它近的一个呢？还是靠它远的那个？**

如果要使程序中的 else 短语与靠它远的一个配对，则该程序应如何修改？

提示：参见例 4.3。

3. **上机调试本章例 4.4 程序，并验证其正确性。**

如果将开关语句中每个 case 后面的语句序列中的 break 都去掉，将出现什么现象？为什

么？请上机试试。

4. 将本章例 4.5 程序中的 switch 语句改写为 if 语句，并保证与该程序的功能相同。上机调试并说明之。

5. 将本章例 4.12 求素数程序按下列要求进行修改。

该程序中给出的求素数算法效率较低，请按下列算法修改：

求某个数是否是素数，不必从 2 一直试除到它自身，数学上可以证明，只需从 2 试除到该数的平方根的值就可以了。这种算法可以提高效率。

按上述算法修改该程序后，上机验证其正确性。

6. 上机调试本章例 4.17 程序，并将输出结果与分析结果相比较。

请修改该程序，要求在程序中不出现 continue，修改时尽量少地变动原程序。

7. 上机调试本章练习题 4.4 中 1，3，5，6 题，并验证分析结果。

按下列要求对每个程序进行修改，然后输出结果。

(1) 将第 1 题中 while 循环体内的 break 语句去掉，将退出循环的条件放在 while 关键字后边的括号内，并使该程序功能不变。

(2) 将第 3 题的前一个 if 语句中的 elseif 体内的 if-else 语句内的 else 短语，修改成为与距它远的 if 配对，即将该程序中的 if-else if 语句形式改为 if-elseif-else 形式。

(3) 将第 5 题中开关语句内 case 5 后面的语句序列的最后加一条 break 语句，即为：

```
case 5: switch(b)
        {
          case 5: i++;break;
          case 6: j++;break;
          default:i++;j++;
        }
        break;
```

分析修改前后输出结果是否相同。

(4) 将第 6 题中程序的分析结果与上机结果进行比较。

提示：字符数组 input[ ]中存放的字符串内 1 和\1 是不同的，其中 1 表示字符“1”，而\1 表示为数字 1。另外，开关语句中，case 1 后面的语句序列是一条 while 循环语句，其循环体为空语句。

8. 编程练习。

(1) 本章练习题 4.5 中第 1 题。

求 100 之内的自然数中奇数和。

分析：该题可用循环实现，三种循环方式都可以，比较方便的是用 for 循环。循环是从 1 开始到 100，由于求奇数之和，每个循环后，循环变量增 2，将奇数和存放在变量 s 中。

程序：

```
#include<iostream.h>
void main( )
{
    int s=0
```

```
    for(int i=1;i<100;i+=2)
        s+=i;
    cout<<s<<endl;
}
```

练习题

请将上述程序中的 for **循环修改成为** while **循环，并保证其功能不变，并上机调试。**

(2) **编程输出下列图案。**

```
        *
      * * *
    * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

分析：将上面图案分成如下两部分：

```
      *
    * * *
  * * * * *
* * * * * * *
```

和

```
    * * * * *
      * * *
        *
```

先编写打印上面图案的程序如下：

```
int i,j;
for (i=1;i<4;i++)
{
   for(j=1;j<i+6;j++)
   {
      if(j<=6-i)
         cout<<' ';
      else
         cout<<'*';
     }
    cout<<'\n';
}
```

打印下面图案的程序如下：

```
for(i=4;i>1;i--)
```

```
{
   for(j=1;j<i+5;j++)
      if(j<=7-i)
         cout<<' ';
      else
         cout<<'*';
   cout<<'\n';
}
```

请将这两部分程序合在一起，运行后将打印出该题所要求的图案。

练习题

请再编程打印出如下图案，即一个空的菱形。

```
        *
      *   *
    *       *
  *           *
    *       *
      *   *
        *
```

(3) **求两个整数的最大公约数和最小公倍数。**

分析：最大公约数是同时能够整除这两个数的最大整数；最小公倍数是能被这两个数同时整除的最小整数。两者之间关系如下：

最小公倍数=两个数之积/最大公约数

先求两个数的最大公约数：

假设两个整数为 m **和** n**，并且** m＞n**。**

使用辗转相除法(下面程序中，t **是一个中间变量)：**

```
while(n!=0)
{
    t=m%n;
    m=n;
    n=t;
}
```

为了求最小公倍数，使用原来的 m **和** n**，可将它们分别保存在变量** m0 **和** n0 **中。**

程序如下：

```
#include<iostream.h>
void main( )
{
      int m,n,t;
      m=15;
```

```
    n=10;
    if(m<n)
  {
        t=m;
        m=n;
        n=t;
    }
    int m0=m,n0=n;
    while(n!=0)
    {
        t=m%n;
        m=n;
        n=t;
    }
    cout<<"最大公约数为"<<m<<endl;
    cout<<"最小公倍数为"<<m0*n0/m<<endl;
  }
```

(4) **求下列分数序列的前** 15 **项之和：**

2/1，3/2，5/3，8/5，13/8，21/13 …

分析：先分析得出该分数序列的通项公式。假定分子为 i，**分母为** j，**则前一项为** i/j，**而后一项是** (i+j)/ i。**又假定求得的和存放在** sum **中，先将第一项存到** sum **中，开始时** sum **为** 0。

```
sum+=i/j;
```

下一项 i **和** j **为**

```
t=i;     //t 是一个临时变量保存 i
i+=j;    //下一项的 i
j=t;     //下一项的 j
```

循环相加 15 **次。**

程序如下：

```
#include<iostream.h>
void main( )
{
    int a;
    double i=2,j=1,sum=0,t;
    for(a=1;a<=15;a++)
    {
        sum+=i/j;
        t=i;
        i+=j;
```

```
            j=t;
        }
        cout<<"SUM="<<sum<<endl;
    }
```

(5) **输入** 4 **个** int **型数，按由大到小的顺序输出。**

分析：首先使用输入语句从键盘上输入 4 **个** int **型数，分别放在** a,b,c **和** d **中。**

当 a<b **时，将** a,b **交换，**a **中数比** b **大；**

当 a<c **时，将** a,c **交换，**a **中数比** b,c **都大；**

当 a<d **时，将** a,d **交换，**a **中数比** b,c,d **都大。**

这时 a **中存放最大的数。接着，**

当 b<c **时，将** b,c **交换，**b **中数比** c **大；**

当 b<d **时，将** b,d **交换，**b **中数比** c **和** d **都大。**

这时 b **中存放是次大的数。再接着**,

当 c<d **时，将** c **与** d **交换，**c **中数比** d **大。**

这时 c **中是次次大的数，**d **中是最小的数。**

最后，按 a,b,c,d **顺序输出，即是由大到小的顺序。**

程序如下：

```
#include<iostream.h>
void main( )
{
        int a,b,c,d,t;
        cout<<"输入 4 个整数:"；
        cin>>a>>b>>c>>d;
        if(a<b)
        { t=a; a=b; b=t;}
        if(a<c)
        { t=a ;a=c; c=t;}
        if(a<d)
        { t=a; a=d; d=t;}
        if(b<c)
        { t=b; b=c; c=t ;}
        if(b<d)
        { t=b; b=d; d=t;}
        if(c<d)
        { t=c; c=d; d=t;}
        cout<<"按由大到小顺序输出如下：\n";
        cout<<a<<','<<b<<','<<c<<','<<d<endl;
}
```

执行该程序，显示如下信息：

输入 4 个整数：18　21　10　35

输出结果如下：

按由大到小顺序输出如下：

35，21，18，10

2.5　第 5 章上机练习指导

上机目的要求

1. 掌握关于函数的定义和说明、参数和返回值以及调用方式的有关概念及实现方法。
2. 掌握内联函数、重载函数和设置函数参数默认值的功能和方法。
3. 了解变量和函数的存储类种类、作用和定义方法，并在程序中能够正确引用。

上机练习题

1. 上机调试本章例 5.4 程序，并验证其输出结果。

通过调试该程序，要求对设置函数默认参数的若干规则进行深入了解。为此，请将该程序按下列要求进行改动，并分析改动后出现的问题。

(1) 设置的默认参数可以是表达式，但表达式中出现的应是全局变量。请将该程序中放在 main( )前的下列两个语句放在主函数内：

```
int q(5),p(7);
int sum_int(int a,int b=p+q,int c=p*q);
```

调试后会出现什么现象？

(2) 规定在设置默认参数时应从右端向左端设置。请将该程序中 sum_int( )函数的默认参数设置作如下改动：

```
int sum_int(int a=p+q,int b=p*q,int c);
```

调试后会出现什么现象？

(3) 如果有函数的说明时，设置默认参数应放在函数的说明中，而不是放在函数的定义中。

请将该程序中对函数 sum_int( )设置的默认参数放在函数的定义中，会发生什么现象？

2. 上机调试本章中例 5.10 程序，并获得输出结果。

本程序中使用了内联函数，这是 C++语言与 C 语言的区别之一。在 C++语言中可以用内联函数，而 C 语言中不允许。C++语言中的内联函数的作用相当于 C 语言中的带参数的宏定义，它们都是用替换代替调用。在一定条件下，可以提高运行效率。

通过本程序的调试，进一步熟悉内联函数的使用方法。

请将本例中两处出现的关键字 inline 去掉，变成一般函数的调用，该程序仍然可以运行，并具有相同的结果，请上机验证。

3. 上机调试本章例 5.12 程序，并验证其输出结果。

重载函数是 C++语言与 C 语言的又一个区别。使用重载函数时，一定要使所定义的若干

个同名函数在参数的个数上、类型上或顺序上有所区别，否则无法实现重载。

在本例程序中有 3 个 max( )函数进行重载，它们之间的参数在个数上是否相同？

4. 上机调试本章例 5.17 程序，并将输出结果与书中给出的结果进行比较。

将该例程序的 main( )和 fun( )函数都出现的 static int b 的 static 说明符去掉，其他不变。调试时会出现问题吗？输出结果与教材中的结果相同吗？为什么？

5. 上机调试本章例 5.18 程序，将输出结果与教材中结果进行比较。

该程序是一个由 3 个文件组成的。多文件程序的运行方法在教材中第 1.3 节中讲述过。请参照所讲述的方法将本例程序进行编译、连接和运行。通过本例学会实现多文件程序的方法。

通过调试该例程序，掌握下述各种存储类变量的用法。

(1) 外部存储类变量的定义、说明和引用。

(2) 外部静态类变量的定义及引用。

(3) 自动类变量的定义和引用。

6. 上机调试本章练习题 5.4 的第 1 题，并将分析结果与上机输出结果进行比较。

将本例程序中 fun( )函数内的“ static int a; ”改为“ int a(0); ”，即去掉 static 的说明符。调试运行后，结果有什么不同？为什么？进而掌握内部静态类变量的特点以及与自动类变量的区别。

7. 上机调试本章练习题 5.4 中第 4 题，并将分析结果与上机输出结果进行比较。

该例题程序中 fac( )函数是一个递归函数，使用递归调用的方法求一个整数的阶乘。如果将该函数为“ static int b=1; ”改变为“ int b=1 ;”，请观察有何变化？进而说明内部静态变量的重要作用。

8. 编程练习

(1) 从键盘上输入 10 个浮点数，求其和及平均值，要求写出求和及平均值的函数。

分析：该程序有 3 个函数：main( ),sum( )和 average( )。主函数 main( )中包含有如下操作：输入 10 个浮点数存放在一个数组中，使用循环语句从键盘上输入值，接着分别调用 sum( )函数求 10 个浮点数之和，调用 average( )函数求 10 个浮点数的平均值，最后将 10 个浮点数的和值与平均值输出显示。

程序如下：

```
#include<iostream.h>
double s,sum(double b[ ],int n),average(int n);
void main( )
{
    double a[10];
    cout<<"Input 10 doubles: ";
    for(int i=0;i<10;i++)
        cin>>a[i];
    s=sum(a,10);
    double ave=average(10);
    cout<<"SUM="<<s<<', '<<"AVERAGE="<<ave<<endl;
}
```

```
double sum(double b[ ],int n)
{
    double sum=0;
    for(int i=0;i<10;i++)
        sum+=b[i];
    return sum;
}
double average(int n)
{
    return s/n;
}
```

(2) **从键盘上输入** 10 **个** int **型数，去掉重复的，将其剩余的由大到小排序输出，要求写出一个排序函数。**

分析：先将 10 **个** int **型数从键盘上输入，并放在一个** int **型数组中。接着，在数组中将重复的数组元素去掉。其方法是在比较中发现有重复的就用数组中最末元素替换，同时更新数组元素个数。最后，将数组中的元素使用排序函数** sort( )**进行排序，并输出显示最后排序好的数组元素。**

排序函数 sort( )**是按照“冒泡”排序算法进行编程的。**

程序内容如下：

```
#include<iostream.h>
void main( )
{
    int a[10],n=10;
    void sort(int b[ ],int n);
    cout<<"Input 10 integer: ";
    for(int i=0;i<n;i++)
        cin>>a[i];
   for(i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
          if(a[i]= =a[j])
          {
              a[j]=a[--n];
               j--;
          }
    }
    sort(a,n);
    for(i=0;i<n;i++)
        cout<<a[i] <<"    ";
```

```
    cout<<'\n';
}
void sort(int b[ ],int n)
{
    for(int i=1;i<n;i++)
        for(int j=0;j<n-i;j++)
          if(b[j]<b[j+1])
          {
              int t=b[j];
               b[j]=b[j+1];
               b[j+1]=t;
          }
}
```

(3) **给定某个年、月、日，例如，**2000 **年** 7 **月** 25 **日。计算出这一天是该年的第几天。要求写出计算闰年的函数和计算日期和函数。**

分析：该程序将包含一个主函数和两个被调用函数：一个是计算某一年是否是闰年的函数 leap( )**，另一个是计算日期的函数** sum_day( )**。**

主函数中，首先从键盘上输入某天的年、月、日，存放在相应的变量 year,month **和** day **中。接着，调用** sum_day( )**函数，计算出这一天是该年的第几天，计算时按非闰年计算，即** 2 **月份** 28 **天。然后，再判断该年是否是闰年，并且该月是否是大于** 2 **月。如果是闰年，且月份又大于** 2 **月，则计算的天数加** 1 **。最后将其结果输出显示。**

计算闰年函数 leap( )**是按照闰年的定义，使用逻辑表达式返回。闰年时返回** 1**，否则返回** 0**。**

计算日期函数 sum_day( )**是根据输入的月份和日数，按指定的每月的总天数的数组进行相加计算，这时** 2 **月份按** 28 **天计算，将每月天数存放在一个数组中，使该数组的下标与月份数相同。例如，**5 **月** 4 **日，计算天数时如下所示：**

days_mronth[1]+days_month[2]+days_month[3]+days_month[4]+4**，**

并将这个表达式值返回。

程序内容如下：

```
#include<iostream.h>
int sum_day(int,int),leap(int);
void main( )
{
    int year,month,day;
    cout<<"请输入一个日期(yyyy mm dd):";
    cin>>year>>month>>day;
    int days=sum_day(month,day);
    if(leap(year)&&month>2)
        days++;
```

```
        cout<<"是这一年的第"<<days<<"天.\n";
}
int sum_day(int month,int day)
{
        static int days_month[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
        for(int i=1;i<month;i++)
                day+=days_month[i];
        return day;
}
int leap(int year)
{
        int leap=year%4==0&&year%100!=0||year%400=0;
        return leap;
}
```

当输入 2000 **年** 7 **月** 25 **日时，输出信息为：**

请输入一个日期(yyyy mm dd)**：**2000 7 25

是这一年的第 207 **天**.

(4) **写出一个函数，要求将输入的十六进制数转换成十进制数。要求函数形参用指针或引用。**

分析：由于十六进制数的表示中，除了有数字 0～9**，还有** 6 **个字母** a,b,c,d,e,f**。因此，输入十六进制数时，应采用** char **型，将它存放在一个字符数组中。然后，调用一个转换函数** htoi( )**，将输入的字符型数组中的十六进制数转换为十进制的。在转换函数中，应考虑到下述三类不同字符的转换方法：**

① **数字字符**'0'**至**'9'

② **小写字母**'a'**至**'f'

③ **大写字母**'A'**至**'F'

程序内容如下：

```
#include<iostream.h>
void main( )
{
        int htoi(char s[ ]);
        cout<<"输入一个十六进制数:";
        char s1[20]=" ";
        cin.read(s1,20);
        cout<<endl;
        int n=htoi(s1);
        cout<<"该数转换为十进制数为"<<n<<endl;
}
```

```
int htoi(char s[ ])
{
    int n=0;
     for(int i=0;s[i]!= '\0';i++)
     {
          if(s[i]>= '0'&&s[i]<= '9')
               n=n*16+s[i]- '0';
          if(s[i]>= 'a'&&s[i]<= 'f')
               n=n*16+s[i]- 'a'+10;
          if(s[i]>= 'A'&&s[i]<= 'F')
               n=n*16+s[i]- 'A'+10;
     }
     return n;
}
```

执行该程序显示下述信息：

输入一个十六进制数：ffff〈ctrl+z〉

该数转换为十进制数为 65535

(5) **输入** 5 **个学生** 4 **门功课的成绩，然后求出：**

a.**每个学生的总分**

b.**每门功课的平均分**

c.**输出总分最高学生的姓名和总分数**

分析：首先定义一个 char **型数组** name[5][10]**用来存放** 5 **个学生的姓名，再定义一个** int **型数组** score[5][4]**来存放** 5 **个学生的** 4 **门功课成绩，再定义一个数组** al[5]**用来存放每个学生的总成绩。该程序将有** 4 **个函数组成：**

① **主函数** main( )，**包含所有定义的数组，并赋初值。先调用求每个学生** 4 **门功课总分的函数** all_scor( )，**通过** for **循环将每个学生功课总分显示在屏幕上，同时存放在数组** al[5]**中。**

再调用求每个学生 4 **门功课平均成绩的函数** aver_scor( )，**通过** for **循环将每个学生功课的平均成绩输出显示在屏幕上。**

最后，调用计算 5 **个学生中总分最高的函数** high_scor( )，**返回最高分，并通过传址调用获得最高分学生下标序号，用来输出该学生姓名。**

② **计算学生总分函数** all_scor ( )，**该函数有一个参数，它是一维数组，返回值为** int **型变量，即返回该学生** 4 **门功课的总分。**

③ **计算学生平均分函数** aver_scor( )，**该函数有一个数组参数，返回值为** double **型量。该函数中调用了** all_scor ( )**函数。**

④ **求出最高总分及该总分对应的下标值函数** high_scor( )。**该函数有** 2 **个参数：一个是数组，另一个是** int *型指针。**该函数返回值为** int **型量。**

程序内容如下：

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

```
int all_scor(int a[ ]),high_scor(int a[ ],int *i);
double aver_scor(int sc[ ]);
void main( )
{
    char name[ ][10]={"Ma","Wang","Li","Huang","Kang"};
    int score[ ][4]={{89,86,75,90},{91,82,75,80},
        {78,93,85,80},{90,79,95,85},{68,75,81,83}};
    int al[5];
    cout<<"每个学生功课的总分: ";
    for(int i=0;i<5;i++)
        cout<<(al[i]=all_scor(score[i]))<<"    ";
    cout<<"\n 每个学生功课的平均成绩: ";
    for(i=0;i<5;i++)
        cout<<aver_scor(score[i])<<"    ";
    int n=0,high;
    high=high_scor(al,&n);
    cout<<"\n 学生姓名: "<<name[n] <<"  最高分: "<<high<<endl;
}
double aver_scor(int sc[ ])
{
    return (double)all_scor(sc)/4;
}
int all_scor(int a[ ])
{
    int s=0;
    for(int j=0;j<4;j++)
        s+=a[j];
    return s;
}
int high_scor(int a[ ],int *i)
{
    for(int j=1;j<5;j++)
        if(a[0]<a[j])
        {
            a[0]=a[j];
            *i=j;
        }
    return a[0];
}
```

执行该程序后，输出结果如下：

每个学生功课的总分：340　328　336　349　307

每个学生功课的平均成绩：85　82　84　87.25　76.75

学生姓名：Huang　**最高分：**349

(6) **使用递归方法将一个** n **位整数转换为一个字符串。**

分析：首先从键盘上输入一个 n **位整数**(n＞1)**。然后，调用转换函数** convert( )**将整数** n **位整数转换为字符串，其长度为** n**。**

转换函数 convert( )**定义为递归函数。递归条件如下：**

```
if((a=n/10)!=0)
    convert(a);
```

其中，n **是一个待处理的** n **位整数，**a **是一个** int **型变量。当** n **是** 1 **位整数时，**if **体将不被执行。当** n **为** 2 **位整数时，执行** if **体，这时** a **为** 1 **位整数。依次类推。**

程序分析：

```
#include<iostream.h>
void main( )
{
    int num;
    void convert(int);
    cout<<"输入一个整数: ";
    cin>>num;
    cout<<"输出的字符串是 ";
    convert(num);
    cout<<'\n';
}
void convert(int n)
{
    int a;
    if((a=n/10)!=0)
        convert(a);
    char c=n%10+'0';
    cout<<c;
}
```

执行该程序，显示如下信息：

输入一个整数：1357

输出的字符串是 1357

练习题

在本例中,当输入负整数时，将如何处理呢？请修改程序后上机调试。

(7) **使用重载函数的方法定义两个函数，用来分别求出两个** int **型数的点间距离和浮点型数的点间距离。**

分析：首先定义两个重载函数。

```
double distance(int x1,int y1,int x2,int y2)
{
    double s;
    s=sqrt((x1−x2)*(x1−x2)+(y1−y2)*(y1−y2));
    return s;
}
double distance(double x1,double x2,double y1,double y2)
{
    double s;
    s=sqrt((x1−x2)*(x1−x2)+(y1−y2)*(y1−y2));
    return s;
}
```

这里，前一个 distance( )**函数是用来计算两个** int **型数的点间距离，而后一个** distance( ) **函数是用来计算两个** double **型数的点间距离。计算两个点间距离公式如下：**

$$S=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

程序内容如下：

```
#include<iostream.h>
#include<math.h>
void main( )
{
    double distance(int,int,int,int),distance(double,double,double,double);
    int x1=5,y1=8,x2=12,y2=15;
    double xd1=1.5,yd1=5.2,xd2=3.7,yd2=4.6;
    cout<<"两个 int 型数的点间距离：";
    double disi=distance(x1,y1,x2,y2);
    cout<<disi<<endl;
    cout<<"两个 double 型数的点向距离：";
    double disd=distance(xd1,yd1,xd2,yd2);
    cout<<disd<<endl;
}
double distance(int a1,int b1,int a2,int b2)
{
    double s=sqrt((a1−a2)*(a1−a2)+(b1−b2)*(b1−b2));
    return s;
}
```

```
double distance(double a1,double b1,double a2,double b2)
{
    return sqrt((a1−a2)*(a1−a2)+(b1−b2)*(b1−b2));
}
```

(8) **编写一个程序验证：任何一个充分大的偶数**(＞=6)**总可以表示成两个素数之和，要求编写一个求素数的函数** prime( )**，它有一个** int **型参数，当参数值为素数时返回** 1**，否则返回** 0**。**

分析：先从键盘上输入一个偶数 n(＞=6)**，将它表示为** i **与** j **之和：**

n=i+j;

当 i **和** j **都为素数时，则为所求。**

程序内容如下：

```
#include<iostream.h>
void main( )
{
    int prime(int);
    int n;
    cout<<?输入一个大于等于 6 的偶数: ?;
    cin>>n;
    int i,j;
    cout<<?两个素数之和的所有可能:\n?;
   {
    for(i=2;i<n/2;i++)
        j=n−i;
            if(prime(i)&&prime(j))
                cout<<"N="<<i<<'+'<<j<<endl;
    }
}
int prime(int n)
{
    int j=2;
    while(n%j!=0)
        j++;
    if(n= =j)
        return 1;
    else
       return 0;
}
```

2.6 第 6 章上机练习指导

上机目的要求

1. 熟悉指针的概念和应用，了解指针的值和类型。
2. 掌握指针的定义格式和赋值方法以及指针的运算。
3. 学会使用指针表示数组元素的方法和指针作函数参数及返回值的用法。
4. 熟悉引用的概念，掌握引用的定义方法，学会引用在 C++程序中的应用。

上机练习题

1. 上机调试本章例 6.2 程序，并将运行结果与书中给出的结果比较。

通过该例搞清楚如下问题：

指针运算实质上是地址运算，但是，指针运算与地址运算又是不同的。指针运算中加 1，不是地址值加 1，而是地址值加上该指针所指向的变量在内存中所占的字节数。例如，在 32 位机中，指向 int 型数组的元素指针加 1，实际上是地址值加 4。另外，指针运算与地址运算的表示是不同的。地址运算时，要在指针名前加上(int)，即进行类型强制。

该例题中，还可看到关于指针的定义、赋初值和赋值，还有一个指针加 1 的运算以及指针的输出显示。

2. 上机调试本章例 6.4 程序，进而分析二维数组的各种表示方法。

通过该程序中对于各种不同表示的地址值和元素值的输出显示，进一步搞清二维数组的各种地址和元素值的表示方法。

二维数组的地址值常用的有两种：

一种是二级指针的地址值，即为行地址值。该例中 a,a+1,分别是第 0 行和第 1 行的行地址值，它们的大小分别与第 0 行和第 1 行的首列元素地址值相等。

另一种是一级指针的地址值，即为元素地址值，该例中，*a,a[0],a[1],*(a+0),*(a+1)等都是某一行首列元素的地址值，其中，*a,a[0]和*(a+0)为第 0 行首列元素的地址值，a[1]和*(a+1)是第 1 行首列元素的地址值。

该例中，&a[0]和&a[1]分别与 a 和 a+1 是等价的，即表示该数组第 0 行和第 1 行的地址值。

二维数组的元素值表示方法有如下几种：

(1) 二级指针表示，该例中有**a,**(a+1),*(*(a+1)+1)，它们分别表示为该数组的首元素值，第 1 行第 0 列元素值和第 1 行第 1 列元素值。

(2) 下标表示，该例中 a[1][1]表示第 1 行第 1 列元素值。

(3) 指针下标混合表示，该例中，(*(a+1))[0]和*(a[1]+1)分别表示该数组的第 1 行第 0 列元素值和第 1 行第 1 列的元素值。

3. 上机调试本章例 6.8 程序。

该程序中定义了一个指向一维数组的指针 p。其指向具有 4 个 int 型元素的一维数组。通常，对于指向一维数组的指针是用一个列数与它所指向的一维数组大小相同的二维数组的行地址给它赋值。在该例中，用 a+1 给 p 赋值。a+1 是二维数组 a 的第 1 行的行地址，它们都

是二级指针的地址值，即让 p 指针指向二维数组 a 的第 1 行。于是可用指针 p 来表示它所指向的数组 a 的各个元素。*(*p+j)表示数组 a 的第 1 行的各列元素，其中 j 为 0 至 3；*(*(p+1)+j)表示数组 a 的第 2 行的各列元素，j 同前；*(*(p−1)+j)表示数组 a 的第 0 行的各列元素，j 同前。

练习题

请读者将该例中 main( )函数内语句“ p=a+1 ;”改为“ p=a ;”，其余不变，上机调试输出结果，比较与修改前有何不同？为什么？为使输出结果与原来的保持一样，需对输出语句中的哪些表达式进行修改？如何修改才能保证输出结果与原来一样？

4. 上机调试本章例 6.9 程序，并验证输出结果的正确性。

该程序中，定义了一个字符型的指针数组 name，并给它赋了初值。使用该数组存放了 13 个字符串。该数组的大小为 13。

字符型指针数组用来存放若干个字符串是十分方便的，它比用二维字符数组存放字符串节省内存空间。一个一维的一级字符型指针数组相当于一个二维的字符型数组。一维指针数组的大小便是存放字符串的个数。在本例中，name[i]或者*(name+i)(i 为 0～12)表示了每个字符串的起始地址。

请读者将*name[13]改写为一个二维的字符型数组，并用字符串给它赋初值，其余不变，输出该程序结果。

5. 上机调试本章例 6.13 程序，并将调试后的输出结果与教材中给出的结果进行比较。

通过调试该程序回答下列各问题：

(1) 在定义引用时，如果不给它赋初值，而在定义后给它赋值是否可以？即将语句

```
int &refv=val1;
```

改写为

```
int &refv;
refv=val1
```

(2) 修改该程序验证：引用值被改变了，则被引用的变量值也同样改变。

(3) 一个引用被定义后，能否可以改变被引用的值？能否改变被引用的变量？

从该例程序中，进一步掌握引用的概念。

6. 上机调试本章中例 6.16 程序，并通过键盘输入验证该程序的正确性。

该程序的主要目的是用来验证：引用可以是函数的返回值，并且返回引用的调用函数可为左值。该程序中，通过下列语句

```
fun(ch,tn,tc)++；
```

来验证了这一点。

由于引用是变量的别名。该例中，fun( )函数的两个形参 n 和 c 都是引用，当该函数被调用后，n 和 c 分别是变量 tn 和 tc 的别名，当函数返回值为 n 时，对它进行增 1 操作，实际上就是对于 tn 进行增 1 操作；同样地，对 c 增 1 操作，就是对 tc 进行增 1 操作。

7. 本章练习题 6.4 第 2 题，上机验证其输出结果与分析结果是否相同。

该程序中使用了字符指针，并用它作为函数 fun( )的形参。函数 fun( )实际上是一个字符串比较函数。使用该函数对下列的字符串运行后，分析并验证输出结果。

① p1="abc"; p2="abc";

② p1="ab"; p2="abc";

③ p1="abc"; p2="ab";

④ p1="abci"; p2="abck";

⑤ p1="abcij"; p2="abcief";

从输出结果中可看出该程序中还存在什么问题？如果作为字符串比较函数，还应作如何修改？

在 while **循环的条件表达式中，***s1 &&*s2 **的作用是什么？去掉它们是否对结果没有影响？**

8. **本章练习题** 6.4 **中第** 4 **题，上机验证输出结果与分析结果。**

结合本例搞清指针的级别。在下列地址值中，哪些是属于一级指针的地址：哪些是属于二级指针的地址？

① ba ② a[1] ③ a+1 ④ a[0]+1 ⑤ p ⑥ *(p+1)

⑦ p[0] ⑧ pp ⑨ *pp ⑩ s ⑪ *s ⑫ q

该例中，p,pp,s **和** q **都与二维数组** a **有关系，准确描述出它们之间的关系。进而熟悉一个二维数组元素的各种表示方法。**

9. **本章练习题** 6.4 **中第** 8 **题，结合上机调试掌握引用的概念及其用途。**

在本章练习题 6.4 **中第** 6，7 **题中已出现过引用，并且给出了引用的定义方法、引用与被引用变量的关系以及引用用作函数的形参，而本例程序是将引用作为函数的返回值。**

该例程序中，函数 fun( )**是一个返回值为引用的函数，该函数将返回的是一个一维数组** a **的某个元素。因此，通过给该函数的返回值赋值，而达到给一个数组的各个元素赋值的目的。**

请读者记住：通常，函数调用可以作为赋值表示式的右值；而返回值为引用的函数调用，可作为赋值表达式的左值。

10. **编程练习。**

本章练习题 6.5 **中各小题。**

(1) **有** 3 **个已知的字符串，按由大到小的顺序输出。**

分析：已知三个字符串分别存放于三个字符数组中，并规定每个字符数组都可以放下三个字符串中最长的字符串，免得在实现字符串处理时越界。

先将第一个字符串与第二个字符串进行比较，将大者放在第一个字符串中；再将第一个字符串与第三个字符串进行比较，将大者放在第一个字符串中；最后将第二个字符串与第三个字符串进行比较，确定第二大者。

程序内容如下：

```
#include<iostream.h>
#include<string.h>
void swap(char p1[ ],char p2[ ]);
void main( )
{
    char s1[10]="one",s2[10]="two",s3[10]="three";
    if(strcmp(s1,s2)<0)
        swap(s1,s2);
    if(strcmp(s1,s3)<0)
```

```
            swap(s1,s3);
        if(strcmp(s2,s3)<0)
            swap(s2,s3);
        cout<<s1<<'\n'<<s2<<'\n'<<s3<<'\n';
}
void swap(char p1[ ],char p2[ ])
{
        char p[10];
        strcpy(p,p1);
        strcpy(p1,p2);
        strcpy(p2,p);
}
```

(2) **将** 10 **个不等长的字符串放在一个指针数组中，对它实现如下操作。**

a. **查找某个字符串**

b. **修改某个字符串**

c. **删除某个字符串**

d. **复制某个字符串**

e. **排序** 10 **个字符串**

分析：用一个字符型指针数组存放 10 **个已知的字符串，根据题目中要求的操作实现** 5 **种功能。**

程序内容如下：

```
#include<iostream.h>
#include  <string.h>
char *lookup_str(int i);    //查找函数
char *modifi_str(int i,char s[ ]);    //修改函数
void delete_str(int i);    //删除函数
char *copy_str(int i);    //复制函数
void sort_str( );    //排序函数
char *str[10]={"for","int","while","switch","case","default",
            "break","continue","goto","return"};
void main( )
{
        //查找操作
        int n;
        char s[80];
        cout<<"查找第几个串:";
        cin>>n;
        cout<<"该串是 "<<lookup_str(n)<<endl;
        //修改操作
```

```
        cout<<"修改第几个串:";
        cin>>n;
        cout<<"该串修改为:";
        cin>>s;
        cout<<"该串修改后为 "<<modifi_str(n,s)<<endl;
        //删除操作
        cout<<"删除第几个串:";
        cin>>n;
        delete_str(n);
        cout<<"第"<<n<<"个串被删除了\n";
        //复制操作
        cout<<"复制第几个串:";
        cin>>n;
        char *ss=copy_str(n);
        cout<<"第"<<n<<"个串被复制了\n";
        //排序操作
        sort_str( );
        cout<<"排序后:\n";
        for(int i=0;i<10;i++)
                cout<<*(str+i)<<"    ";
        cout<<endl;
}
char *lookup_str(int i)
{
        return *(str+i-1);
}
char *modifi_str(int i,char s[ ])
{
      return *(str+i-1)=s;
}
void delete_str(int i)
{
        *(str+i-1)=" ";
}
char *copy_str(int i)
{
        return *(str+i-1);
}
void sort_str( )
{
```

```
    for(int i=0;i<10;i++)
        for(int j=0;j<10;j++)
            if(strcmp(*(str+i),*(str+j))>0)
            {
                char *p=*(str+i);
                *(str+i)=*(str+j);
                *(str+j)=p;
            }
}
```

(3) **将一个长度为** n **个字符的字符串，用函数实现其逆序输出。**

分析：编写一个函数，用来对已知字符串进行逆序排列输出。可以用下述方法：先求该字符串的长度，再求其长度的一半，将前后两半字符串对应字符进行交换，即首字符与最后一个字符交换，第二个字符与最后字符的前一个字符交换……直到交换一遍。

程序内容如下：

```
#include<iostream.h>
#include<string.h>
void main( )
{
    void inverse(char *);
    char s[50];
    cout<<"输入一个字符串:";
    cin>>s;
    inverse(s);
    cout<<"反序后的字符串:" <<s<<endl;
}
void inverse(char *p)
{
    int n =strlen(p);
    int limit=n/2;
    for(int i=0;i<limit;i++)
    {
        char t=*(p+i);
        *(p+i)=*(p+n-i);
        *(p+n-i)=t;
        n--;
    }
}
```

(4) **有** n **个小孩按顺序排成一圈。从第** 1 **个小孩开始作** 1 **至** 3 **报数，凡报数为** 3 **的小孩从圈中出来，求最后出圈的小孩的顺序号是多少？**

分析：将 n 个小孩排成一圈，并从 1 到 n 的编号。从第 1 个小孩报数 1 至 3，报数为 3 的小孩出圈，其标志为编号置 0。在 1 至 3 报数时，要判断报数的小孩编号为非 0。当编号为 0 时，到下一个小孩。当前一个小孩编号为 n 时，下一个小孩编号要么为 0，要么为(n+1)。

程序内容如下：

```
#include<iostream.h>
void main( )
{
    int n,num[100];
    cout<<"输入排号人数:";
    cin>>n;
    int *p=num;
    for(int i=0;i<n;i++)
        *(p+i)=i+1;
    i=0;
    int k=0;   //k 是 1 至 3 报数的计数变量
    int a=0;   //a 为退出圈子的人数
    while(a<n-1)
    {
        if(*(p+i)!=0)
            k++;
        if(k= =3)
        {
            *(p+i)=0;   //退出圈子的人的编号被设为 0
            k=0;
            a++;
        }
        i++;
        if(i= =n)
            i=0;
    }
    while(*p= =0)
        p++;
    cout<<"最后留下的第 "<<*p<<" 号"<<endl;
}
```

执行该程序后，输入如下信息：

输入排号人数：100↵

最后留下的第 91 号

(5) **编程实现两个字符串的交换。**

例如,

char *p1=?hello?；

char *p2=?good?；

使用引用作函数参数，交换后为

p1: "good?

p2: "hello?

分析：对指针的引用可说明如下：

int *p

int *&pref=p；

这里，pref **是一个对指针** p **的引用。**

在该程序中，编写了一个函数。该函数的形参是两个对于指针的引用，在该函数体内对这两个引用进行交换，相当于对调用函数中两个实参的指针进行了交换。

程序内容如下：

```
#include<iostream.h>
void main( )
{
    void swap(char * &,char * &);
    char *s1="hello";
    char *s2="good";
    swap(s1,s2);
    cout<<s1<<'\n'<<s2<<endl;
}
void swap(char * &p1,char * &p2)
{
    char *p;
    p=p1;
    p1=p2;
    p2=p;
}
```

2.7 第 7 章上机练习指导

上机目的要求

1. **熟悉结构变量的定义、结构成员的表示、结构变量与指向结构变量指针的常用赋初值及赋值的方法。**

2. **掌握结构在数组和函数方面的用法。通过上机练习熟悉结构成员可以是数组，数组元素又可以是结构变量；并熟悉结构变量和指向结构变量的指针作函数参数，熟悉函数返回值**

在程序中的应用。

3. 了解联合与结构两种构造数据类型的区别。熟悉联合变量的定义、赋值和使用方法。

上机练习题

1. 上机调试本章中例 7.3 程序，并验证其输出结果。

将该程序按下列要求进行修改，然后上机调试。

(1) 将程序中，assign_value( ),extract_value( )和 print_value( )三个函数中形参 struct card *c_ptr 改为结构变量，其他程序作相应修改后，上机调试，并输出相同结果，说明用结构变量和用指向结构变量的指针作函数参数的不同。

(2) 原程序中使用函数 assign_value( )给 52 张扑克牌赋值，请将其改为用初始值表给结构数组 card[52]进行初始化。修改后调试，并获得相同输出结果。比较两种方法的不同。

2. 上机调试本章例 7.7 程序，并体会结构变量在处理不同类型事物中的应用。

该例中定义了一个描述学生成绩的结构模式，其结构名为 student。该结构成员有不同类型的变量：字符指针和 double 型数组。这种描述使用结构最方便，多个学生的描述将采用结构数组。

请将该程序作如下修改后，再调试。

(1) 修改为通过键盘输入 5 个学生的成绩。

(2) 该程序中的查找函数 find( )的返回值是指向结构变量的指针，能否改成查找函数返回值为结构变量呢？如何修改？比较这两种方法哪一种好？

(3) 请修改该程序，增加一个查找学生最高分(总分)的函数；并查找最高分的学生，输出姓名及最高分数。

3. 上机调试本章例 7.10 中程序，并搞清楚几个问题。

(1) 结构中可以包含联合变量的成员，而联合中又可以包含有结构变量的成员。它们之间在使用上是可以相互嵌套的。

(2) 结构 student 中，成员“char off_in;”实际上是一种标识。请试一下，这种标识可应用枚举量给出？

(3) 在该程序中，增加一个查找学生地址的函数。当给定姓名时，请输出该学生的地址。

4. 编程练习。

本章练习题 7.5 中给出两个编程题，下面给出的程序仅供参考。

(1) 使用一个含有年、月、日成员的结构变的，指定某个日期后，计算该日在本年中是第几天。考虑闰年问题。

程序内容如下：

```
#include<iostream.h>
struct date
{
    int day,month,year;
};
int month_days[ ]={0,31,28,31,30,31,30,31,31,30,31,30,31};

void main( )
```

```
{
    struct date date1;
    int days=0;
    cout<<"Input date: ";
    cin>>date1.year>>date1.month>>date1.day;
    for(int i=1;i<date1.month;i++)
        days+=month_days[i];
    if((date1.month>2)&&((date1.year%4= =0)&&(date1.year%100!=0)
                        ||(date1.year%400= =0)))
        days++;
    cout<<"Year="<<date1.year<<"    Month="<<date1.month<<
        " Day="<<date1.day<<"    No.="<<days+date1.day<<endl;
}
```

(2) **某书店出售下列计算机书籍，其书名、数量和单价如下所示：**

Pascal	40	21.5
FoxPro	50	20.0

按下列要求编程：

a. **显示各种书名、数量和单价**
b. **按书名查找某种书籍的数量**
c. **给定书名和数量计算应付的金额**
d. **增添新的书名数量和单价。**

例如，

Office	60	24.0

程序内容如下：

```
#include<iostream.h>
#include<string.h>
struct book
{
    char name[10];
    int quantity;
    double unit_price;
};
struct book books[10]={{"Pascal",40,21.5},{"FoxPro",50,20.0},
                       {"C",80,25.0}};
void main( )
{
    cout<<"显示各种书名: \n";
    for(int i=0;i<3;i++)
        cout<<books[i].name<<', '<<books[i].quantity<<', '
```

```
                                        <<books[i].unit_price<<endl;
    cout<<"\n 输入书名:\n";
    char book1[10];
    cin>>book1;
    for(i=0;i<3;i++)
        if(strcmp(book1,books[i].name)= =0)
            cout<<"该书数量为:    "<<books[i].quantity;
    cout<<endl<<endl;
    cout<<"输入书名和购量: \n";
    int number;
    cin>>book1>>number;
    for(i=0;i<3;i++)
        if(strcmp(book1,books[i].name)= =0)
            cout<<"应付金额为: "<<number*books[i].unit_price;
    cout<<endl<<endl;
    int n=2;
    cout<<"增添新书(书名  数量  单价):\n";
    n++;
    cin>>books[n].name>>books[n].quantity>>books[n].unit_price;
    cout<<books[n].name;
    cout<<endl;
}
```

请上机调试该程序，如果感到该程序中有些操作不方便，例如，想要增添多本新书等，请读者自行修改。

2.8 第 8 章上机练习指导

上机目的要求

1. **熟悉类的定义格式和类中成员的访问权限。**
2. **掌握对象的定义方法以及对象的初始化和赋值的操作。**
3. **了解成员函数的特性、静态成员、友元、类的作用域及对象生存期等概念。**

上机练习题

1. **上机调试本章中的例** 8.4 **程序，验证书中给出的结果是否正确。**

在该程序中，将 TPoint **类的带有两个参数的构造函数进行修改，在函数体内增添下述语句：**

```
cout<<"Constructor Called.\n";
```

按下列要求进行调试，进而熟悉对象初始化的各种方法：

(1) **在主函数体内，增加下述说明语句：**

TPiont P4=TPoint(3,5)；

这是通过强制类型方法对创建的对象 P4 **初始化，并添加语句输出显示对象** P4 **的成员值。**

(2) **在函数体内，添加下列说明语句：**

TPiont P5=(1,8);

该语句与下述语句等价；

TPiont P5(1,8);

(3) **在函数体内，添加下列说明语句：**

TPiont P6,P7(2);

调试程序会出现什么现象？为什么？如何解决？(对已有的构造函数进行适当修改)。

练习题

总结对象初始化的方法。

2. **上机调试下述程序，并对输出结果给予解释。**

程序内容如下：

```
#include<iostream.h>
class A
{
   public:
      A(int i=0)
      { a=i; cout<<"Constructor "<<a<<endl; }
      ~A( )
      { cout<<"Destructor "<<a<<endl; }
      void print( )
      { cout<<a<<endl; }
   private:
      int a;
};
void main( )
{
      A a1(1),a2(2);
      a1=5;
      a1.print( );
      a2=A(10);
      a2.print( );
}
```

执行该程序输出结果如下：

```
Constructor 1
Constructor 2
Constructor 5
```

```
Destructor 5
5
Contructor 10
Destructor 10
10
Destructor 10
Destructor 5
```

程序分析：

该程序的类 A **中，定义一个带有一个缺省参数的构造函数和一个析构函数，它们中都有输出显示信息。**

在主函数中，首先调用带有一个缺省参数的构造函数创建两个对象 a1 **和** a2**，对应屏幕上显示两行信息。接着，执行下述语句**

```
a1=5;
```

这时，调用一个参数的构造函数将由数值 5 **创建一个无名对象(又称临时对象)，并将它赋值给** a1**，这时无名对象生存期结束并被析构。于是，屏幕上又出现下述两条信息：**

```
Constructor 5
Destructor 5
```

执行语句“ a1.print( );**”后，输出显示** 5**。**

再执行下列语句：

```
a2=A(10);
```

调用带一个参数的构造函数，将数值 10 **强制成为类** A **对象。该对象为无名对象，再将它赋值给** a2 **后，将它析构。于是又出现下述两条信息：**

```
Constructor 10
Destructor 10
```

执行语句“ a2.print( );**”后，输出显示** 10**。**

退出程序前，将前面创建的两个对象 a1 **和** a2 **析构。请注意，析构对象的顺序与创建对象的顺序相反，即先析构对象** a2**，它的成员** a **的值为** 10**；再析构对象** a1**，它的成员** a **的值为** 5**。**

在本例中，讲述了在给对象赋值时，当右值表达式与左值类型不同时，在上述情况下，可以通过调用构造函数创建无名对象来实现。而在强制一个数值为该类的类型时，也需调用构造函数创建无名对象来实现。

3. **上机调试本章例** 8.5 **中的程序，并分析该程序的输出结果。说明拷贝初始化构造函数的用途。**

搞清楚下列问题是对理解该程序有帮助的。该程序中，共调用了几次构造函数(两个参数的)？共调用几次拷贝构造函数(即拷贝初始化构造函数)？共调用几次析构函数？析构函数何时被调用？

上述问题的回答如下：

(1) **共调用** 4 **次构造函数和** 3 **次拷贝构造函数，共创建** 7 **个对象。其中主函数中** 4 **个，被调用的** fun( )**函数中** 1 **个，还有两临时对象：一个是函数调用时，用实参对象对形参初始化；**

另一个在函数返回值时创建一个临时对象。

(2) 创建 7 个对象，则需要调用 7 次析构函数。当一个对象的生存期结束时，系统将自动调用析构函数来析构该对象。

另外，为搞清楚调用构造函数的次数，可将该程序中，构造函数 TPiont(int x,int y)的主函数体内添加一条输出信息的语句。

4. 上机调试本章中例 8.10 的程序，掌握静态成员在该程序中的使用。

具体要求如下：

(1) 如何定义静态数据成员和成员函数？

(2) 如何对静态数据成员初始化？

(3) 静态成员函数中出现的静态成员与非静态成员有何区别？

(4) 如何调用静态成员函数？

(5) 如何理解“静态成员不是属于某个对象的，而是属于类的所有对象的。”这句话？

上述问题请读者通过调试该程序进一步体会。

5. 上机调试下列程序，通过调试输出结果，分析回答几个问题。

(1) 在对象创建之前就存在了静态数据成员，对吗？

(2) 静态成员函数中可以直接引用静态成员，而不能直接引用非静态成员，对吗？

(3) 一般成员函数中可以直接引用静态成员和非静态成员，对吗？

(4) 在静态成员函数中，引用非静态成员时要通过对象，对吗？通过成员限定可否？

(5) 构造函数是否可以对类中静态的数据成员进行初始化？被初始化的静态数据成员是保持该值到下一次被改变时为止？

上述问题都需要通过该程序进行验证所回答的结果。

程序内容如下：

```
#include<iostream.h>
class S
{
    public:
        S(int i,int j)
        { S::s=i;n=j; }
        S(int i)
        { n=i; }
        static int gets( )
        { return s; }
        int getsn( )
        { return n+s; }
        static int getn(S &m)
        { return m.n; }
    private:
        int n;
        static int s;
};
```

```
int S::s=5;
void main( )
{
    cout<<"S::gets( )="<<S::gets( )<<endl<<endl;
    S a(12,36);
    cout<<"("<<a.gets( )<<","<<a.getn(a)<<")"<<endl;
    cout<<"n+s="<<a.getsn( )<<endl<<endl;
    S b(45,21);
    cout<<"("<<b.gets( )<<","<<b.getn(b)<<")"<<endl;
    cout<<"n+s="<<b.getsn( )<<endl<<endl;
    S c(10);
    cout<<"("<<c.gets( )<<","<<c.getn(c)<<")"<<endl;
    cout<<"n+s="<<c.getsn( )<<endl;
}
```

6. **上机调试本章例** 8.13 **中程序，分析不同存储类对象的创建和析构情况。**

该程序中共有 5 **个不同存储类的对象，其中，外部类对象** A3**，外部静态类对象** A4**，自动类对象** A1 **和** A5**，内部静态类对象** A2**。**

不同存储类对象具有不同的作用域和生存期。一个对象被析构的时间取决于该对象的生存期，当一个对象的生存期结束时，该对象将被析构。

7. **上机调试本章练习题** 8.4 **中第** 5 **题。**

该程序中定义一个类 Set**。该类中的数据成员有一个** int **型数组** elems[100]**，还有标识数组中元素个数的** int **型变量** PC**。**

该类中定义了对 int **型数组的操作如下：**

(1) **使用默认构造函数创建一个对象，使该对象的数组成员为空的。**

(2) **使用拷贝初始化构造函数复制一个已知对象的数组。**

(3) **使用成员函数** Empty( )**可将一个对象的数组成员置空。**

(4) **使用成员函数** IsEmpty( )**来判断一个对象的数组成员是否为空。**

(5) **使用成员函数** Add( )**可向某对象的数组成员中增加元素。**

(6) **使用成员函数** IsMemberOf( )**来判断一个** int **型数是否是某个对象的数组成员中的一个元素。**

(7) **使用成员函数** Print( )**可将某个对象的数组成员的各个元素值输出显示。**

(8) **使用成员函数** reverse( )**可将某个对象的数组成员中的各个元素值反序排列。**

上机调试出上述 8 **个功能后，再请读者增加下述两个功能：**

(1) **从对象的数组成员中删除数值等于某个已知数值的元素。**

(2) **将一个对象的数组成员的元素添加到另个对象的数组成员的元素后面，组成一个新对象的数组成员。**

8. **编程练习。**

本章练习题的 8.5 **题。在一个程序中，实现下述要求：**

a. **构造函数重载**

b. **成员函数设置默认参数**

c. **有一个友元函数**

d. **使用不同的构造函数创建不同的对象**

程序内容如下：

```
#include<iostream.h>
#include<math.h>
class Myclass
{
   public:
        Myclass(double i=0)
        { x=y=i;}
        Myclass(double i,double j)
        { x=i; y=j; }
        Myclass(Myclass &m)
        { x=m.x; y=m.y; }
        friend double distance(Myclass &a,Myclass &b);
   private:
        double x,y;
};
double distance(Myclass &a,Myclass &b)
{
        double dx=a.x−b.x;
        double dy=a.y−b.y;
        return sqrt(dx*dx+dy*dy);
}

void main( )
{
        Myclass m1,m2(5),m3(3,4);
        Myclass m4(m3);
        cout<<"The distance1: "<<distance(m1,m3)<<endl;
        cout<<"The distance2: "<<distance(m2,m3)<<endl;
        cout<<"The distance3: "<<distance(m3,m4)<<endl;
        cout<<"The distance4: "<<distance(m1,m2)<<endl;
}
```

该程序基本满足上述要求，调试后输出结果，并检查该结果是否正确。

2.9 第9章上机练习指导

上机目的要求

1. **通过上机练习搞清下述对象的定义格式、赋值方法以及引用方法：**

(1) **对象指针** (2) **对象引用** (3) **对象数组**

(4) **对象指针数组** (5) **常对象** (6) **子对象**

(7) **堆对象**

2. **搞清下述概念及其程序中的用法：**

(1) **指向类成员的指针：指向类数据成员的指针和指向类成员函数的指针。**

(2) **类的常成员：包含类的常数据成员和类的常成员函数。**

(3) **类型转换函数。**

上机练习题

1. **上机调试本章例** 9.1 **的程序。**

通过调试该程序，回答下列问题：

(1) **如何定义一个指向某类的数据成员的指针？该例中，**pc **就是指向类** A **的某数据成员的指针。**

(2) **如何给指向类的数据成员指针赋值？该例中，“** pc=& A::c; **”就是将类** A **中公有成员** c **的地址值赋给** pc**，使** pc **指向类** A **的数据成员** c**。**

(3) **如何定义一个指向某类的成员函数的指针？该例中，**pfun **就是指向类** A **的某成员函数的指针，该函数有一个参数是** int **型的，该函数返回值也是** int **型的。**

(4) **如何给指向类的成员函数指针赋值？该例中，“** pfun=A::fun; **”就是将类** A **中公有成员函数** fun( )**的函数名赋给** pfun**，使** pfun **指向类** A **的成员函数** fun( )**。**

(5) **如何在程序中使用指向类的成员的指针？该问题请读者结合本例回答。**

2. **上机调试本章例** 9.3 **的程序。**

调试该程序应掌握使用对象引用作函数参数，实现引用调用的方法。

该例程序中，有两处使用对象引用作函数形参，一个是类 M **的成员函数** copy( )**中使用对象引用作形参，另一个是一般函数** fun( )**中使用对象引用作形参。当函数形参为对象引用时，其实参使用对象名；所实现的函数调用与使用指针的传址调用具有相同功能和特点。另外，对象引用的成员表示与对象的成员表示相同。**

3. **上机调试本章例** 9.5 **程序。**

先将该程序输入，调试后得到输出结果与书中给出的结果进行比较。

在该程序的 M **类两个构造函数的函数体中，都加上相应的输出显示信息的语句；再定义一个析构函数，该函数体内只有一条输出显示信息的语句；调试后分析新的输出结果。这时，将会发现该程序执行过程中，先后调用了** 9 **次构造函数和** 9 **次析构函数，其中调用默认构造函数** 3 **次，调用两个参数的构造函数** 6 **次。仔细分析可知，在执行下列语句**

```
mm1[1]=M(5,9);
```

时，先调用两个参数的构造函数创建一个临时对象，将该对象值赋给 mm1[1]**后，它将被析构。**

同样，执行下列语句

```
mm1[2]=M(2,7);
```

时，也是如此。

该例程序中，定义了两个对象数组，一个是外部类的 mm1，另一个是自动类的 mm2。其中，mm1 有默认值，mm2 被初始化，此时都需调用相应的构造函数。在给数组元素赋值的两条语句中，自动创建了临时对象，其中 M(5,9)可看作是强制类型，即将(5,9)强制成为类 M 的对象，这时需调用相应的构造函数。

4. 上机调试下列程序，学习和掌握对象指针数组的定义、赋初值、赋值以及使用的方法。

程序内容如下：

```
#include<iostream.h>
class T
{
   public:
      T( )
      { i=0; f=0.0; cout<<"Constructor(0)\n"; }
      T(int n)
      { i=n; f=0.0; cout<<"Constructor(1)..."  <<i<<endl; }
      T(int n,double d)
      { i=n; f=d; cout<<"Constructor(2)..." <<i<<','<<f<<endl; }
      ~T( )
      { cout<<"Destructor..." <<i<<','<<f<<endl; }
      int Geti( )
      { return i; }
      double Getf( )
      { return f; }
   private:
      int i;
      double f;
};

void main( )
{
      T *p0=new T[2];
      T *p1[ ]={new T( ),new T( )};
      T *p2[ ]={new T(2),new T(6)};
      T *p3[ ]={new T(6,8.9),new T(4,2.3)};
      T *p4[2];
      p4[0]=new T(2,1.5);
      p4[1]=new T(5,2.4);
      for(int i=0;i<2;i++)
```

```
        cout<<"p4["<<i<<"]=("<<p4[i]->Geti( )<< ', '<<p4[i]->Getf( )<< ') '<<endl;
    delete [ ]p0;
    for(i=0;i<2;i++)
    {
        delete p1[i];
        delete p2[i];
        delete p3[i];
        delete p4[i];
    }
}
```

上机调试该程序后，分析该程序的输出结果。

该程序中先创建一个动态的对象数组 p0**，然后创建** 4 **个对象指针数组** P1**，**P2**，**P3 **和** P4**，其中** P1**，**P2 **和** P3 **进行了初始化，**p4 **被赋了值。程序退出前使用运算符** delete **释放动态对象数组和对象指针数组。**

5. **上机调试下列程序，该程序主要练习常类型在程序中的各种用法。**

(1) **一般变量名前加** const**，例如**

```
const int x=a1.fun1( );
```

其中，x **是一般的常整型量。**

(2) **关于常指针分三种情况：**

① **在“** const A *p1=&a1; **”中，**p1 **是一个指针，该指针所指向的对象是常量，但是，**p1 **的地址值是可以改变的。**

② **在“** A *const p2=&a1; **”中，**p2 **本身是一个常量，而它所指向的对象是可以改变的。**

③ **在“** const A *const p3=&a1; **”中，**p3 **本身是一个常量，而它所指向的对象也是一个常量。**

(3) **常量对象是指某个对象是一个常量。本例中，**const A a2(5); **其中对象** a2 **是一个常量对象。**

(4) **常成员函数是指将** const **关键字写在函数头之后，函数体之前的那种函数。例如，在该例语句“** const int fun1( ) const { return a；} **”中，函数** fun1( )**是常成员函数，该函数的返回值是常** int **型数。**

常成员函数是指该函数的 this **指针所指向的是常对象。**

程序内容如下：

```
#include<iostream.h>
class A
{
  public:
    A(int i)
    { a=i; }
    const int fun1( ) const
    { return a; }
```

```
        int fun2( )
        { return a*a; }
    private:
        int a;
};

void main( )
{
        A a1(8);
        const int x=a1.fun1( );
        int y=a1.fun2( );
        cout<<x<<','<<y<<endl;
        const A a2(5);
        int x1=a2.fun1( );
//      int y1=a2.fun2( );
        cout<<x1<<endl;
        const A *p1=&a1;
        p1=&a2;
//      *p1=a2;
        cout<<p1->fun1( )<<endl;
        A * const p2=&a1;
       *p2=a2;
//      p2=&a2;
        cout<<p2->fun1( )<<endl;
        const A * const p3=&a1;
//      p3=&a2;
//      *p3=a2;
        cout<<p3->fun1( )<<endl;
}
```

调试该程序，将得到一个输出结果，请读者自行分析。

该程序中，被注释的语句都是调试时有错误信息的语句，即有语法错误的语句。下面分析这些语句的错误原因。

(1) **由于对象** a2 **是常对象，**fun1( )**函数是常成员函数，**a2.fun1( )**是可以的，而** a2.fun2( ) **是错误的，即常对象不能调用非常成员函数；而非常对象可以调用常成员函数或非常成员函数，即** a1.fun1( )**和** a1.fun2( )**都是合法的。**

(2) **程序中** p1 **是一个常对象指针，因此，“** *p1=a2; **”是非法的,而“** p1=&a2; **”是合法的。**

(3) **程序中** p2 **是一个常地址指针，因此，“** p2=&a2; **”是非法的，而“** *p2=a2; **”是合法的。**

(4) **程序中** p3 **是一个常对象常地址指针，因此，“** p3=&a2; **”和“** *p3=a2; **”都是非法的。**

6. **上机调试下述程序，该程序出现如下成员，这些成员的初始化应放在成员初始化表中进行。**

(1) **子对象**

(2) **引用**

(3) **常类型变量**

程序内容如下：

```
#include<iostream.h>
class M
{
    public:
        M(int i,int j)
        { m1=i;m2=j; cout<<"Constructor...M\n"; }
        void print( )
        { cout<<m1<<','<<m2<<endl; }
    private:
        int m1,m2;
};
class N
{
    public:
        N(int i,int j,int k,int l):m(i,j),a(k),r(n)
        { n=l; cout<<"Constructor...N\n"; }
        void print( )
        {
                m.print( );
                cout<<n<<','<<r<<','<<a<<endl;
        }
    private:
        M m;
        const int a;
        int &r;
        int n;
};

void main( )
{
        M x1(7,8);
        x1.print( );
        N y1(3,4,5,6);
        y1.print( );
```

```
}
```

调试该程序，得到输出结果。

在该程序中，请注意类 N 的构造函数的成员初始化表里有三项，它们分别给该类的数据成员 m(子对象)，a(常类型变量)和 r(引用)进行初始化。

请读者试一下，将对数据成员 n 的初始化也放到成员初始化表中是否可以？

7. 上机调试本章例 9.14 程序。

调试中注意使用运算符 new 创建动态对象数组和用运算符 delete 释放动态数组的方法。下列语句是用来创建动态数组的语句：

```
B *pb=new B[3];
```

其中，pb 是一个指向动态对象数组的指针，该动态数组有 3 个元素，每个元素是类 B 的对象，创建时要调用默认的构造函数。

接下来的 3 条语句是给动态对象数组的 3 个元素赋值。赋值时，将一组数据强制成类 B，这时调用相应的构造函数创建临时对象，将该临时对象值赋给对应的对象数组元素后，临时对象被析构。

8. 上机调试下列程序。

程序内容如下：

```
#include<iostream.h>
class X
{
   public:
      X(int i,double j)
      { x=i; y=j;}
      operator int( );
      operator double( );
   private:
      int x;
      double y;
};
X::operator int( )
{
      return x*x;
}
X::operator double( )
{
      return x+y;
}

void main( )
{
```

```
    X n(8,2.5);
    int a=n;
    double b=n;
    cout<<a<<','<<b<<endl;
    cout<<int(n)<<','<<double(n)<<endl;
}
```

该程序类 X **中说明了两个类型转换函数，它们是：**

```
operator int( );
operator double( );
```

类型转换函数的特点是：一不带参数；二不能指定返回类型。返回类型实际是关键字 operator **后面跟的类型，本例中分别是** int **型和** double **型。类型转换函数是成员函数。习惯上，将** operator int **看成是函数名。在该程序中，**

```
X n(8,2.5);
```

创建了一个 X **类的对象** n**。**

```
int a=n;
```

可以被解释为

```
int a=n.operator int( );
```

执行后，a **被赋值为** 64**。**

同样，

```
double b=n;
```

可以被解释为

```
double b=n.operator double( );
```

执行后，b **被赋值为** 10.5**。**

下列语句中，

```
cout<<int(n)<<', '<<double(n)<<endl;
```

使用了强制类型转换，int(n)**是将类** X **对象** n **转换成** int**，于是调用类型转换函数** operator int( )**，其值为** 64**。同样，**double(n)**也将调用类型转换函数** operator double( )**，其值为** 10.5**。**

请读者注意：前面曾讲过构造函数具有类型转换的作用。构造函数所提供的类型转换的方向与类型转换函数所提供的类型转换方向是相反的。例如，带有一个 int **型参数的构造函数提供将一个** int **型数转换为一个对象；而类型转换函数** operator int( )**却提供将一个对象转换为一个** int **型数。**

9. **上机调试本章例** 9.18 **程序。**

该程序中包含了较为丰富的类和对象方面的知识，可见教材中 258 **页的讲述。**

通过调试该程序，请读者在原程序基础上增加一些教材中 257 **页和** 258 **页所讲述的功能。**

10. **上机调试本章练习题的** 9.5 **题。**

通过调试该程序，读者可以根据情况再增加一些关于字符串处理方面的功能。

2.10 第10章上机练习指导

上机目的要求

1. 掌握派生类的定义方法和派生类构造函数的定义方法。
2. 掌握不同继承方式的情况下，基类成员在派生类中的访问权限。
3. 了解多继承中的二义性、支配规则以及虚基类的概念。

上机练习题

1. 上机调试下述程序，并对程序进行修改后再调试，指出调试中的出错原因。

程序内容如下：

```
#include<iostream.h>
class A
{
    public:
        void Seta(int i)
        { a=i; }
        int Geta( )
        { return a; }
    public:
        int a;
};
class B:public A
{
    public:
        void Setb(int i)
        { b=i; }
        int Getb( )
        { return b; }
        void Show( )
        { cout<<"A::a="<<a<<endl; }
    public:
        int b;
};

void main( )
{
        B b1;
       b1.Seta(6);
```

```
    b1.Setb(3);
    b1.Show( );
    cout<<"A::a="<<b1.a<<endl;
    cout<<"B::b="<<b1.b<<endl;
    cout<<"A::a="<<b1.Geta( )<<endl;
    cout<<"B::b="<<b1.Getb( )<<endl;
}
```

执行该程序后，输出如下结果：

```
A::a=6
A::a=6
B::b=3
A::a=6
B::b=3
```

按下列要求对程序修改，然后调试，对出现的错误分析其原因。

(1) **将派生类** B **的继承方式改为** private **时，会出现哪些错误或不正常现象？为什么？**

(2) **将派生类** B **的继承方式改为** protected **时，会出现哪些错误或不正常现象？为什么？**

(3) **将派生类** B **的继承方式恢复为** public **后，再将类** A **中数据成员** int **型变量** a **的访问权限改为** private **时，会出现哪些错误或不正常现象？为什么？**

(4) **派生类** B **的继承方式仍为** public**，将类** A **中数据成员** int **型变量** a **的访问权限改为** protected **时，会出现哪些错误或不正常现象？为什么？**

通过上述问题调试和分析，要求搞清如下问题：

(1) **在公有继承方式时，基类中成员在派生类中的访问权限如何？派生类的对象对基类中哪些成员可以访问？**

(2) **在私有继承方式时，基类中成员在派生类中的访问权限如何？派生类的对象对基类中哪些成员可以访问？**

(3) **在保护继承方式时，基类中成员在派生类中的访问权限如何？派生类的对象对基类中哪些成员可以访问？**

(4) **三种不同继承方式的特点是什么？**

2. **上机调试下列程序，并分析输出结果。**

程序内容如下：

```
#include<iostream.h>
class M
{
   public:
      void Setm(int i,int j)
      { m1=i; m2=j;}
      void Showm( )
      { cout<<"M: "<<m1<<', '<<m2<<endl; }
   protected:
```

```
        int m1;
    private:
        int m2;
};
class N:public M
{
    public:
        void Setn(int i,int j)
        { n1=i;n2=j; }
        void Shown( )
        {
                cout<<"N: "<<n1<<','<<n2<<endl;
                cout<<"M: "<<m1<<endl;
        }
    protected:
        int n1;
    private:
        int n2;
};
class P:public N
{
    public:
        void Setp(int i)
        { p=i; }
        void Showp( )
        {
                cout<<"P: "<<p<<endl;
                cout<<"N: "<<n1<<endl;
                cout<<"M: "<<m1<<endl;
        }
    private:
        int p;
};

void main( )
{
        P x;
        x.Setm(1,2);
        x.Setn(3,4);
        x.Setp(5);
```

```
    x.Showm( );
    x.Shown( );
    x.Showp( );
}
```

执行该程序后，输出如下结果：

```
M:1,2
N:3,4
M:1
P:5
N:3
M:1
```

对原程序进行下列修改，然后进行调试，对出现的错误说明其原因。

(1) **将派生类** N **的继承方式改为** protected**。**

(2) **将派生类** N **的继承方式改为** private**。**

(3) **将派生类** N **的继承方式改为** public **时，将派生类** P **的继承方式改为** protected**，再改成** private**。**

3. **上机调试本章例** 10.4 **程序，分析该程序的输出结果。**

进一步搞懂如下问题：

(1) **在调用派生类默认构造函数创建无参对象时，系统自动调用基类中的默认构造函数。**

(2) **在使用强制数据为类类型给对象数组元素赋值时，先调用构造函数创建临时对象，并将该临时对象给对象数组元素赋值，然后析构该临时对象。**

(3) **当对象结束生存期时，系统自动调用析构函数。在调用析构函数时，先调用派生类的，再调用基类的，这一顺序正好与调用构造函数的顺序相反。**

4. **上机调试本章例** 10.7 **程序，分析该程序的输出结果。**

分析该程序的重点在于分析该程序中出现的一般函数 fun( )**。该函数的形参是** M &p**，即是** M **类的对象引用。而程序中调用该函数时，实参是** N **类的对象**(*pn)**。看起来，形参与实参的类型是不同的。但实际上语法没有错误，其原因是形参与实参虽然类型不同，但是它们类型适应。请读者分析：它们为什么是类型适应的？类型适应的条件是什么？并请读修者改派生类** N **的继承方式为** protected **或** private **后，看一下是否还存在类型适应？**

5. **上机调试本章例** 10.8 **程序。**

调试该程序应注意如下问题：

(1) **该程序中有** 4 **个类：**A1**，**A2**，**A3 **和** D**，注意它们之间的关系。**

(2) **派生类** D **继承了两个基类** A1 **和** A2**，并有一个成员是类** A3 **的子对象。其构造函数的说明如下：**

```
D(int i,int j,int k,int l):A2(i),A1(j),A3(k);
```

在多继承派生类的构造函数中，成员初始化表中包含了对该派生类的所有基类的构造函数。

(3) **多继承派生类的析构函数也包含其所有基类的析构函数，请读者在本程序中通过定义析构函数来说明这一点。**

(4) **派生类构造函数调用基类构造函数的顺序取决于定义派生类时所给定的基类顺序，与派生类构造函数中成员初始化表里基类构造函数出现的顺序无关。请读者修改一下关于派生类** D **定义中基类** A1 **和** A2 **的顺序，观察输出结果中调用类** A1 **和类** A2 **构造函数的顺序。**

6. **上机调试下列程序，并分析输出结果。**

程序内容如下：

```
#include<iostream.h>
class A
{
    public:
        A(int i)
        { a=i; cout<<"A:: "<<a<<endl; }
        void Print( )
        { cout<<a<<','; }
        ~A( )
        { cout<<"Destructor ...A\n"; }
    private:
        int a;
};
class B1:public A
{
    public:
        B1(int i,int j):A(i)
        { b1=j; cout<<"B1:: "<<b1<<endl; }
        void Print( )
        { A::Print( ); cout<<"B1:: "<<b1<<endl; }
        ~B1( )
        { cout<<"Destructor ...B1\n"; }
    private:
        int b1;
};
class B2:public A
{
    public:
        B2(int i,int j):A(i)
        { b2=j; cout<<"B2:: "<<b2<<endl; }
        void Print( )
        { A::Print( ); cout<<"B2:: "<<b2<<endl; }
        ~B2( )
        { cout<<"Destructor ...B2\n"; }
    private:
```

```
        int b2;
};
class C:public B1,public B2
{
    public:
        C(int i,int j,int k,int l,int m):B1(i,j),B2(k,l)
        { c=m; cout<<"C:: "<<c<<endl; }
        void Print( )
        {
                B1::Print( );
                B2::Print( );
                cout<<c<<endl;
        }
        ~C( )
        { cout<<"Destructor ...C\n"; }
    private:
        int c;
};

void main( )
{
        C c(3,4,5,6,7);
        c.Print( );
}
```

执行该程序，输出显示如下结果：

```
A::3
B1::4
A::5
B2::6
C::7
3,B1::4
5,B2::6
7
Destructor…C
Destructor…B2
Destructor…A
Destructor…B1
Destructor…A
```

将原程序作下列修改：

(1) 派生类 B1 的继承方式前加 virtual(虚拟)关键字，说明类 A 为虚基类；同样，派生类 B2 的继承方式前也加 virtual。

(2) 在派生类 C 的构造函数的成员初始化表中，在后面增加一项 A(i)，用逗号与前一项分隔。

按上述要求修改完成后，再执行修改后的程序，将输出显示如下结果：

```
A::3
B1::4
A::5
B2::6
C::7
3,B1::4
3,B2::6
Destructor…C
Destructor…B2
Destructor…B1
Destructor…A
```

将该结果与前面结果比较，发现有哪些不同？为什么？

提示：从虚基类的作用分析。

7. 编写一个日期和时间的处理程序，要求在给定日期和时间后，分隔日期和时间，并分别输出显示。

程序内容请见本章例 10、例 11 程序。

该程序完成上述功能后，请再增加修改日期和时间的功能。

2.11 第 11 章上机练习指导

上机目的要求

1. 熟悉函数重载和运算符重载的定义和使用方法。
2. 了解静态联编和动态联编的概念。掌握动态联编的条件：公有继承和虚函数。

上机练习题

1. 调试下列程序，分析函数重载的条件。

程序内容如下：

```
#include<iostream.h>
class Square
{
   public:
      int squ(int i);
      double squ(double i);
      long squ(long i);
```

```
};
int Square::squ(int i)
{
    return i*i;
}
double Square::squ(double i)
{
    return i*i;
}
long Square::squ(long i)
{
    return i*i;
}

void main( )
{
    Square value;
    int a=5;
    double b=2.5;
    long c=125;
    cout<<"The square of "<<a<<" is "<<value.squ(a)<<endl;
    cout<<"The square of "<<b<<" is "<<value.squ(b)<<endl;
    cout<<"The square of "<<c<<" is "<<value.squ(c)<<endl;
}
```

调试该程序得出输出结果。

该程序中有三个成员函数重载，重载条件是参数个数相等，参数类型不同。

2. **上机调试本章中例** 11.2 **程序。**

在该程序中，有构造函数重载和运算符重载。从而了解如何在程序中定义运算符重载函数和使用运算符重载函数，并注意，在例 11.2 **程序中使用成员函数形式定义了** 4 **个重载运算符函数。在例** 11.3 **程序中使用友元函数形式定义这** 4 **个重载运算符函数，完成相同功能。请读者分析两种不同形式在定义和使用上的区别。**

3. **上机调试下列程序，该程序是对运算符**++**定义的重载运算符。通过对该程序的调试和分析，进而对运算符**++**的理解。**

程序内容如下：

```
#include<iostream.h>
class N
{
  public:
    N(int i)
```

```
        { n=i; }
        int operator ++( );
        int operator ++(int x);
        void Print( )
        { cout<<n<<endl; }
    private:
        int n;
};
int N::operator ++( )
{
        n++;
        return n;
}
int N::operator ++(int x)
{
        x=n;
        n++;
        return x;
}

void main( )
{
        N n1(5);
        int x=++n1;
        cout<<"x="<<x<<endl;
        n1.Print( );
        N n2(5);
        x=n2++;
        cout<<"x="<<x<<endl;
        n2.Print( );
}
```

执行该程序后，输出如下结果：

```
x=6
6
x=5
6
```

该程序采用成员函数的形式定义运算符重载函数++。下面采用友元函数的形式定义运算符重载函数++，类 N 定义如下：

```
class N
```

```
{
    public:
        N(int i)
        { n=i; }
        friend int operator ++(N &);
        friend int operator ++(N &,int);
        void Print( )
        { cout<<n<<endl; }
    private:
        int n;
};
int operator ++(N &a)
{
        a.n++;
        return a.n;
}
int operator ++(N &a,int x)
{
        x=a.n;
        a.n++;
        return x;
}
```

主函数与前面程序相同，执行后输出结果也一样。

4. **上机调试下列程序，从该程序中分析静态联编(编译时多态性)和动态联编(运行时多态性)的区别。**

程序的内容如下：

```
#include<iostream.h>
class Point
{
    public:
        Point(double i,double j)
        { x=i;y=j; }
        double Area( )
        { return 0.0; }
    private:
        double x,y;
};
class Rectangle:public Point
{
```

```
    public:
        Rectangle(double i,double j,double k,double l):Point(i,j)
        { w=k;    h=l; }
        double Area( )
        { return w*h; }
    private:
        double w,h;
};

void main( )
{
        Point p(3.5,7);
        double A=p.Area( );
        cout<<"Area="<<A<<endl;

        Rectangle r(1.2,3,5,7.8);
        A=r.Area( );
        cout<<"Area="<<A<<endl;
}
```

执行该程序后，输出结果如下：

```
Area=0
Area=39
```

该结果是按静态联编执行的，即体现出了编译时对 Area( )**函数的多态性。**

下面要获得动态联编，需要对程序做如下修改，先将类 Point **中** Area( )**函数前面加上关键字** virtual,**即说明该函数是虚函数,再使用下述主函数：**

```
void main( )
{
        Point *p;
        Rectangle r(1.3,2.4,5.6,7);
        p=&r;
        double A=p->Area( );
        cout<<"Area="<<A<<endl;
        Rectangle rl(1.3,2,5.6,8.5);
        Ponint &rr1=r1;
        A=rr1.Area( );
        cout<<"Area="<<A<<endl;
}
```

执行该程序输出结果如下：

```
Area=39.2
```

Area=47.6

请读者将类 Point **中** Area( )**函数前的** virtual **关键字去掉，调试并分析输出结果。**

上述是动态联编，即在运行时对 Area( )**函数实现多态性。**

5. **上机调试下列程序。**

该程序中出现下述语法现象，请读者在调试和分析中掌握。

(1) **成员函数调用虚函数实现动态联编。**

(2) **构造函数调用虚函数实现静态联编。**

(3) **析构函数调用虚函数实现静态联编。**

(4) **虚函数与所在类的访问权限无关。**

程序内容如下：

```
#include<iostream.h>
class B
{
    public:
        B( ) { }
        virtual void f1( )
        { cout<<"B::f1( )\n"; f2( ); }
        void f2( )
        { cout<<"B::f2( )\n"; f3( ); }
        virtual void f3( )
        { cout<<"B::f3( )\n"; }
        ~B( ) { }
};
class D:public B
{
    public:
        D( )
        { f2( ); }
        void f3( )
        { cout<<"D::f3( )\n"; f4(); }
        void f4( )
        { cout<<"D::f4( )\n"; }
        ~D( )
        { f3( ); }
};

void main( )
{
        D d;
        cout<<'\n';
```

```
    d.f1( );
    cout<<'\n';
    B *ptr;
    ptr=new D;
    cout<<'\n';
    ptr->f1( );
    cout<<'\n';
    delete ptr;
}
```

执行该程序输出结果如下：

```
B::f2( )
D::f3( )
D::f4( )

B::f1( )
B::f2( )
D::f3( )
D::f4( )

B::f2( )
D::f3( )
D::f4( )

B::f1( )
B::f2( )
D::f3( )
D::f4( )

D::f3( )
D::f4( )
```

如果去掉类 B **中的两个** virtual **关键，即不定义任何虚函数，上述程序输出如下结果：**

```
B::f2( )
B::f3( )

B::f1( )
B::f2( )
B::f3( )

B::f2( )
```

B::f3( )

B::f1( )
B::f2( )
B::f3( )

D::f3( )
D::f4( )

在该程序中，将类 B **中的某个虚函数访问权限改为** private,**仍不影响其输出结果。**

6. **上机调试本章例** 11.13 **程序。**

该程序使用指向类成员函数的指针来调用虚函数，也将采用动态联编。程序中指针 pf **是一个指向函数的指针，指向类的虚函数** f( )**。**

7. **上机调试下列程序，并分析输出结果。**

在该程序中出现了抽象类和纯虚函数。

将下列程序本章例 11.14 **程序相比较，说明它们之间的异同点。**

程序内容如下：

```
#include<iostream.h>
const double PI=3.1415;
class Shap
{
   public:
      virtual double Area( )=0;
};
class Triangle:public Shap
{
   public:
      Triangle(double h,double w)
      { H=h;W=w; }
      double Area( )
      { return 0.5*H*W; }
   private:
      double H,W;
};
class Rectangle:public Shap
{
   public:
      Rectangle(double h,double w)
      { H=h;W=w; }
      double Area( )
      { return H*W; }
```

```
    private:
        double H,W;
};
class Circle:public Shap
{
    public:
        Circle(double r)
        { R=r; }
        double Area( )
        { return PI*R*R; }
    private:
        double R;
};
class Square:public Shap
{
    public:
        Square(double s)
        { S=s; }
        double Area( )
        { return S*S; }
    private:
        double S;
};

double Total(Shap *s[ ],int n)
{
        double sum=0;
        for(int i=0;i<n;i++)
                sum+=s[i]->Area( );
        return sum;
}

void main( )
{
        Shap *s[5];
        s[0]=new Square(8.0);
        s[1]=new Rectangle(3.0,8.0);
        s[2]=new Square(12.0);
        s[3]=new Circle(8.0);
        s[4]=new Triangle(5.0,4.0);
```

```
    double sum=Total(s,5);
    cout<<"SUM="<<sum<<endl;
}
```

执行该程序后，输出结果如下：

```
SUM=443.056
```

8. **编程练习。**

给定一个 int **型数值** n**，编程按不同进制输出，包含十进制、八进制和十六进制。要求使用纯虚函数** print( )**。**

程序内容如下：

```
#include<iostream.h>
class N
{
  public:
    N(int i)
    { n=i;}
    virtual void print( )=0;
  protected:
    int n;
};
class H:public N
{
  public:
    H(int i):N(i)
    { }
    void print( )
    { cout<<"hex: "<<hex<<n<<endl; }
};
class D:public N
{
  public:
    D(int i):N(i)
    { }
    void print( )
    { cout<<"dec: "<<dec<<n<<endl; }
};
class O:public N
{
  public:
    O(int i):N(i)
```

```
    { }
    void print( )
    { cout<<"oct: "<<oct<<n<<endl; }
};

void main( )
{
    D d(128);
    H h(128);
    O o(128);
    d.print( );
    o.print( );
    h.print( );
}
```

执行该程序，输出结果如下：

```
dec:128
oct:200
hex:80
```

2.12 第 12 章上机练习指导

上机目的要求

1. **通过上机调试程序进一步了解引进模板的作用，熟悉函数模板和类模板的定义格式。**
2. **理解函数模板和类模板的应用。**

上机练习题

1. **上机调试本章例 12.3 程序，了解使用冒泡法进行排序操作的函数模板定义方法，并模仿该函数模板写出选择排序法的函数模板。**

选择排序法的原理如下：

先从所有要排序的数中找出最小的一个放在首元素位置；再从剩下的数中找出最小的放在首元素后面的位置，依次类推。每次都是从未排序的整数中找出一个最小的，并将它放在前面。最后，会将给定的若干个数从小到大排好序。

程序内容如下：

```
#include<iostream.h>
#include<string.h>
template<class Stype>  void select(Stype *item,int n);
void main( )
{
```

```
        char str[ ]="gdacbefh";
        select(str,(int) strlen(str));
        cout<<"The sorted string is "<<str<<endl;

        int num[ ]={6,2,7,5,4,1,3,8};
        select(num,8);
        cout<<"The sorted numbers are :";
        for(int i=0;i<8;i++)
            cout<<num[i] <<"    ";
        cout<<'\n';
};
template<class Stype>
void select(Stype *item,int n)
{
        int k;
        for(int i=0;i<n-1;i++)
        {
            k=i;
            for(int j=i+1;j<n;j++)
                if(item[j]<item[k])
                    k=j;
            if(k!=i)
            {
                int t=*(item+i);
                *(item+i)=*(item+k);
                *(item+k)=t;
            }
        }
}
```

2. **上机调试本章例** 12.6 **程序。**

本程序中使用了前面定义过的栈的类模板。

通过该例熟悉对类模板的定义和使用方法，同时了解栈的工作原理。

3. **上机调试本章例** 12.8 **程序。**

本程序定义了两个类模板：Listob＜Tdata＞**和** List＜Tdata＞**，它们之间是继承关系。**

该程序中给出了链表的 5 **种基本操作，读者如有兴趣还可以在此基础上添加新的操作。**

要求使用该程序中的类模板，对不同类型的结点数据进行操作。例如，char **型、**float **型或** deable **型等。**

4. **编程练习。**

使用函数模板采用 Shell **排序法，对已知数组中数据进行排序。**

Shell **排序方法原理如下：**

对所有要排序的数据项按某个间距量进行比较或交换；然后，再将其间距量减小，再比较或交换。最后，其间距量为 1，**排序结束。使用这种减少间距量的方法进行排序，通常使用的间距量为** 9，5，3，2，1。

程序内容如下：

```
#include<iostream.h>
#include<string.h>
template<class Stype>void shell(Stype *item,int n);
void main( )
{
    char str[ ]="gdacbefh";
    shell(str,(int) strlen(str));
    cout<<"The sorted string is "<<str<<endl;

    int num[ ]={6,2,7,5,4,1,3,8};
    shell(num,8);
    cout<<"The sorted numbers are ";
    for(int i=0;i<8;i++)
        cout<<num[i] <<"    ";
    cout<<'\n';
};
template<class Stype>
void shell(Stype *item,int n)
{
    int gap;
    Stype x;
    int a[ ]={9,5,3,2,1};
    for(int i=0;i<5;i++)
    {
        gap=a[i];
        for(int j=gap;j<n;j++)
        {
            x=item[j];
            for(int k=j-gap;x<item[k]&&k>=0;k-=gap)
                item[k+gap]=item[k];
            item[k+gap]=x;
        }
    }
}
```

5. **编程练习**

使用类模板编写一程序，要求该程序对数组元素进行求和、排序和查找。通过一个 int **型数组和** double **型数组的实例进行测试。**

程序内容如下：

```
#include<iostream.h>
#include<iomanip.h>
template<class T>
class Array
{
    public:
        Array(T *data,int i)
        { pset=data;a=i; }
        Array( )
        {   }
        void sort( );
        int seek(T keyword);
        T sum( );
    private:
        T *pset;
        int a;
};
template<class T>
void Array<T>::sort( )
{
    T b;
    for(int i=0;i<a;i++)
        for(int j=a-1;j>i;j--)
            if(pset[j-1]>pset[j])
            {
                b=pset[j-1];
                pset[j-1]=pset[j];
                pset[j]=b;
            }
}
template<class T>
int Array<T>::seek(T keyword)
{
    for(int i=0;i<a;i++)
        if(pset[i]= =keyword)
            return i;
        return -1;
```

```
}
template<class T>
T Array<T>::sum( )
{
        T s=0;
        for(int i=0;i<a;i++)
                s+=pset[i];
        return s;
}

void main( )
{
        int idata[8]={9,3,2,6,1,5,7,10};
        double ddata[8]={9.3,3.5,2.6,7.2,6.1,5.8,9.2,2.0};
        Array<int>Int(idata,8);
        Array<double>Dou(ddata,8);

        cout<<"Sum of idata is : "<<Int.sum( )<<endl;
        Int.sort( );
        cout<<"Result of sorting idata as follow: "<<endl;
        for(int i=0;i<8;i++)
                cout<<setw(6)<<idata[i];
        cout<<'\n';
        cout<<"Now,idata["<<Int.seek(5)<<"]=5"<<endl;
        cout<<'\n';
        cout<<"Sum of ddata is : "<<Dou.sum( )<<endl;
        Dou.sort( );
        cout<<"Result of sorting ddata as follow: "<<endl;
        for(i=0;i<8;i++)
                cout<<setw(6)<<ddata[i];
        cout<<'\n';
        cout<<"Now,ddata["<<Dou.seek(7.2)<<"]=7.2"<<endl;
}
```

运行该程后，输出结果如下：

```
Sum of idata is :43
Result of sorting idata as follow:
   1   2   3   5   6   7   9   10
Now,idata[ ]=5
Sum of sorting ddata as follow:
```

2 2.6 3.5 5.8 6.1 7.2 9.2 9.3

Now,ddata[5]=7.2

6. **上机调试下列程序。**

该程序中使用了说明多个类型参数的类模板，熟悉这种类模数的使用方法。

程序内容如下：

```
#include<iostream.h>
template<class M1,class M2>
class A
{
    public:
        A(M1 i,M2 j)
        { x=i;y=j; }
        void print( );
    private:
        M1 x;
        M2 y;
};
template<class M1,class M2>
void A<M1,M2>::print( )
{
    cout<<"x="<<x<<'\t';
    cout<<"y="<<y<<endl;
}

void main( )
{
    A <int ,double>A1(125,6.789);
    A <char,int>A2('R',789);
    A <char *,char*>A3("good","night");
    A <int,char *>A4(345,"class");
    A <long,char>A5(34567L, 'G');
    A1.print( );
    A2.print( );
    A3.print( );
    A4.print( );
    A5.print( );
}
```

执行该程序，输出显示如下结果：

```
x=125          y=6.789
x=R            y=789
```

x=good y=night

x=345 y=class

x=34567 y=G

2.13 第13章上机练习指导

上机目的要求

1. 熟悉标准流对象cin和cout的使用方法。
2. 熟悉插入符和提取符的用法及重载。
3. 熟悉系统提供的输入操作和输出操作函数。
4. 掌握格式化的输入/输出方法。
5. 掌握磁盘文件的输入/输出方法。
6. 掌握字符串的输入/输出方法。

上机练习题

1. 上机调试本章例13.5程序。

从该程序中找出C++程序进行屏幕输出的若干方法：

(1) 使用插入符输出显示一个字符的方法。

(2) 使用插入符输出显示一个字符串的方法。

(3) 使用成员函数put( )输出显示一个字符串的方法。

(4) 使用成员函数write( )输出显示一个字符串的方法。

2. 上机调试本章例13.9程序。

通过调试本程序，要求掌握若干键盘输入方法。该程序中出现了下述的输入函数，学会它们的使用方法：

(1) getline( )函数

(2) write( )函数

另外，还应掌握下述两种输入方法：

(1) 提取符>>

(2) get( )函数

3. 上机调试本章例13.12程序。

本程序使用控制输出格式的成员函数进行输出操作。该程序中出现了下述控制输出格式的成员函数：

(1) width( )

(2) fill( )

(3) setf( )

(4) precision( )

搞清上述函数的功能、参数及用法。

4. 上机调试本章例13.13程序。

该程序使用控制符(操作子)进行格式输出，掌握该程序中出现的下述操作子的功能及用

法：

(1) setw( )

(2) resetiosflags( )

(3) setiosflags( )

(4) setfill( )

(5) setprecision( )

(6) dec,hex 和 oct

5. 上机调试本章例 13.19 程序。

该程序用来将一个已知文件 file2.dat 复制成另一个文件 file3.dat，它相当于 DOS 系统中的 copy 命令。通过该程序的调试进一步掌握下述问题：

(1) 如何以读方式或写方式打开一个文本文件？该例中提供了一种方法，试用其他方法上机调试。

(2) 对一般磁盘文件的读写函数与对键盘、屏幕的读写函数是否相同？

(3) 为使该程序应用更加方便，采用主函数 main( )带有参数的形式，请读者添加主函数参数后上机调试。主函数参数形式如下：

```
void main (int argc,char * argv[ ])
{
  //函数体
}
```

6. 上机调试本章例 13.20 程序。

通过调试该程序掌握对一个二进制文件的操作。具体要求如下：

(1) 该程序中用 struct 定义一个类，其名为 person。又定义一个对象数组 people[5]。熟悉对象数组的定义和初始化方法。

(2) 程序中打开了一个文件 file4.dat，该文件打开方式为可读可写，并为二进制文件。请读者再用其他方法打开该文件，要求满足该打开需求。

(3) 掌握下列程序中所用函数的功能和用法。

① write( )函数

② seekp( )函数

③ read( )函数

④ close( )函数

(4) 该程序的主函数可以带有参数，请读者添加主函数参数后进行调试。

7. 上机调试本章例 13.22 程序。

通过调试该程序，学会对随机文件读写的操作方法。具体要求如下：

(1) 如何打开一个又写又读的二进制文件？

(2) 如何使用 write( )函数向文件内写数据？

(3) 如何使用 tellp( )函数定位写指针？

(4) 如何使用 seekp( )函数来设置写指针的位置？

(5) 如何使用 read( )函数来从打开文件中读数据？

(6) 如何使用 seekg( )函数来设置读指针的位置？

(7) 分析该程序的输出结果。

8. **上机调试下列程序，掌握关于字符串流的操作。**

(1) **本章例** 13.26 **程序；**

(2) **本章例** 13.27 **程序；**

(3) **本章例** 13.28 **程序。**

第 3 章　习题答案和难点分析

3.1　第 1 章习题答案和解答

3.1.1　习题答案

习题 1.1　**(略)**

习题 1.2　**选择填空**

C D A A B

习题 1.3　**判断题**

正确的：(1)(3)(5)(6)(10)

习题 1.4　**分析程序结果**

(1) BeiJing ShangHai
　　TianJing

(2) x=8,y=7
　　x−y=1

(3) r=k
　　i+j=26

习题 1.5　**改错并分析输出结果**

(1) **有三个错：**①main( )**函数前要加** void **作类型说明；**②**程序头应加文件包含命令，即** ＃include〈iostream.h〉**；**③**函数体中输出语句结束应加分号。**

执行该程序输出结果如下：

This is a program.

(2) **该程序有两处错：**①**变量** x **应先说明再使用，即在主函数体内加** int x**；** ②**输出语句** cout<<＂y=<<y<<\n＂;**应改为** cout<<＂y=＂<<y<<＂\n＂;

执行该程序，当给 x **变量赋值为** 5 **时，输出结果如下：**

y=25

(3) **该程序仅有一处错：变量** b **定义后应先赋值再使用，否则** b **中的值是无意义的。**

执行该程序，假定给 b **赋值为** 10 **时，输出结果如下：**

a+b=17

习题 1.6　**回答下列各问题**

(1) **答：在编写** C++**程序时应注意** 3 **个问题。**

① C++**程序中，定义任何函数都应指出该函数的类型。对于主函数来讲，大多采用如下两种方式：**

方式一

```
void main( )
```

```
{
        …
}
```

此种方式将主函数说明为无返回值的函数。

方式二

```
int main( )
{
        …
    return 0；
}
```

方式二将主函数说明为 int 型函数，在函数体内加写一条带有表达式的 return 语句。此返回值不影响程序结果。

② 编写 C++程序时，经常在程序头加上如下的文件包含命令：

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

因为程序中输入输出语句出现的对象和运算符包含在文件 iostream.h 中。

③ C++程序中语句与 C 语言程序相同，每条语句结束都必须加分号。

(2) 答：C++程序中所出现的变量或对象都必须先说明后使用，否则出现编译错。这一点与 C 语言相同。说明变量时不一定都放在函数体的开头，可以根据需要在程序中随时说明，这一点与 C 语言不同。

(3) 答：使用 cout 和插入符运算符<<输出字符串时，所用的表达式可以是字符数组名或字符指针名，也可以是字符串常量，但需用双撇号括起来。例如，下列写法是正确的：

```
cout<< "good morning" <<endl;
```

输出显示字符串 “good morning”。

(4) 答：程序中说明的变量，如果有默认值，则其可以使用；否则必须赋值后才能使用。外部存储类和静态存储类变量说明后有默认值，而自动存储类和寄存器类变量说明后无默认值。因此，没有默认值的变量在使用前必须先赋值。

(5) 在编译时可以发现一个程序中的语法错误。当这个程序通过编译连接后，生成了可执行文件，运行后获得输出结果。当然该结果不一定是正确的，因为可能由于算法的错误造成输出结果的错误，而算法错在编译连接时是检查不出来的。

3.1.2 习题解答

本章中没有难解问题。初次编写 C++程序时应该掌握其编程规则，注意 C++编程与 C 语言编程的区别。

C++语言的程序结构与 C 语言程序结构基本相似。其程序都是由文件组成，文件由函数组成，函数由语句组成，语句由单词组成。一个程序实际上是一个函数串。程序中的若干个函数有且仅有一个是主函数，执行程序时从主函数开始，其余的函数是由主函数或被主函数的调用的函数所调用。组成函数体的每条语句都以分号结束。

C++程序与 C 语言程序不同之处主要表现在如下几点：

(1) C++程序中所定义的每一个函数都必须指出其类型，主函数也不例外。

(2) C++程序的函数体内可以随时说明变量，而不必像 C 语言程序那样一定要把变量说明放在函数体的开头。

(3) C++语言的类型要求比 C 语言严格，这表现在类型转换多用强制方法，函数说明使用原型说明。

(4) C++程序中引入了引用概念，指针使用比 C 语言程序少。

(5) C++语言是面向对象的程序设计语言，程序中使用类和对象。

3.2 第 2 章习题答案和解答

3.2.1 习题答案

习题 2.1 **(略)**

习题 2.2 **选择填空**

B D A B D D C C B C

习题 2.3 **判断题**

描述正确的有：(2)(4)(8)(10)

习题 2.4 **分析程序结果**

(1) 90；34.92；12.7865
67，65

(2) P
1234567
1.98765

(3) 3，4，5，8
5.6，？，？
5，8，2

(4) abde
abcde
edcba
abcd,ijkl,sxyz
74HK

(5) 1.5e+006
0.005
mnp□□□□□xyabc
\abc‖mnp＇xyz＇

3.2.2 习题解答

第 2.4 **题的**(1)**题**

该程序编译时出现 Warning **型编译错，这是用常双精度浮点数给单精度浮点型变量赋值引起的。消除该** Warning **型错的方法是将**

```
float b=34.92；
```

改写为

```
float b=(float)34.92;
```

另外，该程序中下列语句

```
cout<<c+1<<’，’<<c−1<<endl;
```

的输出结果是 67，65，**而不是字符** C，A。

如果要输出显示字母 C，A，则需用下列输出语句

```
cout<<(char)(c+1)<<’，’<<(char)(c−1)<<endl;
```

一般地讲，使用 cout 和插入运算符 << 输出字符变量的值时，输出的是该变量的字符；输出字符表达式时，因为字符表达式(如，c+1，c−1 等)的值是算术值，因此输出结果是 int 型值。

第 2.4 题的(2)题

该程序是一个有关一维和二维数组的定义、初始化及输出的程序。程序中，数组 d 是一个具有 5 个元素的一维数组，对该数组的前 3 个元素 d［0］,d［1］,d［2］进行初始化，其值分别是 1.2，3.4，5.6，其余的 2 个元素没有被初始化，它们的值都为 0。在下列输出语句中，

```
cout <<d［2］<<＇,＇<<d［3］<<＇,＇<<d［5］<<endl;
```

输出显示的结果为

```
5.6，0，？
```

其中，？表示是一个无意义的值，实际显示为一个随机数。因为数组 d 共有 5 个元素，它们分别表示为 d［0］，d［1］，d［2］，d［3］，d［4］，其值分别为 1.2，3.4，5.6，0 和 0。而 d［5］不是数组 d 的元素，它只是在地址上有意义，表示数组元素 d［4］后面内存地址中存放的数值，而且显示它是一个无意义的数。

第 2.4 题中(5)题

该程序中输出双精度浮点数 1.5e6 时，按其规定格式输出显示的结果为 1.5e+006；而输出双精浮点数.5e−2 时，按规定格式输出显示的结果为 0.005。当用小数点表示数值位数太多时，则用 e 指数表示。例如，当输出.5e−4 时，输出结果是 5e−005。

该程序中，下列输出语句

```
cout<<＂abc\rmnp\txyz\babc\n＂；
```

输出显示一个字符串。在用双撇号括起来的字符中有一般可打印字符，也有转义序列字符，如\r,\t,\b 和\n，输出时按其功能执行。具体输出过程如下：

先输出 abc；再输出\r 时，光标移至字符 a 处，接着输出 mnp 时，用 mnp 将 abc 覆盖了；接着输出\t，这是一个水平制表符，按通常规定，它将从指定位置向右跳过 8 个字符，它是从最左列开始，每次跳过 8 个字符段。这时输出显示结果如下：

```
mnp□□□□□
```

接着输出 xyz；又输出\b，\b 是后退一个字符，于是光标在字符 z 下，再输出字符 a，将字符 z 覆盖，又输出 bc；最后输出\n，即换行，将光标移下一行首列。于是获得下述结果：

```
mnp□□□□□xyabc
```

该程序中的其他输出结果请自行分析。

3.3 第3章习题答案和解答

3.3.1 习题答案

习题 3.1 (**略**)

习题 3.2 **选择填空**

A A D D D D C A D D

习题 3.3 **判断题**

描述正确的有：(1)(2)(4)(10)

习题 3.4 **分析程序结果**

(1) 63，45
18，45
88，3

(2) 3
80
1
9，20

(3) 53，22
−2，0
1
1

习题 3.5 **求表达式值及** a，b **值**

(1) 0
(2) 1
(3) 3
(4) 16
(5) 12
(6) 1

习题 3.6 **编程(略)**

3.3.2 习题解答

本章主要讲述运算符和表达式，要求掌握各种表达式的求值方法。

第 3.2 **题中**(5)**题**

该题判断 4 **个表达式中哪一个是非法的。表达式是否合法，应根据运算符使用正确与否进行判断。例如，有的运算符只能作用于** int **型操作数，如** %(**求余数**)**，如果将它作用于浮点型操作数，则该表达式是非法的。又例如，有的运算符，如**++**或**−−**，只能作用于变量，如果将它作用于常量或表达式，则该表达式是非法的。**

在该题给出的 4 **个表达式中，前两个显然是正确的。表达式**−−a**，按单目运算符结合性从左到右，又根据表达式中的分隔原则，当多个运算符连写中间没有分隔符时，按尽量取大，**

该表达式被分隔为- --a，变量 a 先作减 1 运算后，再取负值，这是合法的。表达式--(-a)用括号将连续出现的两个运算符进行了分隔，该表达式是先将变量 a 求负，然后再将-a 表达式的值进行减 1 操作。该操作是非法的，因为减 1 运算符不能作用于表达式，只能作用于变量。

第 3.2 题中(6)题

该题判断给定的 4 个表达式中哪一个是合法的，判断方法同上。表达式 5.6%4 是非法的，因为求余运算符不能作用于浮点数，只能作用于整型数。表达式 a+1=5 是非法的。因为赋值运算符要求其左值(赋值运算符左边的值)是某种类型的变量，或者是具有地址值的表达式(在指针部分讲到)，而不可为一般表达式。这里，a+1 是一般表达式，因此是非法的。表达式!a*=5 是非法的，因为复合赋值运算符“*=”的左值是一般的表达式!a。表达式“a=2，a+2，2*a”是合法的，该表达式是一个逗号表达式，它由三个表达式用逗号运算符连接起来的，三个表达式分别是 a=2，a+2 和 2*a。

第 3.4 题中(1)题

该程序主要用来进行位操作运算符的练习。位操作运算符是用来进行二进制位操作的，共有 6 种，其中有一种是单目的，其余都是双目的。这六种运算符是：

～按位求反(单目)；& 按位与；| 按位或；^ 按位异或；<<左移；>>右移

该程序中，已知两个 int 型变量 a 和 b，它们的值用二进制表示为

$a=(0\cdots00010110)_2$，$b=(0\cdots00111011)_2$

在 32 位机上，无符号整型数占 4 个字符，即 32 位二进制。上述二进制表示中，“…”表示连续若干个 0 或连续若干个 1，展开后与已标出的总共占 32 位。

$a|b=(0\cdots00111111)_2=(63)_{10}$

$a\&b=(0\cdots00010010)_2=(18)_{10}$

$a\hat{}b=(0\cdots00101101)_2=(45)_{10}$

$\sim a\hat{}\sim b=(1\cdots11101001)_2\hat{}(1\cdots11000100)_2=(0\cdots00101101)_2=(45)_{10}$

a<<=2,a 中值为$(0\cdots01011000)_2=(88)_{10}$

a>>=4,a 中值为$(0\cdots00000011)_2=(3)_{10}$

第 3.4 题中(2)题

该程序中出现的运算符比较多，有 4 个输出语句，5 个表达式。下面分析各个表达式的求值方法。

表达式++i-j--的值，按优先级应该这样计算(++i)-(j--)，由于 i 值为 10，++i 值为 11；由于 j 值为 8，j--值仍为 8，相减后其值为 3。这时，i 值为 11，j 值为 7。

表达式 i=i*=j 的值，按结合性应该这样计算 i=(i*=j)，由于 i 值为 11，j 值为 8，计算 i*=j 后，i 值为 80，再将 80 赋值给 i，最后 i 值为 80，表达式值为 80。

表达式 i=3/2*(j=3-2)的值，按优先级和结合性计算顺序为 i=((3/2)*(j=3-2))，由于 3／2 为 1，3-2 为 1，即 j 值为 1，1*1 为 1，将该值赋给变量 i，i 值为 1，表达式值为 1。

表达式 i&j|1 的值，按优先级计算顺序为(i&j)|1。由于 i 为 10，转换为二进制为$(0\cdots01010)_2$，j 为 8，转换为二进制为$(0\cdots01000)_2$，i&j=$(0\cdots01000)_2=(8)_{10}$。再计算 8|1，二进制表示为$(0\cdots01001)_2$，转换为十进制为 9。

表达式 i+i&0xff 的值，按优先级规定，计算顺序为(i+i)&0xff。由于 i 为 10，i+i 为 20，二进制表示为$(0\cdots000010100)_2$，将它与 0xff 的二进制表示$(0\cdots011111111)_2$按位与，结果为 $(0\cdots000010100)_2$，即$(20)_{10}$。

第 3.4 题中(3)题

该程序中共有 7 个表达式，下面分别计算它们的值。

表达式 3+2 的值为 5，表达式 2+1 的值为 3，输出结果为 53。由于这两个表达式之间没有分隔符，因此，5 与 3 紧接着，这样容易产生误解，所以在输出多个表达式值时，前一个表达式值与后一个表达式值之间最好加一个分隔符，可以是逗号、分号或冒号，也可以是空格符。

表达式 2*9|3<<1 的值，按优先级规则，该表达式计算顺序是((2*9)|(3<<1)),2*9 为 18，将 3 左移 1 位后，其值为 6，18 与 6 按位或后，其值为 22。

表达式 5%3*2/6-2 的值，按优先级和结合性的规则，其计算顺序为(((5%3)*2)/6)-2，先计算 5%3 值为 2，2*2 值为 4，4/6 值为 0，0-2 值为-2，该表达式值为-2。

表达式 8=3<=2&6 的值，按优先级规则，该表达式计算顺序为(8=(3<=2))&6，先计算 3<=2 其值为 0，8=0 其值为 0，0&6 其值为 0，该表达式值为 0。

表达式!('3'>'6')||4<9 的值，按优先级规则，计算顺序为(!('3'>'6'))||(4<9)。计算'3'>'6'的值为 0，!0 值为 1，这时已可确定该表达式的值为 1，不需再计算(4<9)的值。

表达式 6>=3+2-('0'-6)的值，按优先级和结合性规则，该表达式计算顺序为 6>=((3+2)-('0'-6))，这里'0'用字符 0 的 ASCII 值。查知'0'的 ASCII 的值为 48，48-6 为 42，3+2 值为 5，5-42 值为-37，6>=-37 值为 1。该表达式值为 1。

3.4 第 4 章习题答案和解答

3.4.1 习题答案

习题 4.1 (略)

习题 4.2 选择填空

C A B D D A C B B D

习题 4.3 判断题

描述正确的有(2)，(5)～(9)。

习题 4.4 分析程序结果

(1) 49
47
43
41

(2) 7
OK

(3) 6
8

(4) 5
3
1
-1

(5) 1，2

(6) SWITCH WAMP

(7) 20

习题 4.5 编程(略)

3.4.2 习题解答

第 4.2 题中(4)题

该题实际上是计算 do-while 循环的循环次数。开始时 i=5，执行一次 do-while 循环的循环体后，i 值为 3。由于这时表达式 i!=0 值为 1，再执行一次 do-while 循环的循环体，这时 i 值为 1，同样 i!=0 值为 1，再执行一次循环体后，i 值为−1，这时表达式 i!=0 值仍为 1。无论再执行多少次循环体，i 值永不会为 0。因此，该循环的循环次数应该是无限。

第 4.2 题中(5)题

计算 for 循环的循环次数。开始时，i 为 0，j 为 10，计算表达式 i=j=10 的值为非 0，执行循环体，再计算 i++和 j−−，然后计算表达式 i=j=10 的值仍为非 0。由于表达式 i=j=10 的值永远为非 0，因此，该循环的次数为无限。

第 4.2 题中(6)题

计算 while 循环的循环次数。先计算该循环的循环条件表达式 i(0)的值为 0，于是退出该 while 循环。因此，该循环执行 0 次。

第 4.2 题中(8)题

该题判断开关语句 4 种写法中哪一个是正确的。

A **是错误的。因为 case 子句中，case 后面的表达式应为常整型表达式，不能是变量。所以，在 case 后面用变量 a,b 是错误的。**

B **是正确的。**

C **是错误的。因为在 case 后面不能是逗号分隔的两个整型数，也不能是逗号表达式。**

D **是错误的。因为在 case 后面不能接带有变量的表达式。**

第 4.4 题中(1)题

该程序使用了 while 循环，在 while 循环体内出现了 break 语句和 continue 语句，这两个语句都出现在同一个 if 语句中。执行 while 循环时，计算−−a 值为 49，即非 0，执行循环体。由于表达式 a==40 值为 0，跳过 break 语句。又由于 a 这时不是 2 的倍数，也不是 3 的倍数，跳过 continue 语句，输出 a 的值，即显示 49。再次计算−−a 值为 48，即非 0，执行循环体，这时 a 不等于 40，跳过 break 语句，a 为 2 的倍数，执行 continue，继续计算−−a 的值为 47，由于 a 不等于 40，不是 2 的倍数，也不是 3 的倍数，跳过 break 语句和 continue 语句，输出 a 的值为 47。接着，每次循环 a 减 1，由于 46，45，44 和 42 时，都跳过 break 语句，执行 continue 语句，这些数没有输出，而 a 为 43 和 41 时，跳过 break 语句，又跳过 continue 语句，并输出显示 43 和 41。当 a 为 40 时，执行 break 语句，则退出 while 循环，结束该程序。

第 4.4 题中(2)题

该程序中有 do-while 循环，循环体内 if 语句中有 break 语句。do-while 循环先做一次循

环体，输出显示 7。计算表达式 a==9 时，其值为 0，则退出 do-while 循环，输出显示 ok。结束该程序。

第 4.4 题中(3)题

该程序中使用两个 if 语句。第一个是 if-else if-else 语句，在 else if 体内又嵌套一个 if 语句。第二个是一个简单的 if 语句，if 体又是一个简单的 if 语句，该语句的 if 体是一个 if-else if 语句。

执行 if 语句时，关键是判断其条件是否成立。条件成立，即表达式值为非 0，执行该 if 体或 else if 体。先分析第一个 if-else if-else 语句表达式 !a 值为 0，表达式 b 为非 0，执行 else if 体。由于 c 为非 0，改变 d 值为 5，退出该 if 语句，d 增 1 后，输出 d 为 6。再分析第二个 if 语句，由于 a<b 为非 0，又表达式 a!=3 为非 0，执行 if 体，该 if 体是一个 if-else if 语句。由于 !c 为 0，c 为非 0，则改变 a 值为 5，分别退出两个 if 语句。再做 d+=2 后，d 值为 8，输出 d 值为 8。

第 4.5 题中(4)题

该程序出现 do-while 循环，该循环体内包含一个开关语句。先执行一次循环体，定义变量 a，其值为 7。执行开关语句，计算表达式 a%2 的值为 1，执行 case 1 后面的语句，改变变量 a 值为 6，退出开关语句，a 值减 1 后，输出显示 a 值为 5。计算循环条件表达式 a>0 为非 0，则第二次执行循环体，表达式 a%2 值为 1，执行 case 1 后面语句，a 减 1，退出开关语句，a 再减 1，输出显示 a 值为 3。由于 a>0，再次执行循环体，输出显示 a 值为 1。这时，a 的值大于 0，再次执行循环体，输出显示 -1 后，这时 a<0，则退出 do-while 循环，结束程序。

第 4.4 题中(5)题

该程序中出现了开关语句嵌套的形式，即在开关语句中的 case 子句的语句序列中又出现开关语句。

执行程序中的开关语句，表达式 a 值为 5，执行 case 5 后面的语句序列，该语句序列又是一个开关语句。计算表达式 b 值为 6，执行 case 6 后面的语句序列，变量 j 增 1 后为 1，退出内层开关语句，接着执行 case 6 后面的语句序列 (因为这里没有 break)，变量 i 增 1 后为 1，变量 j 增 1 后为 2，执行 break 语句，退出外层开关语句。输出显示 i 和 j 值为 1 和 2。

第 4.4 题中(6)题

该程序比较复杂，程序中出现 for 循环，其循环体是一个开关语句，开关语句中又有循环语句。该程序中定义了一个字符串，字符串中'\1'表示数字 1，'1'表示字 1。通过 for 循环每次从字符串中取出一个字符，直到所有字符取完为止。第一次循环取出字符 'S'，执行开关语句中，default 子句后面的语句序列，输出显示字符 S，执行 continue，结束本次循环。第二次循环取出字符 'W'，同样执行开关语句中 default 后面的语句序列，输出显示字符 W，结束本次循环后，第三次循环取出字符'I'，同样执行开关语句中 default 后面的语句序列，输出显示字符 I。再一次循环取出字符 'L'，执行 case 'L'后面的语句序列，结束本次循环，字符 L 将不被输出显示。再一次循环取出字符 'T'，将被显示在屏幕上，取出字符 'E'，将不被显示，取出

字符'C'和'H'，都被输出显示。取出字符'1'时，退出开关语句，输出空格符后，继续下次循环。取出字符'\1'，即数字1时，执行case 1后面的while循环语句，该循环语句的循环体是空语句。计算条件表达式，取出字符'\11'不等于'\1'，也不等于'\0'，执行一次循环体。再计算条件表达式，取出字符'W'不等于'\1'，也不等于'\0'，又执行一次循环体，再计算条件表达式，取出字符'\1'，它将与'\1'相等，该条件表达式值为0，退出while循环，执行后面的语句序列，结束本次循环。再去执行for循环，取出字符'W'，它将被输出显示，结束本次循环。再取出字符'A'，被输出显示。取出字符'L'，将不被显示，再取字符'L'，也不被显示，再取出'M'，'P'都将被输出显示。最后取出字符'1'，则退出开关语句，输出一空格字符。再次循环取出字符'\0'，则结束for循环，输出换行符，结束整个程序。总结上述分析结果，执行该程序应在屏幕上显示如下信息：

SWITCH WAMP

第4.5题中(7)题

该程序出现宏定义，编译该程序前，要进行宏替换，将

```
q=MAX(m,n+p)*10;
```

换成

```
q=(m)>(n+p)?(m):(n+p)*10;
```

当m为1，n为2，所以p为0时，代入上式

```
q=1>2?1:2*10;
```

计算1>2为0，所以q的值为20。

在宏替换时，只是简单地替换，不应随意增加或减少括号和其他符号。如果将上式替换为

```
q=(m>n+p?m:(n+p))*10;
```

则这种替换是错误的。

3.5 第5章习题答案和解答

3.5.1 习题答案

习题5.1 (略)

习题5.2 选择填空

D A B B D C B A C B

习题5.3 判断题

描述正确的有：(1)(2)(4)(6)(8)(10)

习题5.4 分析程序结果

(1) 7
9
11
13

(5) 8，8，8

(6) Sum1=10
Sum2=15
Sum3=20

(2) 15

(3) 6，1

(4) 5!+4!+3!+2!+1!=153

(7) a=10，b=5

(8) m
2000
good

习题 5.5　**编程(略)**

3.5.2　习题解答

本章主要讲述函数的定义和说明，函数参数及返回值，函数的调用方式，嵌套调用和递归调用以及存储类等内容。这里，关于函数的规则与 C **语言有些不同，请注意这些不同之处。**

第 5.2 **题中**(5)**题**

在 C++**语言中可以设置函数参数的默认值，这一点是** C **语言中没有的。一个函数可以对其部分参数或全部参数设置默认值，设置默认值的参数按照从右至左的顺序，即不允许在设置默认值的参数右边出现没有设置默认值的参数。参数默认值可设置在函数定义语句中，也可以定义在函数说明语句中。如果一个函数具有说明语句时，默认参数一定要设置在说明语句中。**

第 5.2 **题中**(6)**题**

在 C++**语言中引进了引用概念，这也是** C **语言中所没有的。引用实际上是已知变量的别名。引用经常用来作函数参数，实现引用调用。引用调用的特点是实参调用变量值，形参调用引用名，调用时传递的是地址值。这种调用可以在被调用函数中改变调用函数的参数值，它具有** C **语言中指针作函数形参的作用。因此，在** C++**中经常调用引用作函数形参替代指针作函数形参，使得指针的使用明显减少，这样可以避免使用指针所带来的麻烦。**

第 5.4 **题中**(1)**题**

该程序中有两个函数：主函数和被调用函数 fun( )**。由于主函数定义在前，被调用函数定义在后。因此，在调用** fun( )**函数之前，该函数使用原型说明方法进行了说明。**

在主函数中，有一个 for **循环，该循环共循环** 4 **次，循环体调用** fun( )**函数。**

在 fun( )**函数中，定义了两个变量，一个是自动存储类的变量** b**，另一个是内部静态存储类的变量** a**，其中变量** a **的默认值为** 0**，变量** b **的初值为** 5**。第一次调用** fun( )**函数时，执行表达式** a+=2 **后，**a **值为** 2**，输出** a+b **值为** 7**。第二次调用** fun( )**函数时，自动类变量** b **重新创建获初值为** 5**，内部静态变量** a **保存被改变的值** 2**，执行** a+=2 **后，**a **值为** 4**，输出** a+b **值为** 9**。第三次调用** fun( )**函数时，自动类变量又被重新建立，其值为** 5**，内部静态变量** a **保存被改变的值** 4**，再执行** a+=2 **后，**a **值为** 6**，输出** a+b **值为** 11**。同样地，最后一次调用** fun( )**函数时，**b **值仍为** 5**，变量** a **值为** 6**，执行** a+=2 **后，**a **值为** 8**，输出** a+b **值为** 13**。**

该程序分析中应注意两个问题：

① 函数的说明应放在调用该函数之前。

② 在一个函数中，自动存储类变量和内部静态类变量之间有区别。其区别表现在寿命上，即存在性上。自动类变量超出作用域后被释放，内部静态类变量超出作用域后虽然不可

见，但是仍然存在，即不被释放，一旦回到该作用域后仍然保留原来的值。

第 5.4 题中(2)题

该程序中有两个函数：main( )和 sum( )。

main( )函数定义在前，sum( )函数定义在后，程序开始先说明了 sum( )函数，并且使用了函数原型。

该程序中又定义了两个外部存储类的变量 x 和 y，并且进行了初始化。在 main( )函数中使用两个外部类变量 x 和 y 作为调用 sum( )函数的实参。由于外部类变量 x 和 y 是先引用后定义的，因此在引用之前，对 x 和 y 进行了说明。说明外部类变量的格式如下：

extern〈数据类型〉〈变量名表〉；

定义外部类变量时不加任何存储类说明符，只要写在函数体外就可以了。对于外部类变量在一个程序中只能定义一次，可以说明多次(在多文件程序中)。

第 5.4 题中(3)题

该程序中有两个函数：main( )和 f( )。被调用函数 f( )定义在前，调用前不必说明。

该程序在 main( )函数中调用 f( )函数。f( )函数中有一个 if 语句。由于实参为 8，因此，表达式 n＞0 为非 0，执行 if 体，输出 6，1。

第 5.4 题中(4)题

该程序用来求若干个自然数的阶乘之和。该程序有两个函数：一个是主函数，另一个函数用来求一个数的阶乘，即 fac( )函数。

该程序由于求若干个连续数的阶乘之和，所以在函数 fac( )中，采用了一个内部静态存储类变量 b，并用 0!(即 1)给它初始化。当以 1 为实参调用 fac( )函数时，执行语句

```
b*=a;
```

使 b 中保存 1!值。当用 2 为实参调用 fac( )函数时，执行语句

```
b*=a;
```

使 b 中保存 2!值。依次类推。

在 main( )中，通过 for 循环，依次求出 1～5 各个数的阶乘之和。

请读者注意，该程序的 fac( )函数中，内部静态变量 b 用得很巧妙，值得效仿。

第 5.4 题中(5)题

该程序中也有两个函数：main( )和 fun( )函数。

函数 fun( )中有三个参数，前两个是 int 型变量，后一个是指向 int 型变量的指针。在调用该函数时，前两个实参应是一个表达式的值，包含 int 型常量或变量名，后一个实参应该是某个 int 型变量的地址值。在 fun( )函数中，可以通过改变指针形参所指向的变量来改变实参的变量值。fun( )函数中，下列语句

```
*k=j-I;
```

就是通过改变指针形参 k 所指向的变量 a，或 b，或 c 来改变实参 a，b，c 的值。

第 5.4 **题中**(6)**题**

该程序也是由两个函数组成：main( )**和** add( )**。该程序的特点如下：**

① **在** C++**中可以设置参数的默认值。该程序中将函数** add( )**的第二个形参的默认值设置为** 5**。于是，可以用一个实参调用** add( )**函数，也可以用两个实参调用** add( )**函数。**

② **调用函数的实参可以是调用函数，即用该调用函数的返回值作该实参的值，并且还可以嵌套。下列表达式中**

add(m,add(m,add(m)))

先执行 add(m)**，将其返回值** 10 **作为另一个函数** add( )**调用的第二个实参值，再执行** add(m,10)**，又将其返回值** 15 **作为下一个** add( )**调用的第二个实参值；最后，执行** add(m,15)**，输出显示其返回值** 20**。**

第 5.4 **题中**(7)**题**

该程序的特点是使用引用作函数形参。该程序的 swap( )**函数中，有两个以** int **型变量的引用作为形参。在主函数中，调用** swap( )**函数时，用两个** int **型变量名作函数实参。这是引用调用的一个实例。在** swap( )**函数中，交换了两个引用的值，相对应的两个实参变量值** a **和** b **也被交换。该程序就是用来验证这一点的。在** C++**程序中，经常用引用作函数参数，以减少调用时数据传递的开销。**

第 5.4 **题中**(8)**题**

该程序的特点是函数重载。函数重载是 C++**中一个重要概念和用法。**

该程序中定义了三个同名函数 print( )**，这三个函数名字相同，类型相同，参数个数也相同；但是，参数类型不同，于是实现了函数重载。在** main( )**函数中，三次调用** print( )**函数，根据实参的类型来选择不同的** print( )**函数的实现。实参为** 'm' **的** print( )**函数调用** void print(char x)**函数；实参为** m **的** print( )**函数调用** void print(int x)**函数；实参为**?good?**的** print( )**函数调用** void print(char *x)**函数。**

3.6 第 6 章习题答案和解答

3.6.1 习题答案

习题 6.1 (**略**)

习题 6.2 **选择填空**

A D D D C D A A A C A C D

习题 6.3 **判断题**

描述正确的有：(1)，(5)～(11)，(13)。

习题 6.4 **分析程序结果**

(1) 4

(2) 15

(3) 12，10，9，6，5，4，3，2，1，
(4) 4，4，4，4，4
5，5，5，5，5
7，7，7，7，7
8，8，8，8，8
(5) sum=37.86
(6) 5.32，5.32
3.9，3.9
5
(7) a+b=39
(8) 10 11 12 13 14 15 16 17 18 19

习题 6.5　**编程(略)**

3.6.2　习题解答

本章讲述了指针和引用的基本概念和使用方法。指针是 C++**语言和** C **语言都使用的概念，而引用只是** C++**语言中使用的概念。注意掌握指针和引用的区别。**

第 6.2 **题中**(2)**题**

该题中定义了一个指向 int **型变量的指针** p**，并且给它赋了初值** m**。**m **是一个一维的** int **型数组的数组名，其使指针** p **指向数组** m **的首元素。表达式**++*p**，按结合性规定，它等价于**++(*p)**，其含意是取指针** p **所指向的变量的内容后再增** 1**，实际是取数组** m **的首元素** m[0] **的值后再增** 1**，即**++m[0]**。这里，***++p **表达式等价于***(++p)**，其值为** m[1]**；***++m **表达式等价于***(++m)**，其值也为** m[1]**；***p++**表达式等价于***(p++)**，其值为** m[0]**，改变指针** p **指向数组元素** m[1]**。因此，该题应选择** D**。**

第 6.2 **题中**(3)**题**

该题中首先定义了 int **型变量** a **和指向** int **型变量的指针** pa**，又定义了字符型变量** c **和指向字符型变量的指针** pc**。在给指针** pa **的** 4 **种赋值中，**D **是正确的。因为** pa=&c **是错的，不能将一个字符型变量的地址赋给一个** int **型指针；**pa=a **也是错的，不能将一个变量名赋给一个指针；**pa=pc **还是错的，两个不同类型的指针不能互相赋值。表达式**(int *)&c **是将字符型变量** c **的地址值强制成为** int **型变量的地址后，赋给一个指向** int **型变量的指针，这是正确的。**

第 6.2 **题中**(5)**题**

该题定义一个一维数组 a**，并进行了初始化。该数组有** 5 **个元素。又定义一个** int **型指针** p **指向数组** a **的首元素。在给出的** 4 **种表示中，**C **是表示数组元素地址的。**A **和** B **都是对数组元素的地址值再取地址，显然是错的。而** D **是对数组元素的地址值取内容，即为数组元素值。而** C **是对数组元素** p[2] **取地址，即为数组元素的地址值。**

第 6. 2 **题中**(7)**题**

该题中 a **是一个二维数组的名字，**p **是一个指向一维数组的指针名。在给指针** p **赋值的** 4 **种表示中，**A **是正确的。因为** p **是一个指向一维数组的指针，该指针相当于二级指针，因此给** p **赋的地址值应是二级指针的地址值。而** a［1］**是数组** a **的第** 1 **行首元素的地址，即一级指针的地址值；***a **与** a［0］**相同，它也是一级指针的地址值；***a+2 **与&**a［0］［2］**相同，它仍是一个一级指针的地址值。**

第 6. 2 **题中**(8)**题**

该题中 b **是一个二维数组的数组名，**q **是一个一维一级指针数组的数组名，一维一级指针数组名相当于二级指针的地址。该题在** 4 **种给一维一级指针数组的赋值中，**A **是正确的。因为** B**，**C **和** D **中的右值表达式均是一级指针的地址值。**

第 6. 4 **题中**(1)**题**

该程序中定义了一个一维数组 x**，又定义一个指针** p**，并使指针** p **指向一维数组** x **的第** 1 **个元素(即首元素后边的一个元素)。**

程序中*(p+i)**是数组元素** x［i+1］**的指针表示方法。程序中** for **循环共循环** 4 **次，使得变量** b **的值分别为** 1**，**2**，**3**，**4**。**

该程序的特点是关于一维数组的一级指针表示的使用方法。

第 6. 4 **题中**(2)**题**

该程序的主函数中定义了两个字符指针 p1 **和** p2**，函数** fun( )**的两个形参** s1 **和** s2 **也是字符指针。字符指针作函数参数是该程序的特点。**

字符指针的定义格式如下所示：

```
char *p1,*p2;
```

其中，p1**，**p2 **是两个字符指针。字符指针可以用一个字符串直接赋值。例如，**

```
p1=?abcxyr?;
p2=?abcijh?;
```

当字符指针作函数形参时，调用函数的实参可以用字符指针名，也可以用一维字符数组名。该程序使用字符指针 p1 **和** p2 **作** fun( )**函数的实参。**

下面解释该程序中 fun( )**函数的功能。**

while **循环的条件表达式是**

```
*s1  &&  *s2  &&  *s2+ += =*s1+ +
```

其中，s1 **和** s2 **为字符指针，作为该函数的形参。**

当函数调用后，s1 **指针指向** p1 **指针所指向的字符串，**s2 **指针指向** p2 **指针所指向的字符串。开始执行** while **循环时，***s1 **为字符** ’a’**，即非** 0**；***s2 **为字符**’a’**，即非** 0**；***s2+ += = *s1+ + **用来判断***s2 **和***s1 **所指向字符是否相等，并且使字符指针** s1 **和** s2 **都向后移一个字符，即** s1 **指向字符** ’b’**，**s2 **也指向另一个字符串的**’b’**字符。由于** *s1 **和** *s2 **所指向字符相同，即该表达式为非** 0**，整个条件表达式为非** 0**，所以执行一次循环体，该** while **循环的循环体为空语句。**

接着再计算条件表达式，第二次计算条件表达式时，*s1 **为字符** ’b’**，***s2 **为字符** ’b’**，***s2+

+==*s1++为非 0，并使指针 s1，s2 再向后移一个字符。再执行一次循环体。

接着，第三次计算条件表达式的值，*s1 为字符 'c'，*s2 为字符 'c'，*s2+ +==*s1+ +值为非 0，并使指针 s1,s2 再向后移一个字符。执行一次循环体。

接着，第四次计算条件表达式的值，*s1 为字符 'x'，*s2 为字符 'i'，*s2+ += =*s1++表达式值为 0，不再执行循环体，即退出循环。但是 s1 和 s2 都将指向后一个字符。即这时*s1 为字符 'y'，*s2 为字符 'j'。退出循环后，执行下列语句：

```
return *s1−*s2;
```

该语句将返回指针 s1 所指向字符与指针 s2 所指向字符的 ASCII 码值之差，即为'y'-'j'，其值为 15。将该值返回到 main( )，给 n 赋以该值，当输出 n 值时显示为 15。

练习题

可将该程序 fun( )函数中 while 循环改写成如下形式：

```
while(*s1 && *s2 && *s2= =*s1)
{
    s1++;
    s2++;
}
```

这样修改后，请读者分析一下，修改后与原来的程序功能是否一样呢？

另外，原程序的功能是比较两个字符串的字符。从开始字符向后按位置逐一比较，当发现不相同时，则返回该字符后面字符的 ASCII 码之差。修改 while 循环后的程序功能也是这样吗？为什么？

第 6. 4 题中(3)题

该程序特点是用指针表示一维数组元素进行各种操作。该程序只有主函数，函数体内有两种并列的循环语句。开始是 while 循环，后来是 for 循环。

分析后得知，while 循环用来将数组 a 中的元素从后面开始向后移动，当所后移的元素大于 3 时，则不再移动，即退出 while 循环，并将前面移动留出的空位置的元素赋值为 3。再使用 for 循环，将重新改变的数组 a 的各个元素值输出显示，每个元素间用逗号分隔。

练习题

该程序执行输出显示的结果中，在最末尾有一个逗号，如何去掉这个逗号呢？需要对 for 循环语句作哪些修改？请读者修改后并上机验证。

第 6. 4 题中(4)题

该程序中出现的有关指针问题比较多，这里有：

① 二维数组元素的指针表示
② 一维一级指针数组 p
③ 二级指针 pp
④ 指向一维数组的指针 s
⑤ 一级 int 型类指针 q

该程序中出现了对一个二维数组元素的各种不同表示方法。

·使用二维数组的数组名 a 表示：*(a[i］ +j)表示二维数组 a 的第 i 行第 j 列元素；

·使用一维一组指针数组 p 表示：*(*(p+i)+j)表示指针数组 p 的第 i 行第 j 列元素。该指针数组的各个元素分别指向二维数组 a 的各行。

·使用二维指针 pp 表示：(*(pp+i))[j］表示 pp 所指向数组的第 i 行第 j 列元素。pp 是一个指向指针数组 p 的指针。

·使用一级指针 q 表示：*(q+3*i+j)表示指针 q 所指向的二维数组 a 的第 i 行第 j 列元素。指针 q 指向二维数组 a 的首元素。

·使用指向一维数组的指针 s 表示：*(*s+3*i＋j)表示指向一维数组的指针 s 所指向的二维数组 a 的第 i 行第 j 列元素，因为指针 s 指向二维数组 a 的首行。

该程序中出现了各种不同的指针，通过对该程序的分析，学会各种指针的定义和赋值以及它们的使用方法。例如：

```
int *q=&a[0][0];
```

这里，q 是一级的 int 型指针，给它赋值要求是一个 int 型变量的地址，并且是一级指针的地址值。&a[0][0]是二维数组首元素的地址值，它是一级指针地址值，如果写成如下形式：

```
int *q=a;
```

这是错误的，因为 a 是一个二级指针的地址值。

因此，给指针赋值不仅要求类型相同，而且要求级别一致。

第 6. 4 题中(5)题

该程序中出现了指向函数的指针。请注意指向函数的指针的定义格式、赋值以及使用方法。

程序中函数 sum( )有两个形参，它们都是指向函数的指针，该函数说明如下：

```
double sum(double (*g1)(double,double),
                    double (*g2)(double,double));
```

其中，g1 和 g2 分别是两个指向函数的指针，它们所指向的函数是具有两个 double 型参数，并具有 double 型返回值的函数。

给指向函数的参数赋值时，需要用一个相同类型相同参数的函数名，这里的函数名将表示该函数存放在内存中的入口地址。在该程序中，调用 sum( )函数格式如下：

```
sum(g1,g2)
```

其中，g1 和 g2 分别表示两个函数名，这两个函数都是具有两个 double 参数的 double 型函数。这里，指向函数的指针名与函数名使用了相同的名字，完全可以用不同的名字，以免搞混。当调用上述函数时，即将实参函数名 g1 赋给形参指针名 g1，将实参函数名 g2 赋给形参指针名 g2，使得指针 g1 指向函数 g1( )，指针 g2 指向函数 g2( )。

使用指向函数的指针来调用所指向的函数的方法如下：

(*〈指向函数的指针名〉)(〈实参表〉)；

该程序中的下述语句：

```
double c=(*g1)(a,b)+(*g2)(a,b);
```

其将用指向函数的指针名 g1 调用它所指向的函数 g1( )，其实参为 a 和 b；再用指向函数的指针名 g2 调用它所指向的函数 g2( )，并将其返回值相加。然后将其和值赋给 double 型变量 c。

sum( )函数的功能是将上述求得的变量 c 值返回，在主函数中将其返回值输出显示。

关于 g1( )函数和 g2( )函数的定义如该程序所示。g1( )函数用来求出两个参数指定的数值之和，并返回；g2( )函数用来求出两个参数所指定的数值之积，并返回。

第 6. 4 题中(6)题

该程序中使用了引用，这是 C++中引进的一种新概念。简单地理解引用就是一种别名，常用的是变量或对象的引用。引用说明的方法如本程序所示：

```
double &rd=d,&re=e;
```

其中，变量 d 和 e 是已被定义或说明的 double 型变量。而 rd 和 re 是两个被说明的 double 型的引用，它们分别是变量 d 和 e 的别名。值得注意的是，在说明引用时一定要对它进行初始化，即指定对哪个变量或对象的引用。

引用在使用过程中，它的值总是与被引用的变量值相一致的。例如，rd 是变量 d 的引用，如果 d 值为 3.98，则 rd 值也为 3.98；当 rd 值被更改为 2.56，则 d 的也为 2.56。

引用经常被作为函数的形参，调用时实参使用相同类型的变量名。使用引用作形参可以实现在被调用函数中改变调用函数实参的目的。另外，引用还可作函数的返回值。这两点将会在后面的程序中看到。

第 6. 2 题中(7)题

该程序的特点是引用作函数参数。该程序中 fun( )函数说明如下；

```
void fun(int,int&);
```

该函数的第二个参数是 int 型变量的引用，程序中调用该函数的格式如下：

```
fun(5,a);
```

其中，a 是一个 int 型变量名。调用 fun( )函数后，j 将成为 a 的别名，即 j 是变量 a 的引用。在 fun( )函数体内，j 值被改变为 15，于是调用函数中实参 a 的值也改变为 15。这就是引用调用的特点。

程序中通过执行下列调用：

```
fun(8,b);
```

使变量 b 的值改变为 24。

第 6. 4 题中(8)题

该程序的特点是引用作为函数的返回值。程序中出现的 fun( )函数被说明如下：

```
int & fun(int);
```

该函数返回值是 int 型变量的引用。

在程序中，fun( )函数调用作为一个循环体，其格式如下：

```
for(int i(0);i<10;i++)
    fun(i)=a+I;
```

这里，函数调用 fun(i)作为赋值表达式的左值。每次调用 fun( )函数时，总对它的返回值进行赋值。而 fun( )函数的返回值是 int 型变量的引用，可以作赋值表达式的左值。因此，上述 for 循环是用来给 int 型一维数组 aa 的 10 个元素赋值的。程序中又通过另一个 for 循环输出显示数组 aa 各个元素的值。

另外，程序的 main( )中定义的变量 a 是自动类。它与 fun( )函数的形参 a 毫无关系，它们可以用不同的变量名。

3.7 第 7 章习题答案和解答

3.7.1 习题答案

习题 7.1 (略)

习题 7.2 选择填空

D B C A C

习题 7.3 判断题

描述正确的有(1)，(2)，(6)，(10)

习题 7.4 分析程序的输出结果

(1) Ma ping,95.5
(2) Ma 89.5
(3) 6
20
21
22
(4) d size---＞8
d.d size---＞8
abc size---＞8
d.i=97
(5) word value:4241
low value:A
high value:B

习题 7.5 编程(略)

3.7.2 习题解答

第 7.2 题中(2)题

在给定的关于结构变量的定义中，共有两处错。一是同一结构的成员名字不能相同，这里 int x 和 double x 同名是错的。二是定义了结构变量 x,y,z 后应用分号结束，这里少一个分号是错的。至于结构没有名字是允许的，程序中允许出现无名结构。另外，结构成员可以与结构变量具有相同名字。因此，结构成员与结构变量重名是允许的。

第 7.4 题中(1)题

该程序中定义一个结构，其名为 student1，又定义了一个指向该结构变量的指针 p。在主函数中，定义一个结构变量 s1，并且通过赋值，使指针 p 指向结构变量 s1。

程序中给指向结构变量的指针 p 所指向的结构变量赋值。先给该结构中的指针成员赋一个地址值，具体格式如下：

```
p->name=(char *)new char[50];
```

该语句使用运算符 new，在内存中开辟一个可放 50 个字符大小的单元，并将其首地址强制成为 char 型，然后将它赋值给指针 p 所指向的结构变量的指针成员 name。

接着，使用下列格式给指针 p 所指向的结构变量的 name 成员赋值，由于所赋的值是一个字符串，因此采用如下格式：

```
strcpy(p->name,?Ma ping?);
```

这里，函数 strcpy( )是字符串拷贝函数，被包含在文件 string.h 中。该函数的功能是将字符串“ Ma ping ”拷贝到 p 指针所指向的结构变量的 name 成员中。

然后，又给指针 p 所指向的结构变量的另一个成员 score 赋值。格式如下：

```
p->score=95.5;
```

程序中，最后输出指针 p 所指向的结构变量的两个成员值，一个是 name 中的字符串，另一个是 score 中的 double 型值。其中，(*p).score 与 p->score 是等价的。

读者分析该程序后，可能提出这样的问题：主函数中开始定义的结构变量 s1 并没有用过，并且也没有必要让指针 p 指向结构变量 s1。按这种想法，可以将主函数中开始的两条语句去掉。如果真的去掉后，再编译、运行程序时，将会发现是有错误的。其错误原因在于指针 p 没有赋值就被使用了。这种错误是不允许的，有些编译系统提示要关闭程序。这种对指针不赋值就使用的错误是很容易犯的。这一点请读者特别注意，这也是在 C++中尽量少用指针的原因。同样的，对于该结构中指针成员 name 也是如此，必须先给它赋值后，再使用它，否则也会出现类似错误。当然，对指针 p 的赋值方法可用如下形式：

```
p=(strudent1 *)new student1;
```

这样赋值的话，程序 main( )中定义 s1 就没有必要了。请读者试一下。

第 7. 4 题中(2)题

该程序中使用了指向结构变量的指针作函数参数。例如，fun( )函数的形参就是指向结构变量的指针：

```
void fun(struct student2 *s)
{
        …
}
```

fun( )函数的形参 s 是一个指向结构模式 student2 的结构变量的指针。

在调用 fun( )函数时，实参使用地址值，如下语句所示：

```
fun(s+1);
```

其中，s 是一个结构数组名，该数组有 4 个元素，每个元素是指向 student2 结构模式的结构变量。s+1 表示数组 s 的第 1 个元素(第 0 个元素数起)的首地址值。

该程序是通过调用 fun( )函数,输出结构数组中第 1 个元素的成员 name 的值和成员 score 的值。使用指向结构变量的指针 s 表示它的两个成员分别是 s->name 和(*s).score，其值分别是 Ma 和 89.5。

第 7. 4 题中(3)题

该程序中定义了一个结构，其名为 abc。该结构中有一个指针成员。程序中定义了指向该结构变量的指针 p。

在 main( )函数中，定义了一个有两个元素的结构数组 a，并对它进行了初始化。又使指针 p 指向数组 a 的首元素。程序中通过指针 p 表示数组 a 中的元素和成员，进行输出。

p->b 表示数组 a 中首元素的 b 成员中的内容，因为它等价于(p->b)，即数组 x 中首元素 6。

(*p).a 等价于 p->a，表示数组 a 中首元素的 a 成员的值，即 20。

++p->a 等价于++(p->a)，表示对于数组 a 中首元素的 a 成员的增 1，即为 21。

++(*p).a 等价于++(p->a)，表示对于数组 a 中首元素的 a 成员值再增 1，即为 22。前一次增 1 后的 a 值保持为 21。

练习题

读者还可以试分析下列语句的输出结果；分析后并上机试一下。

```
cout<<(++p) ->a<<endl;
cout<<*(p->b+1)<<endl;
```

第 7.4 题中(4)题

该程序中出现了联合变量，程序中开始定义了一个名为 abc 的联合模式，并定义了一个联合变量 d。

主函数中，先给联合变量的字符型成员 c 赋一个值'a'。然后，输出下述 4 个表示式的值。

・sizeof(d)表示联合变量 d 占内存的大小。联合变量 d 有 3 个成员，其中数据类型最高的(占内存空间最大的)是 double 型，在 16 位机中占 8 个字节，因此，d 的大小为 8。

・sizeof(d.d)表示联合变量 d 的 double 型成员 d 占内存的字节数，显示为 8。

・sizeof(abc)表示具有联合模式 abc 的联合变量占内存的字节数，即 8。

・0x000000ff&d.i 的值是一个 int 型数，由于当前变量 d 中存放的是'a'，将它按 int 型的低 16 位输出，即 97。

通过该程序的分析,可验证如下问题：

① 联合变量的若干成员是共址的，在某个时刻内只能保存某个成员的值；

② 联合变量所占内存的字节数是该联合模式中类型最高的成员所占内存的字节数。

第 7.4 题中(5)题

该程序中既有结构变量，又有联合变量，并且结构变量作为联合变量的成员。程序中，联合模式 word1 的一个成员是结构模式 tag 的结构变量 byte。实际中，结构变量可作为联合成员，联合变量也可作为结构成员。

主函数中，给联合变量 W 中的 word 成员赋了值，然后，用联合变量 W 来输出该联合变量的各个成员值，包含结构变量成员的两个成员值。

程序中包含了文件 iomanip.h，该文件在教材的第 13 章中讲述。由于该程序中出现 hex 算子，它表示后面表达式值按十六进制输出，因此，需要包含 iomanip.h 头文件。

3.8 第8章习题答案和解答

3.8.1 习题答案

习题 8.1 (**略**)

习题 8.2 **选择填空**

D C B A C A B B A

习题 8.3 **判断题**

描述正确的有(1) (3) (5) (8) (10)

习题 8.4 **分析程序结果**

(1) Default constructor called.
Constructor called.
a1=0,a2=0
a1=3,a2=9

(2) b1=12,b2=65
a=100

(3) 1
2
3
4
5
5
4
3
2
1

(4) m1=68,m2=156

(5) 1
{ }
{1，2，3，4，5，6，7，8，9，10}
1
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}
{25，24，23，22，21，20，19，18，17，16，15，14，13，12，11，10，9，8，7，6，5，4，3，2，1}

3.8.2 习题解答

第 8.4 题中(1)题

该程序中定义了一个类 A，该类中有 3 个公有的成员函数和 2 个私有的数据成员。在类体外，先后定义了 3 个成员函数，其中，A( )函数是缺省的构造函数。这里要说明的是，一个类中可定义多个构造函数，它们都是重载函数，因为它们有相同的函数名。其中可以定义一个无形参的构造函数，该构造函数称缺省构造函数。如果某个类中没有定义任何构造函数，则系统自动提供一个缺省的构造函数。

在类体外又定义了一个具有 2 个参数的构造函数 A( )。还定义一个 print( )函数，用来输出显示该类中的 2 个私有的数据成员的值。

在 main( )中定义了两个 A 类的对象 x 和 y。在创建对象时，系统将自动调用相应的构造函数给对象进行初始化。在创建对象 x 时，系统调用无参数的缺省构造函数，给对象 x 初始化，使得对象 x 的两个私有数据成员都获得 0 值，并且在屏幕上输出显示如下信息：

Default constructor called.

在创建对象 y 时，系统调用 2 个参数的构造函数，将数值 3 和 9 分别赋给对象的两个私有数据成员 a1 和 a2，并在屏幕上输出显示如下信息：

Coustructor called.

接着，主函数中调用成员函数 print( )，输出对象 x 和 y 的私有数据成员的值。

本程序比较简单，通过分析该程序进一步搞清楚下列有关类和对象概念的具体实现：

① 定义一个类。包括使用的关键字和指定的类名，标识公有成员和私有成员的方法，成员函数和数据成员的说明，成员函数的定义。其既可在类体内，又可在类体外。

② 创建对象。在创建一个对象时，系统根据创建时指定的参数，自动调用相对应的构造函数进行初始化。

③ 构造函数的定义格式。本程序中有 2 个构造函数。

④ 调用成员函数输出某个对象的私有数据成员的值。

第 8.4 题中(2)题

该程序中定义了两个类：A 类和 B 类。这两个类的关系在于 B 类的对象 c 作为 A 类的一个数据成员，即 A 类中包含了 B 类的一个子对象。

在 B 类中，定义了一个缺省构造函数和一个带有 2 个参数的构造函数，还有一个输出数据成员值的函数 printB( )。另外，有 2 个私有的数据成员 b1 和 b2，都是 int 型变量。

在 A 类中，同样定义一个缺省的构造函数和一个带有 3 个参数的构造函数，还有一个输出数据成员值的函数 printA( )。另外，有 2 个私有的数据成员，一个是 int 型变量 a，另一个是 B 类的子对象 c。

由于 A 类中有一个 B 类的对象作为数据成员，因此，在创建 A 类的对象时就应该考虑到如何给子对象赋值。于是，A 类中出现了如下所示的构造函数：

```
A(int i,int j,int k):c(i,j),a(k)
{ }
```

该构造函数中有 3 个参数，其中前两个参数是用来给子对象 c 初始化的，后一个参数用来初始化 A 类的数据成员 int 型变量 a。上式中，c(i,j)表示对数据成员子对象 c 初始化，调用 B 类的两个参数的构造函数来实现；a(k)表示用参数 k 给 A 类的私有数据成员 a 初始化。上述的构造函数也可写成如下形式：

```
A(int i,int j,int k):c(i,j)
{
    a=k;
}
```

两者的区别在于构造函数中的成员初始化表(即冒号后面的部分)中的项数不同。前者的成员初始化表中有两项，而后者仅有一项，即另一项被放到函数体内处理了。

请读者记住，成员初始化表在后面章节中还会遇到。这里的子对象初始化应在成员初始化中进行。

在主函数中，创建一个对象 x，给定 3 个参数，调用带有 3 个参数的构造函数对该对象初始化后，使得对象的 int 型数据成员 a 数值为 100，子对象 c 的两个成员分别为 12 和 65。

第 8.4 题中(3)题

该程序中定义一个类 C。该类中又定义了静态成员：一个静态成员函数 Getc( )和一个静态数据成员 c。□

静态成员函数使用关键字 static 来标识，该函数可以直接引用静态数据成员。

静态数据成员也使用关键字 static 来标识，由于静态数据成员不是属于某一个对象的，而是属于整个类的所有对象的，类似于各个对象中的全局变量。因此，静态数据成员不是调用构造函数进行初始化，而是需要在类体外，在使用它之前进行初始化的，其格式如下：

〈类型说明符〉〈类名〉::〈静态数据成员名〉=〈初值〉；

该程序中，静态数据成员 c 被初始化的格式如下：

```
int C::c=0;
```

这里，由于初始化在类体外进行，需要使用作用域运算符来限定该成员是属于哪个类的。在 C 类中，定义了一个缺省构造函数和一个析构函数。

在 main( )中，创建了 5 个 C 类的对象 c1，c2，c3，c4 和 c5，系统将调用缺省构造函数。每调用一次缺省构造函数，将输出++c 的值，由于 c 被初始化为 0，5 次后调用依次输出++c 的值为 1，2，3，4，5。最后，静态数据成员 c 的值就为 5。主函数中，执行下列语句：

```
cout<<C::Getc( )<<endl;
```

即输出调用 C 类中静态成员函数 Getc( )的返回值，该静态成员函数返回当前静态数据成员 c 的值，为 5。

在结束 main( )之前，要析构所创建的 5 个对象，系统将自动调用析构函数来实现。每调用一次析构函数将在屏幕上输出 c--的值，由于 c 在开始时值为 5，因此，5 次调用析构函数输出后分别为 5，4，3，2，1。

第 8.4 题中(4)题

该程序中定义了一个类 M。在 M 类中，定义了一个具有两个参数的构造函数：一个是

一般成员函数 Sum( )，它用来对两个对象中的成员进行运算；另一个是一般成员函数 print( )，用来输出某个对象的两个数据成员的值。M 类中又定义了两个 int 型的变量 m1 和 m2 作为私有的数据成员。

在 main( )函数中，创建 3 个对象 a,b 和 c。创建对象 a 时，自动调用两个参数的构造函数，使 a 的两个私有数据成员分别获得值为 34 和 78。创建对象 b 和 c 时，自动调用缺省的拷贝初始化构造函数，使得对象 b 和 c 具有与对象 a 相同的数据成员的值。在 main( )中，通过调用成员函数 Sum( )来重新改变对象 c 的数据成员的值，改变的方法由该函数体内两条语句给出。主函数内最后调用成员函数 print( )输出对象 c 的两个数据成员的值。

第 8. 4 题中(5)题

该程序的功能是对一个 int 型数组中的若干元素作如下操作：

① 将数组置空。使用 Empty( )函数。

② 判断数组是否被置空。使用 IsEmpty( )函数。

③ 判断已知数据是否在数组中存在。使用 IsMemberOf( )函数。

④ 向数组增添一个数组中没有的数据，并加到该数组的末尾。使用 Add( )函数。

⑤ 输出数组中所有的数据。使用 Print( )函数。

⑥ 将数组中各个元素值逆向排列。使用 reverse( )函数

此外，还有两个构造函数：一个是缺省构造函数 Set( )，另一个是拷贝初始化构造函数 Set(set &s)，它们用来创建该类的对象。

在类 Set 中，定义了 8 个成员函数，除了两个构造函数外，其余 6 个函数都用来实现上述 6 个功能。在该类中还定义了两个数组成员：一个是 int 型数组 elems[ ]，另一个是 int 型变量 PC。

在主函数中，先创建一个对象 A，自动调用缺省构造函数。接着，用对象 A 调用成员函数 IsEmpty( )，该函数返回值为 0，说明对象 A 是一个空数组，即数组中无有效元素，输出对象 A，其结果为{ }。

然后，又创建一个对象 B，自动调用缺省构造函数。接着，使用 for 循环，通过调用成员函数 Add( )，给对象 B 的数组成员中添加数据。for 循环的循环变量从 1 开始，每次增 1 直到 10，共循环 10 次，使对象 B 的数组中具有从 1～10 的 10 个元素值。输出该对象中数组元素值为{1，2，…10}。接着，调用 IsMemberOf( )函数查找一下对象 B 的数组成员中是否有数值为 5 的元素，B.Is MemberOf(5)的返回值为 1，说明有数值为 5 的元素。又调用 IsEmpty( )函数判断一下对象 B 的数组成员是否是空，B.IsEmptyOf( )返回值为 0，说明该数组不是空的。

再接着，又用 for 循环向对象 B 的数组成员中添加数据项，从 11 到 25，使得 B 的数组成员中有 25 个元素，其值分别为 1 至 25。执行下列语句：

```
SetC(B);
```

创建一个新的对象 C，该对象与 B 对象相同，即数据成员的值相等。上述语句是调用拷贝初始化构造函数创建一个与已有对象相同的新对象。接着，将该对象 C 的数组成员中各元素值输出显示如下：

{1,2,3,4,5,…24,25}

然后调用成员函数 reverse( )将对象 C 的数组成员中各个元素逆向排列，排列后再将其各元素值输出显示，其结果如下：

{25,24,23,22,21,…2,1}

通过对该程序的分析，读者将会体会到使用 C++编程解决实际问题的方法。归结起来其步骤如下：

① 用 C++中提供的手段(类及对象等)来描述客观对象。例如，本例中的若干个 int 型变量。

② 编写函数实现所需的操作功能。例如，本例前面讲过的 6 个操作功能，每种对应一个函数。

③ 根据实际需要编写主函数，使用前面定义的功能，完成所需的操作。

读者可以看到，本程序中的操作还很不完备，还有很多操作没有包括进去，读者如有兴趣的话，还可以增加一些新操作。例如：

- **对数组中的各个数据项进行排序。**
- **将两个不同对象的数组成员的元素进行合并，并去掉相同的。**
- **删除某个对象的数组成员中的某个数组项。**

3.9 第 9 章习题答案和解答

3.9.1 习题答案

习题 9.1 (**略**)

习题 9.2 **选择填空**

C C B A B B A D B D B D

习题 9.3 **判断题**

描述正确的有(2)(4)**至**(10)

习题 9.4 **分析程序结果**

(1) Starting1：

```
Default constructor called.
Default constructor called.
Default constructor called.
Ending1:
Starting2:
Constructor:a=5,b=6
Constructor:a=7,b=8
Constructor:a=9,b=10
Ending2:
Destructor called.a=9,b=10
Destructor called.a=7,b=8
Destructor called.a=5,b=6
Destructor called.a=5,b=6
Destructor called.a=3,b=4
```

Desfrucfor called.a=1,b=2

(2) Default consfrucfor called.

Default consfrucfor called.

Default consfructor called.

Default construcfor colled.

Destructor called.

Constructor 1 called.

Destrurtor called.

Constructor 2 called.

Destructor called.

x=0,y=0

x=7,y=0

x=5,y=9

Desfrucfor called.

Desfrucfor called.

Desfrucfor called.

(3) Consfrucfor called.0

Consfrucfor called.5

Desfrucfor called.5

5

Desfrucfor called.5

(4) Consfrucfor called.10

10

Destrucfor called.10

(5) Default consfrucfor called.

Consfrucfor:real=6,imag=9

Consfrucfor:real=12.2，imag=25.8

0+0i

6.9+0i

12.2+25.8i

Consfrucfor:real=0.8，imag=0.5

Consfrucfor:real=55,imag=0

Default consfrucfor called.

0.8+0.5i

55+0i

0+0i

习题 9.5 **分析程序，回答问题。**

(1) **答：该程序中调用了三个字符串处理函数：**strlen( )，strcpy( )**和** strcat( )**。它们都包含在** string.h **文件中。**

(2) **答：该程序的** String **类中使用了函数重载方法。对构造函数进行了重载，下述两个**

函数是重载构造函数：

String( )和 String(const char *str)

它们一个无参数，另一个有一个参数。

(3) 答：Setc( )函数的功能是改变字符数组 Buffer[ ]中指定元素的字符值。

(4) 答：Getc( )函数的功能是输出显示字符数组中指定元素的字符值。

(5) 答：Append( )函数的功能是将指定的字符串添加到某个对象的字符数组成员中去。

(6) 答：可以。

(7) 答：该程序中两处用了 new 运算符。

(8) 答：输出结果如下：

```
empty
a string
9
a stping
i
this a string
```

3.9.2 习题解答

第 9.2 题中(2)题

该题是关于通过指向类的数据成员的指针给类数据成员赋值的问题。题中，p 是一个指向类 A 数据成员 m 的指针，a 是类 A 的一个对象，判断给数据成员 m 赋值为 5 的哪个表达式是正确的。使用指向类数据成员的指针表示该数据成员的方法是通过使用“.*”运算符。例如，对象 a 的 m 成员，通过指向 m 成员指针 p 表示如下：

```
a.*p
```

对 m 成员赋值为 5，应表示为 a.*p=5。

第 9.2 题中(6)题

该题是关于常指针问题。常指针分两种情况。

一种是地址不变的指针常量，该指针所指向的对象可以改变。例如，

```
char * const ptr;
```

其中，ptr 是一个指向 char 型变量的指针常量。ptr 值不能被改变。

另一种是指针指向的对象不能改变，而指针地址可以改变。例如，

```
const char *pt;
```

其中，pt 是一个指向 char 型变量的常量指针。pt 所指向的 char 型变量不能被改变。

第 9.2 题中(12)题

该题是关于转换函数的概念。转换函数是类的一种特殊的成员函数。它具有成员函数的特征，其功能是用来进行类型转换的，即将它被定义的类的类型转换为转换函数名标识的类型，转换规则由转换函数的函数体给出。定义转换函数的格式如下：

operator〈转换函数名〉( )
{
〈函数体〉
}

该函数不带参数，不加类型说明，在函数名前加关键字 operator。

该题的 4 种描述中，D 是错误的，其余 3 种是正确的。

第 9. 4 题中(1)题

该程序中定义一个类 A，该类中定义了两个构造函数和一个析构函数，还有一个成员函数 Set( )，该函数的功能是用来改变某个类 A 对象的数据成员的值。该类中定义了两个私有数据成员，它们是 int 型变量。在每个构造函数和析构函数的定义中都有一条输出语句，不仅输出一个字符串，而且输出对象的数据成员值。这条输出语句用来标识所创建和所释放的对象。

在 main( )函数中，先输出一个字符串。接着，创建一个具有 3 个元素的对象数组 a[3]，自动调用三次缺省的构造函数，屏幕上显示出三行如下信息：

Default constructor called.

然后，通过使用 for 循环，调用成员函数 Set( )，改变对象数组 a[i]中三个元素的值，使 a[0]对象的两个数据成员 a 和 b 的值为 1 和 2，使 a[1]对象的两个数据成员 a 和 b 的值为 3 和 4，使 a[2]对象的两个数据成员 a 和 b 的值为 5 和 6。再输出字符串“ Ending1 :”。

接着，输出字符串“ Starting2 :”。又创建一个对象数组 b[3]，并使用初始化值表给该数据初始化，使得对象 b[0]的两个数据成员 a 和 b 获初值为 5 和 6，使得对象 b[1]的两个数据成员 a 和 b 获初值为 7 和 8，使得对象 b[2]的两个数据成员 a 和 b 获初值为 9 和 10。这时将调用三次带有两个参数的构造函数，屏幕上将显示出三条输出信息。在程序退出前，需要将所创建的所有没有析构的对象进行析构，通过自动调用析构函数来实现。

第 9. 4 题中(2)题

该程序中定义一个 B 类，在该类中定义了三个构造函数：一个是缺省的，一个是带一个参数的，另一个是带两个参数的。还定义一个析构函数。每个函数在定义中都有一条输出不同字符串的语句。

执行主函数时，先定义一个指向类 B 对象的指针 p。然后，通过使用运算符 new 创建一个对象数组，该对象数组有 3 个元素，并将该数组的首地址值赋给对象指针 p，即让指针 p 指向该一维对象数组的首元素。接着，分别给指针 p 所指向的对象数组的三个元素赋值。系统将分别自动调用缺省构造函数、一个参数的构造函数和两个参数的构造函数来实现对创建的三个对象初始化。再接着，通过 for 循环调用成员函数 Print( )，输出各个对象的数据成员的值，这里 for 循环执行三次循环体，每次输出对象数组 B 中的一个元素的数据成员值。最后，使用运算符 delete 释放用 new 所创建的三个对象。

下面分析该程序的输出显示信息。

首先，使用运算符 new 创建对象数组的三个元素时，自动调用三次缺省构造函数，输出前三行信息。接着，调用缺省构造函数创建一个临时对象，给指针 p 指向的数组的第 0 个元素赋值，然后临时对象被释放，自动调用析构函数。于是在屏幕显示两行信息，它们分别是

调用缺省构造函数和析构函数时函数体内的输出信息。同样地，调用一个参数的构造函数创建临时对象，并给 p[1]赋值，再释放该临时对象，于是又显示出两行信息。再调用两个参数的构造函数创建临时对象，并给 p[2]赋值，再释放该临时对象，又显示出两行信息。执行 for 循环输出对象数组 B 的三个元素的数据成员值，屏幕上又将显示三行信息。最后，执行 delete [ ]p 语句时，释放指针 p 所指向的数组的三个元素时，调用三次析构函数，输出最末三行信息。

第 9.4 题中(3)题

该程序特点是使用了常类型说明符 const 定义常量和常函数。

在该程序所定义的类 A 中，成员函数 Print( )是一个常函数，定义格式如下：

```
void Print( ) const
{
    …
}
```

在 main( )中，定义一个 int 型常量 N，定义格式如下：

```
const int N=5;
```

该语句表示定义 int 型常量 N 的值为 5。

在主函数中，创建一个 A 类的对象 my，这时，自动调用一个参数的构造函数，输出一行信息。因为一个参数的构造函数中设置了参数缺省值，因此，可以这样调用。

执行下列语句时，

```
my=N
```

将一个常量 N 赋值给 A 类对象 my。这时，调用一个参数的构造函数将 int 型常量 N 转换成一个临时对象，将该临时对象的的值赋给对象 my，并将其临时对象释放。释放临时对象时，调用其析构函数，于是屏幕上显示两行信息。在执行下列语句时，

```
my.Print( );
```

输出对象 my 的数据成员值 5。退出程序前，调用析构函数释放对象 my，输出一行信息。

第 9.4 题中(4)题

该程序中先定义一个类 B，该类体中定义一个构造函数，一个析构函数和一个输出显示数据成员值的 Print( )函数；又定义一个私有的数据成员 m。

程序中定义一个普通函数 fun( )，该函数的参数是常引用，该引用是类 B 对象的引用。

主函数中只有一条调用 fun( )函数的语句。执行该语句时，调用函数的实参为 int 型常量 10，用它给形参常对象引用赋值，调用构造函数创建一个对象给引用赋值，输出构造函数中的一行信息。执行 fun( )函数的函数体，输出显示该对象的数据成员值，显示该函数前，将所创建的对象释放，调用析构函数来实现，输出析构函数中的一行信息。

第 9.4 题中(5)题

该程序是一个有关复数表示方法的程序。在类 complex 中，定义了三个构造函数，用来创建不同形式的复数。类中还定义两个一般的成员函数，一个是 Print( )函数，用来按指定格式输出显示复数；另一个是 Set( )函数，用来给对象的数据成员赋值。

分析主函数和输出结果。

首先创建对象 c1，自动调用缺省的构造函数来实现，输出显示一行信息。再创建对象 c2，自动调用一个参数的构造函数来实现，并输出显示一行信息。再创建对象 c3，自动调用带有两个参数的构造函数，并输出显示一行信息。然后，将所创建的三个对象，调用 Print( )，函数按其复数形式输出显示如下：

```
0+0i
6.9+0i
12.2+25.8i
```

第 9. 5 题

该程序中定义一个名为 String 类。该类中定义了一些关于字符串的操作。类中首先定义了一个缺省的构造函数和一个带有一个参数的构造函数。接着又定义了如下的成员函数：

① 函数 Setc( )，改变一个字符。

② 函数 Getc( )，获取一个字符。

③ 函数 Getlength( ),获取对象的字符指针成员中存放的字符串长度。

④ 函数 Print( )，输出对象的字符指针成员中存放的字符串。

⑤ 函数 Append( )，将一个字符串追加到某对象的字符指针成员中所存放的字符串的尾部的。

该表中定义了两个数据成员：一个是 int 型变量 Length，用来存放字符串的长度；另一个是字符指针 Buffer，用来存放指定长度的字符串。该类中还定义一个析构函数。

下面分析该程序中主函数的输出信息。

先创建两个对象 s0 和 s1，创建 s0 时调用缺省的构造函数，创建 s1 时调用一个参数的构造函数。接着，调用成员函数 Print( )，分别输出显示对象 s0、对象 s1 的字符指针成员中所存放的字符串，其结果如下：

```
empty.
a string.
```

然后，输出对象 s1 中字符串的长度，调用 GetLength( )函数，其结果为 9。

再接着，调用 Setc( )函数将对象 s1 的字符串中第 5 个字符改写为'p'，再输出对象 s1 中的字符串：

```
a stping.
```

又调用 Getc( )函数获取对象 s1 中字符指针成员中所存放的字符串的第 6 个字符，并输出显示在屏幕上，其结果为字符 i。

再接着，调用一个参数的构造函数创建对象 s2，再调用 Append( )函数将一个字符串“a string.”追加到对象 s2 的字符指针成员中所存放的字符串“this□”的尾部。

最后，输出对象 s2 和字符指针成员中所存放的字符串如下所示：

```
this a string.
```

3.10 第 10 章习题答案和解答

3.10.1 习题答案

习题 10.1 (**略**)

习题 10.2 **选择填空**

C A A D B

习题 10.3 **判断题**

描述正确的有(1)(4)(5)(6)(10)

习题 10.4 **分析程序结果**

(1) (1，2)
5，6
(10，12)

(2) (1，2)
(6，9)
5，6
(6，9)

(3) (13，22，30，40)

(4) D2 ∷ display( )
pri1=4, pri2=5
pri4=6
pri12=7
D2::display( )
pri1=4,pri2=5
pri4=7
pri12=8

(5) D2::display( )
pri1=1,pri2=4
pri4=6
pri12=7
D2::display( )
pri1=3,pri2=4ı pri4=7ı pri12=8

3.10.2 习题解答

第 10.4 题中(1)题

该程序中定义了两个类：类 A 和类 B，并且类 B 是类 A 的私有继承的派生类。根据私有继承方式的特点，基类中的公有成员在派生类中变成私有成员。但在派生类中，使用派生类的成员函数调用基类中的公有成员函数还是允许的。

在 main( )中，先调用 A 类中的构造函数创建一个 A 类对象 a，再调用 A 类中成员函数 Show( )输出对象 a 的两个数据成员值为(1，2)。

接着，又调用 B 类的构造函数创建一个 B 类的对象 b，该对象具有 4 个数据成员，其中两个是该类本身的，另两个是该类的基类 A 中的两个数据成员。再用 b 对象调用 B 类中的成员函数 fun( )，该函数又调用 A 类中的 Move( )函数，用来更改该对象的两个基类数据成员的值。接着，b 对象调用 B 类中的 Show( )函数，输出显示 b 对象本身的两个数据成员值，即 5，6。再接着，b 对象调用 B 类中的 f1( )函数，该函数又调用 A 类中的 Show( )函数，于是在屏幕上输出显示对象 b 的基类中的两个数据成员，其结果为(10，12)。这两个值是在调用 fun( )函数后被改变过的。

第 10.4 题中(2)题

该程序中，B 类是公有继承基类 A 的派生类。该程序与上一个程序比较，类 A 和类 B 的定义中有很多相似之处，但应注意两个程序中 B 类继承 A 类的继承方式是不相同的。另外，两个程序中，B 类的构造函数的定义形式不同，但是两者是等价的。

在该程序的主函数中，先调用 A 类的构造函数创造一个对象 a，然后调用 A 类中的 Show( ) 函数，输出显示对象 a 的两个数据成员的值为(1，2)。接着，调用 B 类的构造函数创建 B 类对象 b，再用对象 b 调用 B 类中成员函数 fun( )，该函数又调用 A 类中的 Move( )函数，用其来更改对象 b 的基类中的两个数据成员的值。再接着，对象 b 调用 A 类中的 Show( )函数，输出 b 对象的基类中两个数据成员的值为(6，9)；又用对象 b 调用 B 类中 Show( )函数，输出 b 对象本身的两个数据成员的结构为 5，6。最后，b 对象调用 B 类中的 f1( )函数，该函数又调用 A 类中的 Show( )函数，输出对象 b 的基类中的两个数据成员的值。

第 10.4 题中(3)题

该程序中定义了三个类：L 类、R 类和 V 类。它们之间具有如下的继承关系：R 类是公有继承 L 类的派生类，V 类是公有继承 R 类的派生类。

在主函数中，先调用缺省构造函数创建了一个对象 v。接着，用对象 v 调用 R 类中成员函数 InitR( )，改变该对象直接基类中两个数据成员 W 和 H，以及改变间接基类中两个数据成员 X 和 Y 的值。然后，再用对象 v 调用 V 类中的 fun( )函数，该函数又调用 L 类中的成员函数 Move( )，用来更改 v 对象的间接基类中的两个数据成员 X 和 Y 的值。最后，使用输出语句输出显示对象 v 的 4 个数据成员的值，分别调用了 L 类的成员函数 GetX( )和 GetY( )、R 类的成员函数 GetW( )和 GetH( )，其结果为(13，22，30，40)。

第 10. 4 **题中**(4)**题**

该程序中先定义了 4 **个类：**P **类、**D1 **类、**D2 **类和** D12 **类。它们之间关系如下：**P **类分别是** D1 **类和** D2 **类的直接基类，它们都是公有继承的；**D12 **类是一个派生类，它有两个直接基类** D1 **类和** D2 **类，它们分别是私有继承方式和公有继承方式。**

D12 **类的构造函数带有** 7 **个参数，它包含了两个基类的构造函数中各** 3 **个参数和该派生类的一个参数。**

在主函数中，先调用 D12 **类中构造函数创建一个** D12 **类的对象** d**。这时，对象** d **中应有** 7 **个数据成员，除了它本身的一个名为** pri12**、值为** 7 **以外，从基类** D1 **中继承下来的有三个：**D1 **类数据成员** pri3(**其值** 3)**，**D1 **类的基类** p **中的两个数据成员** pri1(**其值为** 1)**和** pri2(**其值为** 2)**。从基类** D2 **中继承下来的有三个：**D2 **类的数据成员** pri4(**其值为** 6)**，**D2 **类的基类** P **中的两个数据成员** pri1(**其值被改为** 4)**和** pri2(**其值被改为** 5)**。接着，对象** d **调用** D12 **类中的成员函数** display( )**，该函数先输出如下字符串：**

```
D2::display( )
```

再调用 D2 **类中的成员函数** display( )**，该函数中先调用** P **类中的** display( )**，输出如下结果：**

```
pri1=4,pri2=5
```

返回后继续执行类 D2 **中的** display( )**函数，输出如下结果：**

```
pri4=6
```

再返回后继续执行类 D12 **中的** display()**函数，显示如下结果：**

```
pri12=7
```

在主函数中，接着用对象 d **调用** D2 **类中的** inc4( )**函数，因为** D12 **类中没有** inc4( )**函数，所以使** D2 **类中数据成员** pri4 **增** 1**。对象** d **再调用** D12 **类中的** inc5( )**函数，使得** D12 **类中数据成员** pri12 **增** 1**。然后，对象** d **调用** D12 **类中的** display( )**函数，与上一次调用** display( )**函数相似，输出如下结果：**

```
D2::display( )
pri1=4,pri2=5
pri4=7
pri12=8
```

与上一次不同的是 pri4 **和** pri12 **的值都被增了** 1**。**

第 10. 4 **题中**(5)**题**

该程序与(4)**题中程序有许多相似之处，其最大的不同就是在该程序中使用了虚基类，使得** D1 **类和** D2 **类使用了同一个基类** P**，而不是像**(4)**题中使用了两个相同的基数** P**。值得注意的是，**D12 **类的构造函数的成员初始值表中增加了** P(p11,p21)**项，这是虚基类** P **的构造函数；而在调用** D12 **类的构造函数时，应该先执行虚基类的构造函数，并且在** D1 **类和** D2 **类的构造函数中不再调用虚基类** P **的构造函数，即虚基类的构造函数只被调用一次。这便是使用虚基类带来的好处。**

在主函数中，首先调用 D12 **类的构造函数创建对象** d**。这时，对象** d **的数据成员中，**pri12 **的值为** 7**，**D1 **类中的** pri3 **的值为** 3**，**D2 **类中的** pri4 **的值为** 6**，而虚基类** P **中的两个数据成员**

pri1 值为 1，pri2 值为 4。接着，对象 d 调用 D12 类中 display( )函数，输出结果如下：

```
D2::display( )
pri1=1,pri2=4
pri4=6
pri12=7
```

再接着，对象 d 调用 P 类中的 inc1( )函数，使得 pri1 增值为 2；对象 d 调用 D2 类中的 inc4( )函数，使得 pri4 增值为 7；对象 d 调用 D12 类中 inc5( )函数，使得 pri12 增值为 8。对象 d 再一次调用 D12 类中的 inc1( )，因为 D12 类本身无该函数，只能调用它从间接基类 P 中继承下来的 inc1( )函数，这时，该函数调用无二义性。调用 P 类中的 inc1( )函数后，使得 pri1 增值为 3。

练习题

如果在(4)**题的程序的主函数中，增加下述语句，将会出现什么问题？请上机调试。**

```
d.D12::inc1( )
```

主函数中最后使用对象 d **调用** D12 **类中的** display( )**函数，输出显示结果如下：**

```
D2::display( )
pri1=3,pri2=4
pri4=7
pri12=8
```

3.11 第 11 章习题答案和解答

3.11.1 习题答案

习题 11.1 (**略**)

习题 11.2 **选择填空**

C D A B A C C A

习题 11.3 **判断题**

描述正确的有(1)(3)(4)(5)(6)(9)(10)

习题 11.4 **分析程序结果**

(1) 15，16，17，18，19，

(2) B ∷ Show( ) called.155
D::Show( ).205

(3) B::virfun( ) Called.
D::virfun( ) Called.

(4) The A version A
The D1 info:4 version 1
The D2 info:100 version A
The D3 info:−25 version 3

(5) 72.3802
21.2

3.11.2 习题解答

第 11.4 题中(1)题

该程序中定义一个类 Matrix。**该类中定义一个运算符**( )**的重载函数。该重载函数返回值为** double &。**该类中定义了三个数据成员：**elem **是指向** double **型变量的指针，**row **和** col **是两个** int **型变量。使用** row **和** col **值分别表示某个矩阵的行数和列数。例如，本程序主函数中，创建的** Matrix **类的对象** m**，它将是一个具有** 5 **行** 8 **列的矩阵。重载运算符**( )**用来将给定的行数和列数转换为数组** elem **的下标值。例如，**m(1,2)**被转换为** elem［10］,**即为数组** elem **的第** 10 **个元素。而重载运算符**( )**的返回值为** double **型变量的引用，可以做赋值表达式的左值，即将一个表达式的值赋给该引用。例如，“** m(1,2)=1+15 **”；是将表达式** 1+15 **的值** 16 **赋给** elem[10]**，而输出** m(1,2)**的值应该是** 16。

在该程序的主函数中，先创建一个 Matrix **类的对象** m。**通过** for **循环，使用重载运算符**( )**给** elem **指针所指向的** double **型数组的某些元素赋值。该** for **循环共执行** 5 **次循环体，分别给** elem[2]，elem[10],elem[18],elem[26],elem[34]**赋值为** 15，16，17，18 **和** 19。**这里使用指针名用下标方式表示数组元素。该** for **循环实际上仅给** 5×8 **矩阵中第** 2 **列的** 5 **个行对应的元素赋了值。读者可以写出给这个矩阵的所有行列元素赋值。另外，还可以在** Matrix **类体中，增加一个成员函数，用来输出矩阵中任一元素的值。**

该程序的特点是使用成员函数的形式具体实现重载运算符的方法。

第 11.4 **题中**(2)**题**

该程序中有两个类：B **类和** D **类。它们之间的关系是** D **类以私有继承方式继承了** B **类。**

在主函数中，创建一个 D **类的对象** d1，**系统将自动调用** D **类中的构造函数来实现。在调用该构造函数时，应先调用其基类** B **的构造函数，使得** B **类数据成员** b **获取值为** 155，**并显示如下信息：**

B::Show( ) called.155

接着，执行 D **类构造函数的函数体，使得** D **类的数据成员** d **获取值为** 205，**接着再调用该类中的** Show( )**函数，输出显示如下信息：**

D::Show( ).205

第 11.4 **题中**(3)**题**

该程序中定义了类 B **和类** D，**并且类** D **是类** B **的派生类，以公有继承方式继承。在** B **类中，定义了两个构造函数，又定义了一个虚函数** virfun( )。**在** D **类中，除了定义两个构造函数外，又定义了一个与** B **类中虚函数** virfun( )**具有相同名、类型和参数的函数，该函数前面虽没加虚函数的关键字** virtual，**但它是一个缺省虚函数关键字的虚函数。**

从上述分析可知，D **类是** B **类的子类型，**D **类和** B **类中又存在有虚函数，存在着动态联编的条件。**

在主函数中，先使用运算符 new 对定义的指向 B 类对象的指针 pb 进行初始化，自动调用 B 类中的缺省构造函数。接着，调用 fun( )函数，该函数是程序中一般函数，它有一个指向 B 类对象指针的形参。调用函数的实参 pb 也是指向 B 类对象的指针，类型一致，将实参地址值赋给形参指针，在函数体中，用该形参指针调用虚函数 virfun( )，输出如下结果：

B::virufun( ) called.

这时，调用的是 B 类中的虚函数 virfun( )。

然后，主函数中，又定义一个指向 D 类对象的指针 pd，它也被运算符 new 初始化。又调用 fun( )函数，实参是指向 D 类对象的指针 pd，形参是指向 B 类对象的指针。看起来二者类型不一致，但是，由于 D 类是 B 类的子类型，因此，两者类型适应，调用成功。执行该函数的函数体，执行下列语句时，

obj->virfun( );

时，由于动态联编，自然调用 D 类中的 virfun( )函数，输出如下结果：

D::virfun( ) called.

第 11. 4 题中(4)题

该程序中定义了 4 个类：A 类、D1 类、D2 类和 D3 类。它们之间关系是 D1 类继承了 A 类，D3 类又继承了 D1 类，D2 类继承了 A 类，这些继承方式都是采用公有继承。在这 4 个类中都定义有一个虚函数 print( )，每个类中都有各个的构造函数。程序中又定义一个一般函数 print_info( ),该函数有一个形参指向 A 类的指针。

在 main( )中，开始创建了 4 个对象：对象 a 是 A 类的对象，创建时自动调用 A 类中的缺省构造函数；对象 d1 是 D1 类的对象，创建时自动调用 D1 类中的一个参数的构造函数；对象 d2 是 D2 类的对象，创建时自动调用 D2 类中的带有一个参数的构造函数；对象 d3 是 D3 类的对象，创建时自动调用 D3 类中带有一个参数的构造函数。创建后，各个对象中数据成员获值如下：

a 对象只有 ver 成员，获值为 'A'；

d1 对象 info 成员值为 4，ver 成员值为 '1'；

d2 对象 info 成员值为 100，ver 成员值为 'A'；

d3 对象 info 成员值为-25，ver 成员值为 '3'。

主函数中又 4 次调用 print_info( )函数，每次调用的实参不同。print_info( )函数的形参指向 A 类对象的指针，要求实参是同类对象的地址值。第一次调用 print_info( )函数时，实参是 &a，与形参类型相同，输出如下结果：

The A version A

第二次调用 print_info( )函数时，实参是 D1 类对象 d1 的地址值。由于 D1 类是 A 类的子类型，因此两者是类型适应，调用 D1 类中的 print( )函数，输出如下结果：

The D1 info:4 version 1

类似地，另外两次调用 print_info( )函数，输出结果分别为

The D2 info:1000 version A

The D3 info:-25 version 3

第 10.4 **题中**(5)**题**

该程序中定义了三个类：Shape **类、**Circle **类和** Rectangle **类。每个类中都定义了一个虚函数** Area( )**。**Circle **类是** Shape **类的派生类，并且是公有继承；同样** Rectangle **类是** Shape **类的派生类，也是公有继承。程序中又定义一个一般函数** fun( ),**该函数有一个参数，该参数是** Shape **类的引用。**

主函数中，先创建一个 Circle **类的对象** c**，自动调用** Circle **类的构造函数来实现。接着，调用** fun( )**函数，实参用** Circle **类的对象名** c**。由于实参与形参类型适应，在该函数体中，再调用** Circle **类中的** Area( )**函数，求得结果为** 72.3802**。然后，主函数内又创建一个** Rectangle **类的对象** r**，自动调用** Rectangle **类的构造函数来实现。再调用** fun( )**函数，使用的实参是** Rectangle **类的对象，由于类型适应，在** fun( )**函数体内，调用** Rectangle **类中的** Area( )**函数，获值** 21.2**。**

3.12 第 12 章习题答案和解答

3.12.1 习题答案

习题 12.1 **(略)**

习题 12.2 **选择填空**

C D A

习题 12.3 **判断题**

描述正确的有：(1)(2)(4)(5)

习题 12.4 **分析程序结果**

(1) 12
9.8
k
abgh

(2) 8.5 7.5 6.5 5.5 4.5 3.5 2.5 1.5 0.5 -0.5

(3) 10
11
12
13 14 15 16 17 18 19 20 21 22
1.1
2.2
3.3 4.4 5.5 6.6 7.7 8.8 9.9 10 11.1 12.2

习题 12.5 **编程(略)**

3.12.2 习题解答

第 12.4 题中(1)题

该程序中定义一个函数模板，它仅有一个模板参数 T。该函数的功能是用来求出两个数中最大者。程序中又对求两个字符串的最大者进行了定义。

在 main( )函数中，使用了 4 种模板函数，它们分别是

```
int max(int x,int y)
double max(double m,double n)
char max(char a,char b)
char *max(char *s1,char *s2)
```

这 4 个模板函数分别是使用 int,double,char 和 char *来替代模板参数 T 的结果。

第 12.4 题中(2)题

该程序中定义了一个类模板 Array，该类模板有 2 个参数，一个类模板参数 M 可被某种特定类型替换，而另一个是 int 型变量 N。在类 Array 中对下标运算符［ ］进行了重载。

在 main( )中，定义了一个模板类的对象 a。在该类中，使用 double 替代 M，并规定 N 为 10。该程序中，该模板类的参数中有一个为常量，这说明了类模板可以对常量进行参数化。

该程序中，类模板的两个参数，一个用来指出数组的类型，另一个用来给出数组的大小。后者参数是 int 型，对不同数组的大小只需给出一个常整型值就可以了。

第 12.4 题中(3)题

分析该程序的功能如下：

该程序中定义一个带有一个类模板参数的类模板 queue。

queue 类中有一个构造函数和一个析构函数，还有一个向队列中存入数据的函数 qstore( )和一个从队列中读取数据的函数 qretrieve( )。该类中还有 4 个数据成员：一个是指向所存放数据的队列的首地址的指针 q；一个是数据队列的长度 length 还有两个 int 型变量 sloc 和 rloc，用来标识存入数据和取出数据的位置。开始时，它们都被置为 0。

在主函数中，先后创建了 queue 类的两个对象 a 和 b，它们大小都设置为 10，前一个为 int 型类型，后一个为 double 型类型。当然，还可以使用类模板 queue 来创建其他类型的环形队列。下面分析环形队列 a 的操作，环形队列 b 与此类似。

通过 for 循环调用成员函数 qstore( )向环形队列 a 中写入 10 个 int 型数值，分别为 10，11，12，…，19。调用成员函数 qretrieve( )从队列中读取一个数值，并显示在屏幕上，其值为 10。接着，又调用 qstore( )函数，将数值 20 写入队列中，又调用 qretrieve( )函数从队列中读取一个数值 11 显示在屏幕上；又调用 qstore( )函数将数值 21 写入队列，再调用 qretrieve( )函数从队列中读取数值 12 显示在屏幕上。然后，又向队列中写入一个数值 22 的数据。最后，通过 for 循环将队列中存放的 10 个数据依次取出并显示在屏幕上，其结果为

```
13  14  15  16  17  18  19  20  21  22
```

关于对环形队列 b 中所存取的 double 型数据的操作与上面类似，不再详述。

通过上述分析，不难看出本程序将通过类模板 queue 可创建存放各种不同类型数据的环形队列，并且对该队列实现存取数据的操作。

3.13 第 13 章习题答案和解答

3.13.1 习题答案

习题 13.1 (略)

习题 13.2 选择填空

B C D B A C A D C D

习题 13.3 判断题

描述正确的有(2)(4)(5)(6)(8)(9)(10)

习题 13.4 分析程序结果

(1) 123456789
aabbccddeeff
mmmnnnpppqqq
ok!

(2) 123456789abcdefghijkl

(3) Hi,good morning!

(4) decVal:1000
otcVal:512
hexVal:4096

3.13.2 习题解答

第 13.2 题中(3)题

该题中，pa 是一个指向 int 型变量的指针，一般使用下述方法输出时，

```
cout<<pa<<endl;
```

输出指针 pa 的十六进制地址值。

如果需要输出某个指针的十进制地址值时，应该使用下述方法：

```
cout<<(long)pa<<endl;
```

即将指针强制为长整型量。

第 13.4 题中(1)题

该程序的 main( )函数中，先定义了两个流对象 infile 和 outfile。

再使用对象 outfile 调用 open( )函数以写入方式打开一个文件，该文件名为 text.dat。通过 if 语句判断打开文件操作是否成功。如果打开文件成功，继续执行程序中后续语句，否则退出该程序。这里，使用“abort( )”；语句实现退出程序。

接着，将 4 个字符串写入到被打开的文件中。这 4 个字符串分别是“123456789”，“aabbccddeeff”，“mmmnnnpppqqq”和“ok!”，再将该文件关闭。

然后，再使用对象 infile 调用 open( )函数以读方式打开 text.dat 文件，并判断打开操作是否成功。如果不成功，则退出程序。如果文件被成功打开，则使用 while 循环，从被打开的文件中使用 getline( )函数，读取一行字符存放在字符数组 textline［80］中，并将它显示到屏幕上，直到该文件中所有行被读出为止。while 循环的条件表达式如下所示：

!infile.eof( )

其中，eof( )是文件结束函数，当文件结束时，eof( )函数返回非 0，否则返回为 0。

第 13.4 题中(2)题

该程序主函数中，先定义一个 fstream 类的对象 file1。用对象 file1 调用 open( )函数以读写方式打开文件 text1.dat，并判断打开操作是否成功。如果不成功，则退出该程序，否则继续执行下面的语句。

接着，定义一个字符数组 textline［］，并给它赋了初值，该数组中存放了一个字符串。通过 for 循环，将存放在 textline 数组中的字符串从头到尾逐个写入被打开的文件中，写入操作使用了 put( )函数。写入完成后，使用 file1 调用 seekg( )函数，将文件的读指针移动到该文件头。

再接着，通过 while 循环，使用 file1 调用 get( )函数每次从文件中读出一个字符存放在 ch 变量中，然后将它显示在屏幕上，直到该文件中全部内容被读完，结束 while 循环，将打开的文件关闭后退出程序。

这是一个典型的文件写读操作的程序。

第 13.4 题中(3)题

该程序是一个有关字符串流操作的程序。主函数中，先定义了 ostrstream 类的一个对象 s，该对象用来存储插入数据的数组对象。接着将一个字符串“Hi,good morning!”插入到对象 s 中。再使用对象 s 调用 str( )函数，返回标识对象 s 的地址值，并赋给字符指针 buf，也就是使字符指针 buf 指向对象 s 中存放的字符串。然后，输出字符指针 buf 所指向的字符串。最后，释放已被定义的 buf 指针。

第 4 章　面向对象方法的小结

C++语言是一种面向对象的程序设计语言，面向对象的程序设计方法是一种新方法，它是程序设计发展史中的一个重大的进步，掌握这种新型的程序设计方法对进行软件开发是很重要的。本书就是力图使读者能够掌握 C++这种程序设计语言，应用面向对象的程序设计方法开发软件。

面向对象方法是从现实世界中存在的客观事物(即对象)出发来构造软件系统的。在分析和解决客观实际问题时，尽可能运用人类的自然思维方式，即从研究的客观事物出发，将它们抽象为系统中的对象，作为构成系统的基本单位，再把具有相同属性和服务的对象抽象为类。类是对若干对象的抽象描述，对象是它所属的类的实例。对象和类是封装性的体现，封装就是把对象的属性和服务构成一个独立单位，并尽可能的隐蔽对象的内部细节。封装是面向对象的一个原则，也是为面向对象技术提供的一种机制。因此，可以说封装是面向对象方法的基础。

从客观事物出发，经过抽象研究可以得到较为一般的类和较为特殊的类。特殊类继承一般类的属性和服务，它自身有新的属性和服务。继承是面向对象方法的关键。继承有单继承和多继承，并且继承有传递性。继承具有重大的现实意义，它大大减轻软件开发工作的强度，它不仅可以实现软件的复用，又可扩展复用的范围。

多态性为开发者带来极大的方便。多态性包含了重载机制，函数和运算符可以重载；它还包含了动态联编和静态联编的机制，这些就使得具有相同名字的函数对应着不同的实现代码；函数代码的连接可以在编译时进行，也可以在运行时进行，这些机制给编程者带来很大的方便。

4.1　面向对象方法的概括

回顾前面讲述的 C++语言的特点和内容，将它们概括，使读者从中更能理解面向对象方法的特点和精华，对读者今后深入学习面向对象方法打下坚实的基础。

类的确定是基础

类和对象是封装机制中两个重要概念，又是 C++语言编程中两种不同的语法成分。类是对象的模板，对象是类的实例。

一、确定类应从发现对象入手

发现对象的出发点是对实际问题和系统功能的研究。

研究实际问题又称研究问题域。问题域是指开发的应用系统所要考虑的整个业务范围。研究问题域需要亲临现场了解第一手材料，并且要熟悉要解决问题所需的业务范围。例如，开发财务软件，就要深入到有关会计工作的第一线，熟悉有关会计工作的业务。研究问题域还要认真听取专家意见，大量阅读有关材料，并做好记录，还要借鉴一些已有的成果，这样可以提高效率。

研究用户需求。用户需求体现在系统功能上，了解用户对要开发的各种要求和期望。为此，应做到一听、二看、三交流、四整理。这就是耐心听用户的要求，认真看有关的书面材料，虚心与用户进行交流，搞清所有疑点，最后将所有的记录整理成文。再从实际问题出发，发现对象，最后落实到系统功能的实现上。所以，问题研究和系统功能是发现对象和确定对象的两个方面，二者是一致的，其目的都是为了解决要开发的问题。

发现对象是为了确定对象。在发现对象时可以找到尽可能有用的侯选对象，宁多勿少，防止遗漏。发现对象后，接着要再进行审查和筛选。这里要正确运用原则。

抽象原则是确定对象和类的基本原则。抽象就意味着取舍，取舍的准则是看所观察的事物及特征是否与当前的目标一致。在使用抽象原则进行取舍时，先是取舍对象，将一些与系统功能无关的对象舍弃掉；进一步再根据系统功能的要求取舍无关的属性和服务。经过合理的取舍后，最后确定了必要的对象和类。

在确定对象和类时，要充分利用封装机制增加类的自身可靠性，即严格限制对类中数据成员的访问，增加数据的安全性。正确使用对类成员的访问权限的规定，尽量少使用不安全的全局量。

二、认识类中的缺省成员

在定义或说明一个类时，编译系统将会自动增加一些成员函数，它们称为缺省成员，供缺省情况下使用。用户应该搞清楚编译系统会自动创建哪些函数？它们功能是什么？它们的格式如何？

① 当用户没有定义任何构造函数时，系统将自动创建缺省构造函数，用来初始化所创建的对象。缺省构造函数(系统提供的)格式如下：

〈类名〉(　)

{　}

该函数名同类名，无参数，函数体为空。

② 当用户没有定义析构函数时，系统将自动创建缺省析构函数，用来删除终止生存期的对象。该函数格式如下：

~〈类名〉(　)

{　}

该函数名同类名，无参数，函数体为空。

③ 当用户没有定义拷贝初始化构造函数时，系统将自动创建一个缺省拷贝初始化构造函数，用来实现用一个已知对象创建一个新对象。该函数原型如下：

〈类名〉(const〈类名〉&〈对象名〉)；

该函数名同类名，该函数参数为该类的对象引用，并为常类型的对象引用。

④ 当用户没有定义重载赋值运算符时，系统将提供一个缺省赋值运算符函数，用来实现动态赋值，即将一个已知对象的值改变另一个对象值。该函数原型如下：

〈类名〉& operator =(const〈类名〉& 〈对象名〉)；

该函数名可认为是“operator =，”其中 operator 是关键字，该函数有一个参数是该类的常对象引用，该函数类型为该类的对象引用。

⑤ 系统还提供两个缺省的取地址运算符函数，用来实现对该对象的取地址运算。其函数格式分别为：

〈类名〉* operator &( )
{ return this; }

和

const〈类名〉* operator &( )const
{return this; }

这两个函数都是用来返回对象的地址值，但是前一个是一般成员函数，后一个是常成员函数。

当用户定义一个空类时，实际上该类并不空，因为系统将为该类提供了上述的 5 种缺省函数。请读者注意，一旦用户在类中定义了相应的成员函数时，则相对应的缺省成员函数将不再提供。

三、一些建议

在程序中设计一个类，实际上是设计一种类型，在设计一个类时，除了考虑到它的若干数据成员外，还要考虑到成员函数，即该类中的服务或行为。下面就是围绕着创建一个好的类提供一些建议。

① 程序中的类总是为了创建对象用的。首先应该考虑如何创建对象和删除对象？

创建对象就要自动调用构造函数，为对象进行初始化。构造函数是一种特殊的成员函数，它可以重载，于是便可出现不同的构造函数，通常有如下几种：

- 不带参数的缺省构造函数，可以用户定义，也可以系统自动提供；
- 带不同参数个数的构造函数，一般取决于该类中数据成员的个数；
- 拷贝初始化构造函数，用来实现用一个已知对象创建一个与其相同的对象。

创建动态对象(即堆对象)时，使用运算符 new，系统也将调用相应的构造函数，为所创建的对象进行初始化。

删除对象时要自动调用析构函数来释放对象。析构函数也是一种特殊的成员函数，由于它没有参数，因此无法重载。在使用运算符 delete 删除动态对象(堆对象)时，系统自动调用析构函数来释放被删除的对象。

② 如何实现对象的赋值？对象初始化与对象赋值是不同的。对象赋值是指动态地改变对象的已有值。通常使用缺省的赋值运算符或重载赋值运算符来实现。

③ 如何确定类中成员的访问权限？这是定义类时应首先考虑的问题。为了实现类的封装，尽量将类中的数据成员说明为私有成员，也可以将个别的成员函数说为私有成员。将类中的多数成员函数作为类的接口被说明为公有成员，程序将通过这些接口来实现对其私有数据成员的访问。考虑到类继承的需要，可将一些成员，特别是数据成员，说明为保护成员。它对派生类来讲，相当于公有成员，而对其类的对象来讲，相当于私有成员。为了简化对类中私有成员访问的复杂性，又引进了友元函数和友元类的概念，这些都体现了 C++语言的灵活性。

④ 类中的类型转换如何实现？类中允许编写类型转换函数来实现不同类型的转换，另外具有一个参数的构造函数也可用来实现类型转换，还可以使用强制类型运算符进行强制转换。C++语言中，类型比 C 语言要严格些，一般地都要使用显式转换，但是也存在少量的隐含转换。

⑤ 类中静态成员的使用会带来什么作用？全局变量在程序中的使用会带来不安全感，

因此，尽量少用。类中的静态成员是一种局部于类中的“全局变量”，它被该类中所有对象所共享，而不是属于某个对象的。它被存储于内存一个公用区域中，适当使用静态成员会给编程带来较大方便。

⑥ 模板的使用会带来哪些方便？有时在提供的一系列类中都具有相似的功能，在这些类中除了成员的数据类型不同而外，其余都相同。于是便引进模板的概念，将其类型进行参数化，这样不仅会带来书写简练的好处，而且还增加程序的可读性。

使用继承是关键

继承性是面向对象方法中最重要的机制，只有封装而无继承的程序设计方法不是面向对象的程序设计方法。继承是面向对象方法的关键。正确使用继承机制编写面向对象的程序是十分重要的技术。

一、继承提供了类的层次结构的实现

在实际问题中，常常存在着具有许多共享特性的若干个类，我们可以利用继承的机制，让这些类派生于同一个基类，这些派生类共享基类的特性。这种机制在软件开发中无疑是减少冗余性，而增强共享性。

客观世界的本身是有层次的，而继承机制就为人们认识客观世界提供一种层次分类的方法，较好地处理了一般类与特殊类的关系。使得程序员可以在一个一般类的基础上很快地创建新的类，而不必从头开始设计每一个类，这便会大大提高软件开发的效率。

二、类的继承和包含的区别

两个类之间有两种不同的关系，一种是通常的继承关系，即由一个基类派生出一个新类；另一种关系是包含关系，即一个类的对象作为另一个类的成员，称为对象成员。这两种不同的描述形式在实际中需要合理使用。例如，有一个类 student 用来描述一个学生，另一个类 address 描述一个人的地址。这两个类可以有如下两种不同的关系：

如果采用继承，可定义如下：

```
class student: public address
{
    //  …
};
```

如果采用包含，即在 student 类中定义一个 address 类的对象成员，可定义如下：

```
class student
{
  private:
    address   studentAddr;
          …
};
```

前一种方式使用继承的方法，说明学生是一个地址；后一种方式使用包含的方法，表明学生包含一个地址属性。这两种形式哪种合理呢？显然，后一种方式是比较合理的。

因此，类的关系中采用哪种方式要具体问题具体分析。又例如，要创建一个研究生类

postgraduate；它与学生类的关系应该是继承还是包含呢？选择继承为更合理，因为研究生就是学生，研究生所具有的特殊属性都可定义在 postgraduate 中，即：

```
class postgraduate: public student
{
    // …
};
```

一般地说，不同类对象之间的联系采用继承的方法来表示，而对象所具有的属性应采用成员来表示。在实际编程中应注意两者的关系。

三、公有继承中派生类对象就是基类对象

如果类 B 公有继承类 A，则类 B 的每一个对象就是类 A 的对象。但是，反之则不然。也可以说在使用类 A 对象的任何地方，也可以使用类 B 的对象。简单地说，类 B 的每个对象“就是一个”类 A 的对象，反之则不然。这一规则又称为类型适应，或称赋值兼容规则。

公有继承中的上述规则在实际应用中是很重要的。在后面讲到的多态性中的动态联编的实现，除了虚函数外，还应满足公有继承的上述规则，否则无法实现动态对象的改变。

四、在类继承的问题中尽量多用单继承

C++语言尽管支持单继承和多继承，但是在实际编程中还是尽量多用单继承，少用多继承。道理很简单，因为多继承毕竟比单继承复杂。

在处理实际问题中，当遇到了两个具有一些共同内容的类时，并不一定让其中的一个类继承另一个类，而可以让两个类同时继承于一个共同的基类。这种处理方法有时会避免出现多继承的问题。

多态性的使用带来灵活和方便

多态性是面向对象方法中的一个重要特性。简单地说，多态性指同一个名字将对应着多种不同的行为或实现。例如，同一个函数名将对应着不同的函数体，这种多态性称为函数重载。同样，在 C++语言中还允许运算符重载，即对已有的运算符可以通过函数定义的形式来定义新的功能。

多态性又称重载性。被重载的函数和运算符是通过调用时的函数参数的个数和对应的类型进行选择。因此，在定义重载函数时应注意在参数个数或类型方面应有所区别，否则将会无法选择。

C++语言允许在类的层次结构中建立相同的成员函数。这些函数在概念上十分相似，但是它们在不同的类中对应着不同的实现。于是，在用同一个函数名调用对象时，将会引起不同的行为，这种功能也称为多态性。而这种多态性的实现与联编有关。所谓“联编”是指将一个函数调用链接上相应的函数体代码。C++语言支持两种联编：静态联编和动态联编。

静态联编是在程序编译时进行的，它的最大优点是速度快。因为在编译时根据源代码调用的标识符选定了函数体，在运行时仅作传递参数、执行函数调用等开销。但是，这种联编方法要求程度员必须预测，在每种情况下所有函数调用将要使用的对象。这样不仅具有局限性，而且有时也是很难办到的。

动态联编是在程序运行时进行的，它虽然执行效率较低，但是灵活性很好。因为动态联

编是在运行时按源代码所指明的对象进行函数体的链接。支持动态联编需要虚函数。虚函数类似于重载函数，但它与一般重载函数不同。虚函数通常是被定义在派生类中的重定义的基类函数，并且在基类中该函数要使用关键字 virtual 进行说明。在派生类中定义的函数与基类中的同名虚函数，如果在参数个数和相应类型以及返回类型等完全一样，则该函数无论是否用 virtual 说明都是一个虚函数。

虚函数是进行动态联编的一个重要前提，在无虚函数的情况下，是不会进行动态联编的。但是，有了虚函数的情况下是否一定进行动态联编呢？回答是不肯定的。在通过对象的指针或引用调用虚函数时，C++系统对该调用进行动态联编。如果通过对象调用虚函数时，则 C++系统对该调用进行静态联编。另外，在构造函数和析构函数中调用函数时，应采用静态联编。它们所调用的虚函数是自身类中或基类中定义的函数，而不是任何在派生类中重新定义的虚函数。

对虚函数的进一步讨论出现了纯函数的概念和抽象类的概念。在有些情况下，基类中定义的虚函数不具有具体意义或不给它一个有意义的定义，则这样的虚函数称为纯虚函数。表示如下：

virtual 〈类型〉〈函数名〉(〈参数表〉)=0；

具有纯虚函数的类称为抽象类。

当一个基类中定义了一个纯虚函数时，该纯虚函数的具体定义留给其派生类。基类中给出的纯虚函数实际上仅起到为派生类提供一个统一的接口作用。

抽象类只能作为基类来派生新类，而不能用来创建对象，但可以用来说明指向对象的指针或引用。成员函数可以调用纯虚函数，而构造函数或析构函数是不能调用纯虚函数的。

使用消息进行对象通信

消息是面向对象方法中又一个较为重要的概念。虽然在 C++语言中没有使用消息这个名字，但是实际上已经使用了消息这个概念，并且可以说是频繁使用了这一概念。在 C++语言中，所谓消息其实就是函数的调用。读者可以回忆一下，在 C++语言的源程序中，怎么能够离得开函数的调用呢？由于历史上的原因，在源程序中人们已习惯于使用函数调用的概念。如果换成使用消息这一概念会使人们感到陌生。可是消息这个概念，在计算机技术和信息技术高速发展的网络时代，已广泛采用面向对象的方法进行软件开发，显得更具有普遍性，更接近于人们日常的思维方式，更具有时代感。所以，消息这个术语在今天更有其现实性，搞清这一概念对进一步理解面向对象方法是十分有益的。

关于信息的定义可以描述如下：

消息就是向对象发出的服务请求，它包含有提供服务的对象标识、服务标识以及输入信息和回答信息。

消息的接收者是提供服务的对象，它对外提供的每一种服务是按消息格式规定好的消息协议，这种消息协议在 C++程序中便是该种服务的具体功能，即通过函数体来实现的。

下面通过一个简单的例子作一说明。

假定有一个类 A，它提供了一种输出显示该类中数据成员值的服务。定义一个类 A 的对象 obj。接着，向对象 obj 发出一个服务请求，该请求就是将该对象的两个数据成员输出显示。写出这条完整的消息，应包含如下内容：

·消息的接收者(对象标识)

・服务标识(函数名)

・符合消息协议要求的参数(函数实参表)

程序内容如下：

```
#include<iostream.h>
class A
{
    public:
        A(int i,int j)
        { a=i; b=j; }
        void print( )
        {
                cout<<"a="<<a<<",b="<<b<<endl;
        }
    private:
        int a,b;
};

void main( )
{
        A   obj(5,8);
        obj.print( );
}
```

这是一个简单的程序，主要用来说明消息在 C++程序的使用。在该程的 main( )函数中，“obj.print( );”语句是一条消息，而实际上是用对象 obj 调用类 A 内的一个公有成员函数 print( )，用来输出对象 obj 的两个数据成员 a 和 b 的值，这条消息是向对象 obj 发出一种服务请求，即要求进行输出数据成员值服务，而这种服务是通过类 A 中的一个成员函数来实现的，即执行该成员函数便可实现这一服务要求。分析这条消息可知：

・消息的接收者是类 A 对象 obj

・实现服务是调用类 A 中成员函数 print( )

・该服务的消息协议中无要求的参数

面向对象方法中，通过消息进行对象之间的通信，这与封装原则是有关系的。因为封装使得对象成为互不干扰的独立单位，只有通过消息才可为它们提供一种合法的动态联系，使它们的行为相互配合，构成一个有机的动态的系统。

综上所述，面向对象方法中存在一些新的概念和机制，C++语言中充分地体现了这些新概念的应用，读者应该结合学习 C++语言来理解和掌握面向对象方法的特点。在这些新概念和新机制中，主要的是本节前面提到的 4 个：

(1) 封装：类和对象

(2) 继承：基类和派生类

(3) 多态：重载和联编

(4) **消息：对象及其服务**

读者在学习中一定要掌握这些基本概念和基础知识，其将为你们今后进一步学习面向对象设计方法的理论打好基础。

4.2 实 例 说 明

这里举个实际例子，通过它进一步熟悉使用 C++**语言进行面向对象方法编程的基本方法。**

编程要求如下：

假定有三种文具用品：圆元珠笔、练习本和尺子。它们共有的属性是品名、单价和数量。另外，圆珠笔有颜色(用一个字符表示)，练习本有页数，尺子有刻度长度。

要求每种文具定义为一个类，当给定某种文具的确定属性后，利用其服务显示出各种文具的品名、数量和单价，并计算出所有这些文具的总金额。在计算某种文具总价钱时，如果 10 **个以上可以打折，小于** 10 **个按其原价，不同文具的打折数不同。**

下面使用两种方法编写该程序，最后比较这两种方法的区别。

方法一

程序内容如下:

```
#include<iostream.h>
class ballpen1
{
   public:
       ballpen1(char *,double,int,char);
       char *Getname( )
       { return name; }
       double Getunit_price( )
       { return unit_price; }
       int Getquantity( )
       { return quantity; }
       double Total( );
       char Getcolor( )
       { return color1; }
   private:
       char *name;
       double unit_price;
       int quantity;
       char color1;
};
ballpen1::ballpen1(char *name1,double price,int number,char col1)
{
```

```
        name=name1;
        unit_price=price;
        quantity=number;
        color1=col1;
}
double ballpen1::Total( )
{
        double total=0;
        if(quantity<10)
                total=unit_price*quantity;
        else
                total=0.85*unit_price*quantity;
        return total;
}
class ballpen2
{
   public:
        ballpen2(char *,double,int,char,char);
        char *Getname( )
        { return name; }
        double Getunit_price( )
        { return unit_price; }
        int Getquantity( )
        { return quantity; }
        double Total( );
        char Getcolor1( )
        { return color1; }
        char Getcolor( )
        { return color2; }
   private:
        char *name;
        double unit_price;
        int quantity;
        char color1,color2;
};
ballpen2::ballpen2(char *name1,double price,int number,char col1,char col2)
{
        name=name1;
        unit_price=price;
        quantity=number;
```

```
        color1=col1;
        color2=col2;
}
double ballpen2::Total( )
{
        double total=0;
        if(quantity<10)
                total=unit_price*quantity;
        else
                total=0.95*unit_price*quantity;
        return total;
}

class notebook
{
   public:
        notebook(char *,double,int,int);
        char *Getname( )
        { return name; }
        double Getunit_price( )
        { return unit_price; }
        int Getquantity( )
        { return quantity; }
        double Total( );
        int Getpagenum( )
        { return pagenum; }
   private:
        char *name;
        double unit_price;
        int quantity;
        int pagenum;
};
notebook::notebook(char *name1,double price,int number,int page)
{
        name=name1;
        unit_price=price;
        quantity=number;
        pagenum=page;
}
double notebook::Total( )
```

```
{
        double total=0;
        if(quantity<10)
                total=unit_price*quantity;
        else
                total=0.9*unit_price*quantity;
        return total;
}

class rule
{
    public:
        rule(char *,double,int,int);
        char *Getname( )
        { return name; }
        double Getunit_price( )
        { return unit_price; }
        int Getquantity( )
        { return quantity; }
        double Total( );
        int Getlength( )
        { return length; }
    private:
        char *name;
        double unit_price;
        int quantity;
        int length;
};
rule::rule(char *name1,double price,int number,int leng)
{
        name=name1;
        unit_price=price;
        quantity=number;
        length=leng;
}
double rule::Total( )
{
        double total=0;
        if(quantity<10)
                total=unit_price*quantity;
```

```
        else
            total=0.75*unit_price*quantity;
        return total;
}

void main( )
{
        ballpen1 pen1("ballpen1",5.5,8, 'r');
        ballpen2 pen2("ballpen2",8.8,12, 'r', 'b');
        notebook book("notebook",4.8,10,80);
        rule rule("rule",2.0,3,50);
        cout<<pen1.Getname( )<<"= = ="<<pen1.Getquantity( )
            <<'*'<<pen1.Getunit_price( )<<endl;
        cout<<pen2.Getname( )<<"= = ="<<pen2.Getquantity( )
            <<'*'<<pen2.Getunit_price( )<<endl;
        cout<<book.Getname( )<<"= = ="<<book.Getquantity( )
            <<'*'<<book.Getunit_price( )<<endl;
        cout<<rule.Getname( )<<"= = ="<<rule.Getunit_price( )
            <<'*'<<book.Getunit_price( )<<endl;
        cout<<endl<<"SUM="<<pen1.Total( )+pen2.Total( )+book.Total( )
            +rule.Total( )<<endl;
}
```

执行该程序后，输出如下结果：

```
ballpen1= = =8*5.5
ballpen2= = =12*8.8
notebook= = =10*4.8
rule= = =2*4.8
SUM=193.52
```

方法二

程序内容如下:

```
#include<iostream.h>
class stationery
{
  public:
    stationery(char *,double,int);
    char *Getname( )
    { return name; }
```

```
        double Getunit_price( )
        { return unit_price; }
        int Getquantity( )
        { return quantity; }
        virtual double Total( )=0;
    protected:
          double unit_price;
          int quantity;
    private:
        char *name;
};
stationery::stationery(char *name1,double price,int number)
{
        name=name1;
        unit_price=price;
        quantity=number;
}

class ballpen1:public stationery
{
    public:
       ballpen1(char *,double,int,char);
        char Getcolor1( )
        { return color1; }
        double Total( );
    private:
        char color1;
};
ballpen1::ballpen1(char *name1,double price,int number,char col):
              stationery(name1,price,number)
{
        color1=col;
}
double ballpen1::Total( )
{
        double total=0;
        if(quantity<10)
              total=unit_price*quantity;
        else
              total=0.85*unit_price*quantity;
```

```
        return total;
}

class ballpen2:public ballpen1
{
   public:
      ballpen2(char *,double,int,char,char);
        char Getcolor2( )
        { return color2; }
        double Total( );
   private:
        char color2;
};
ballpen2::ballpen2(char *name1,double price,int number,char col1,char col2):
                ballpen1(name1,price,number,col1)
{
        color2=col2;
}
double ballpen2::Total( )
{
        double total=0;
        if(quantity<10)
                total=unit_price*quantity;
        else
                total=0.95*unit_price*quantity;
        return total;
}
class notebook:public stationery
{
   public:
        notebook(char *,double,int,int);
        double Total( );
        int Getpagenum( )
        { return pagenum; }
   private:
        int pagenum;
};
notebook::notebook(char *name1,double price,int number,int page):
                stationery(name1,price,number)
{
```

```
        pagenum=page;
}
double notebook::Total( )
{
        double total=0;
        if(quantity<10)
                total=unit_price*quantity;
        else
                total=0.9*unit_price*quantity;
        return total;
}

class rule:public stationery
{
   public:
        rule(char *,double,int,int);
        double Total( );
        int Getlength( )
        { return length; }
   private:
        int length;
};
rule::rule(char *name1,double price,int number,int leng):
          stationery(name1,price,number)
{
        length=leng;
}
double rule::Total( )
{
        double total=0;
        if(quantity<10)
                total=unit_price*quantity;
        else
                total=0.75*unit_price*quantity;
        return total;
}

void main( )
{
        stationery *stati[4];
```

```
    double sum=0;
    stati[0]=new ballpen1("ballpen1",5.5,8, 'r');
    stati[1]=new ballpen2("ballpen2",8.8,12, 'r', 'b');
    stati[2]=new notebook("notebook",4.8,10,80);
    stati[3]=new rule("rule",2.0,3,50);
    for(int i=0;i<4;i++)
        cout<<stati[i]->Getname( )<<"= = ="<<stati[i]->Getquantity( )
            <<'*'<<stati[i]->Getunit_price( )<<endl;
    for(i=0;i<4;i++)
        sum+=stati[i]->Total( );
    cout<<endl<<"SUM="<<sum<<endl;
    delete [ ] *stati;
}
```

上述两个程序有着相同的结果。它们的区别主要表现在下述方面：

(1) **方法一中定义的是** 4 **个独立的类，它们之间是没有联系的。可见它们之间具有许多相同的内容。**

(2) **方法二中定义的** 4 **个类之间是类属关系。类** stationery **是一个抽象类，表示一个抽象的文具。而类** ballpen1、**类** notebook、**类** rule **都是类** stationery **的派生类，而** ballpen2 **是** ballpen1 **的派生类。这些类之间存在着继承关系。这些继承都是公有继承。**

(3) **方法二中为实现动态联编定义了虚函数** Total( )。**在基类** stationery **中用关键字** virtual **说明了** Total( )**函数，在它的派生类中简略了说明。这样对** Total( )**函数可以实现动态联编。对抽象类** stationery **来讲，虽然不可定义对象，但可定义指向对象的指针，该程序中定义了一个对象指针数组** stati[4]。

(4) **方法二中有些数据成员被说明为保护成员，其目的是为了方便在其派生类中的引用。如果说明为私有的，则在其派生类的成员函数中便无法引用。**