软件开发视频大讲堂

十年锤炼 打造经典

C语言

从入门到精通

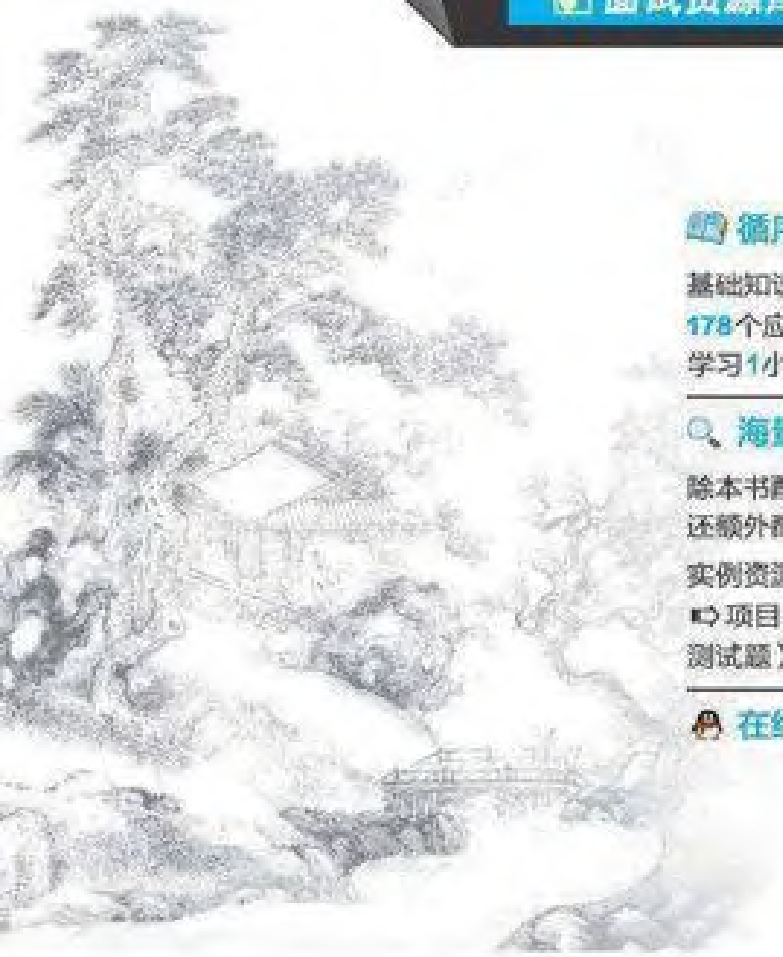
13小时微课视频讲解

明日科技 编著

☑实例资源库 ☑模块资源库 ☑项目资源库

☑面试资源库 ☑测试题库系统 ☑PPT电子课件

（第5版）



循序渐进，实战讲述

基础知识➪核心技术➪高级应用➪项目实战

178个应用示例，134个编程训练，128个综合训练，2个项目案例

学习1小时，训练10小时，打造全新学习生态

海量资源，可查可练

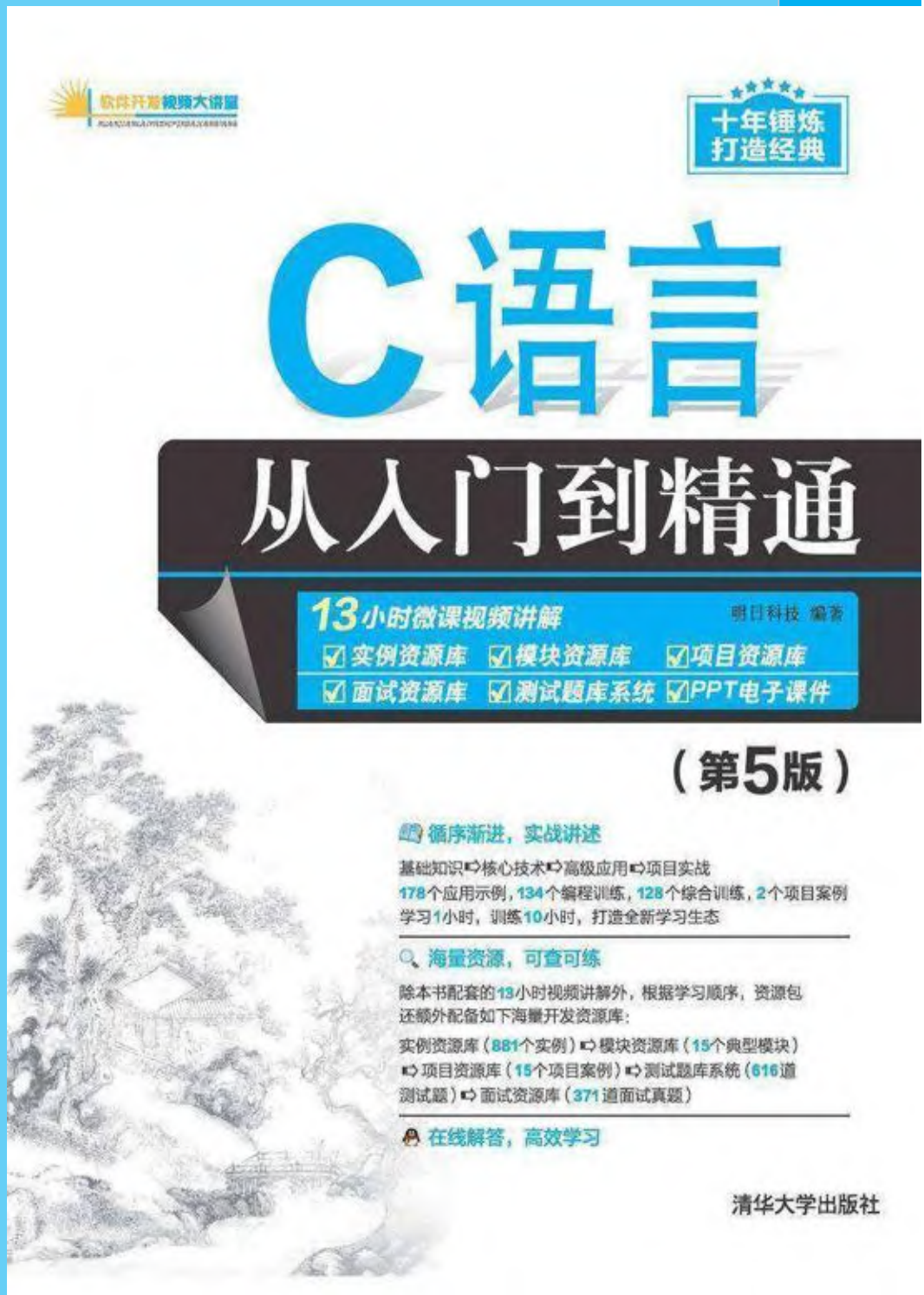
除本书配套的13小时视频讲解外，根据学习顺序，资源包还额外配备如下海量开发资源库：

实例资源库（881个实例）➪模块资源库（15个典型模块）➪项目资源库（15个项目案例）➪测试题库系统（616道测试题）➪面试资源库（371道面试真题）

在线解答，高效学习

清华大学出版社

随时、随地、随心阅读


软件开发视频大讲堂
十年锤炼
打造经典
C语言
从入门到精通
13小时微课视频讲解
明日科技 编著
实例资源库 模块资源库 项目资源库
面试资源库 测试题库系统 PPT电子课件
（第5版）
循序渐进，实战讲述
基础知识➪核心技术➪高级应用➪项目实战
178个应用示例，134个编程训练，128个综合训练，2个项目案例
学习1小时，训练10小时，打造全新学习生态
海量资源，可查可练
除本书配套的13小时视频讲解外，根据学习顺序，资源包
还额外配备如下海量开发资源库：
实例资源库（881个实例）➪模块资源库（15个典型模块）
➪项目资源库（15个项目案例）➪测试题库系统（616道
测试题）➪面试资源库（371道面试真题）
在线解答，高效学习
清华大学出版社

当当云阅读

软件开发视频大讲堂

C 语言从入门到精通

（第 5 版）

明日科技　编著

清华大学出版社

北　京

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版 权 所 有 ， 侵 权 必 究 。 举 报 ： 010-62782989 ， beiqinquan@tup.tsinghua.edu.cn。

图书在版编目（CIP）数据

C语言从入门到精通／明日科技编著．—5版．—北京：清华大学出版社，2021.9

（软件开发视频大讲堂）

ISBN 978-7-302-58567-1

Ⅰ．①C… Ⅱ．①明… Ⅲ．①C语言—程序设计 Ⅳ．①TP312.8

中国版本图书馆CIP数据核字（2021）第132329号

责任编辑：贾小红

封面设计：刘 超

版式设计：文森时代

责任校对：马军令

责任印制：沈 露

出版发行：清华大学出版社

网 址：http://www.tup.com.cn，http://www.wqbook.com

地　　址：北京清华大学学研大厦A座

邮　　编：100084

社 总 机：010-62770175

邮　　购：010-62786544

投 稿 与 读 者 服 务：010-62776969，c-service@tup.tsinghua.edu.cn

质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市金元印装有限公司

经　　销：全国新华书店

开　　本：203mm×260mm

印　　张：26.5

字　　数：724千字

版　　次：2010年7月第1版　2021年9月第5版

印　　次：2021年9月第1次印刷

定　　价：79.80元

产品编号：092539-01

内容简介

《C语言从入门到精通（第5版）》从初学者的角度出发，以通俗易懂的语言、丰富多彩的实例，详细介绍了使用C语言进行程序开发需要掌握的各方面知识。全书分为4篇，共18章，内容包括C语言概述、算法、数据类型、运算符与表达式、常用的数据输入／输出函数、选择结构程序设计、循环控制、数组、函数、指针、结构体和共用体、位运算、预处理、文件、存储管理、网络套接字编程、单词背记闯关游戏和防空大战游戏。书中所有知识都结合具体实例进行介绍，涉及的程序代码给出了详细的注释，读者可以轻松领会C语言程序开发的精髓，快速提高开发技能。

另外，本书除了纸质内容，附赠资源包中还给出了海量开发资源库，主要内容如下：

- ☑ **微课视频讲解：总时长13小时，共215集**
- ☑ **实例资源库：881个实例及源码详细分析**
- ☑ **模块资源库：15个经典模块开发过程完整展现**
- ☑ **项目资源库：15个企业项目开发过程完整展现**
- ☑ **测试题库系统：616道能力测试题**
- ☑ **面试资源库：371道企业面试真题**
- ☑ **PPT电子教案**

本书可作为软件开发入门者的自学用书，也可作为高等院校相关专业的教学参考书，还可供开发人员查阅、参考。

如何使用本书开发资源库

在学习《C语言从入门到精通（第5版）》时，随书附赠的资源包中提供了“Visual C++开发资源库”系统，可以帮助读者快速提升编程水平和解决实际问题的能力。《C语言从入门到精通（第5版）》和“Visual C++开发资源库”配合学习的流程如图1所示。

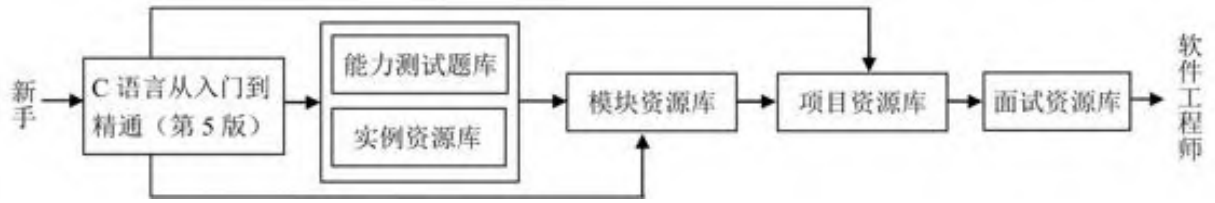


图1　本书与附赠开发资源库配合学习的流程

打开资源包中的“开发资源库”文件夹，运行Visual C++开发资源库.exe程序，即可进入“Visual C++开发资源库”系统，主界面如图2所示。



图2　Visual C++开发资源库主界面

在学习本书某一章节时，可利用实例资源库对应章节提供的大量热点实例和关键实例巩固所学编程技能，提高编程兴趣和自信心；也可以利用能力测试题库的对应章节进行测试，检验学习成果。具体流程如图3所示。

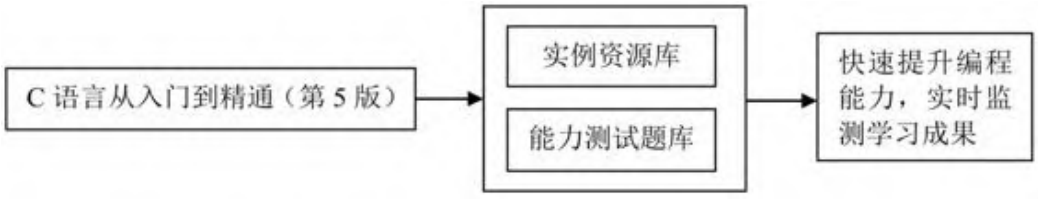


图3　使用实例资源库和能力测试题库

对于数学逻辑能力和英语基础较为薄弱的读者，本书提供了数学及逻辑思维能力测试和编程英语能力测试，以供读者进行练习和测

试，如图4所示。

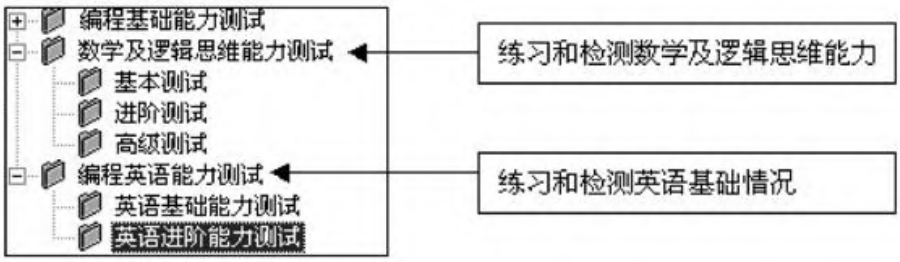


图4　数学及逻辑思维能力测试和编程英语能力测试目录

学习完本书后，读者可通过模块资源库和项目资源库中的30个经典模块和项目，全面提升个人综合编程技能和解决实际开发问题的能力，为成为C语言软件开发工程师打下坚实基础。具体模块和项目目录如图5所示。

万事俱备，该到软件开发的主战场上接受洗礼了。面试资源库中提供了大量国内外软件企业的常见面试真题，同时还提供了程序员职业规划、程序员面试技巧、企业面试真题汇编和虚拟面试系统等精彩内容，是程序员求职面试的绝佳指南。面试资源库的具体内容如图6所示。



图5　模块资源库和项目资源库目录

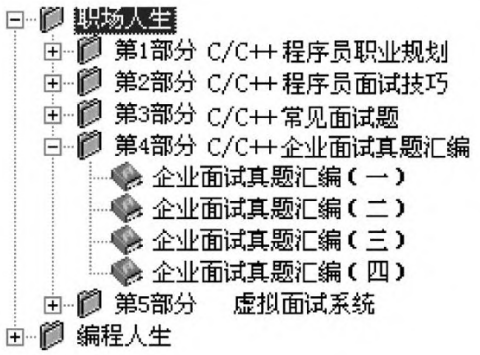


图6 面试资源库具体内容

前　言
Preface

丛书说明：“软件开发视频大讲堂”丛书（第1版）于2008年8月出版，因其编写细腻、易学实用、配备海量学习资源和全程视频等，在软件开发类图书市场上产生了很大反响，绝大部分品种在全国软件开发零售图书排行榜中名列前茅，2009年多个品种被评为“全国优秀畅销书”。

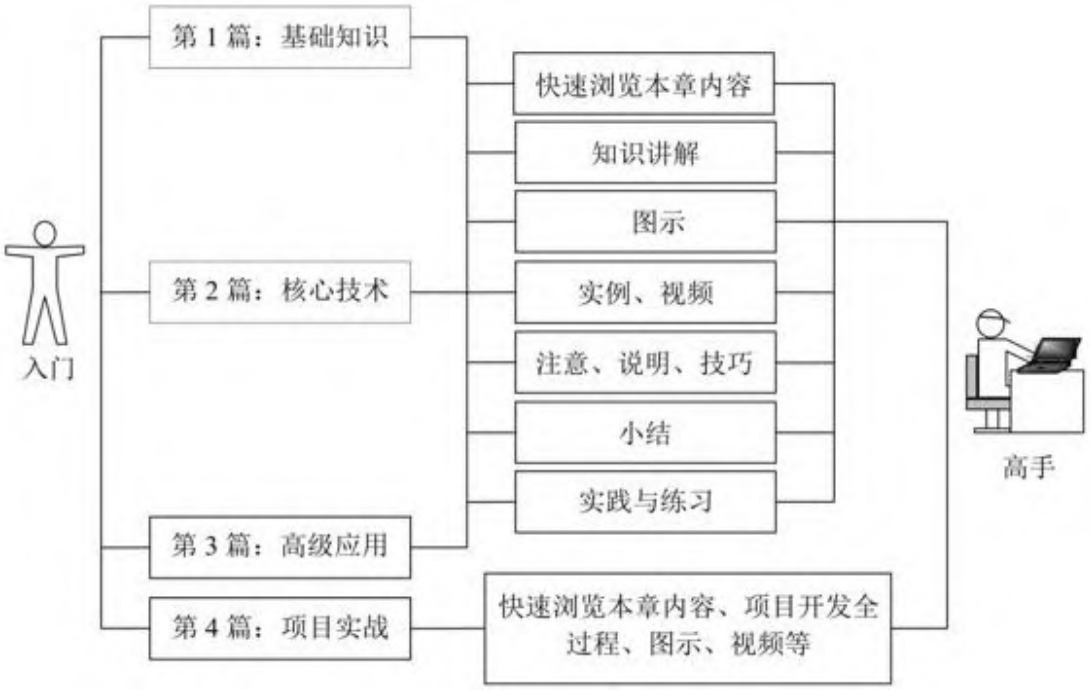
“软件开发视频大讲堂”丛书（第2版）于2010年8月出版，第3版于2012年8月出版，第4版于2016年10月出版，第5版于2019年3月出版。丛书连续畅销12年，迄今累计重印620次，销售400多万册。不仅深受广大程序员的喜爱，还被百余所高校选为计算机、软件等相关专业的教学参考用书。

“软件开发视频大讲堂”丛书（第6版）在继承前5版优点的基础上，将开发环境和工具更新为目前最新版本，并且重新录制了教学微课视频。并结合目前市场需要，进一步对丛书品种进行完善，对相关内容进行了更新优化，使之更适合读者学习。同时，为了方便教学使用，还提供了教学课件PPT。

C语言是Combined Language（组合语言）的简称。作为一种计算机设计语言，它同时具有高级语言和汇编语言两者的特点，因此受到广大编程人员的喜爱。C语言的应用非常广泛，既可以编写系统程序，也可以编写应用程序，还可以应用到单片机及嵌入式系统的开发中。这就是为什么大多数开发人员初学编程都选择C语言的原因。

本书内容

本书提供了从C语言入门到编程高手所必需的各类知识，共分为4篇，大体结构如下图所示。



第1篇：基础知识。本篇讲解了C语言基础知识，主要内容包括C语言的历史和特性、C语言的开发环境、算法、数据类型、运算符与表达式、常用的数据输入／输出函数、选择结构程序设计和循环控制等，本篇结合流程图和大量的实例，并通过视频的指导讲解，帮助读者夯实语法基础，为以后的编程学习奠定坚实的基础。

第2篇：核心技术。本篇介绍了C语言中的数组、函数和指针这三大部分内容，并将前面所学的基础内容融入其中，是更高级的程序设计内容。读者学习完这一部分，能够编写一些简单的C语言应用程序。

第3篇：高级应用。本篇介绍了结构体和共用体、位运算、预处理、文件、存储管理和网络套接字编程的内容。读者学习完这一部分，能够设计出较复杂的程序，并且涉及的范围更广。

第4篇：项目实战。本篇通过两个游戏项目，运用软件工程的设计思想，介绍如何进行软件项目的开发。书中按照“需求分析→系统设计→功能设计”的步骤，带领读者一步一步地亲身体验项目开发的全过程。

本书特点

☑ **由浅入深，循序渐进。**本书以初、中级程序员为对象，带领读者先从C语言基础学起，再学习C语言的程序结构，然后学习C语言的高级应用，最后学习开发两个游戏项目。讲解详尽，层次清晰，在叙述过程中会给出相应的实例，以便于读者理解。在讲解实例时分步骤进行，使读者一目了然，从而快速掌握书中内容。

☑ **微课视频，讲解详尽。**为便于读者直观感受程序开发的全过程，书中重要章节配备了教学微课视频（总时长13小时，共215集），使用手机扫描正文小节标题一侧的二维码，即可观看学习。便于初学者快速入门，感受编程的快乐和成就感，进一步增强学习的信心。

☑ **基础示例＋编程训练＋综合练习＋项目案例，实战为王。**通过例子学习是最好的学习方式，本书核心知识讲解通过“一个知识点、一个示例、一个结果、一段评析、一个综合应用”的模式，详尽透彻地讲述了实际开发中所需的各类知识。全书共计有178个应用示例，134个编程训练，128个综合练习，2个项目案例，为初学者打造“学习1小时，训练10小时”的强化实战学习环境。

☑ **精彩栏目，贴心提醒。**本书根据学习需要在正文中设计了很多“注意”“说明”“技巧”等小栏目，让读者在学习的过

程中更轻松地理解相关知识点及概念，更快地掌握个别技术的应用技巧。

☑ **海量资源，可查可练。**本书提供了强大的“Visual C++开发资源库”，包含实例资源库（881个实例）、模块资源库（15个典型模块）、项目资源库（15个项目案例）、测试题库系统（616道能力测试题）和面试资源库（371道企业面试真题）。

读者对象

☑ 自学编程的入门读者

☑ 编程爱好者

☑ 大中专院校的老师和学生

☑ 相关培训机构的老师和学员

☑ 做毕业设计的学生

☑ 初、中级程序开发人员

☑ 程序测试及维护人员

☑ 参加实习的“菜鸟”级程序员

读者服务

本书提供了大量的辅助学习资源，读者可扫描“文泉云盘”二维码，或登录清华大学出版社网站（www.tup.com.cn），在对应图书页面下查阅各类学习资源的获取方式。

文泉云盘



☑ 视频讲解资源

读者可先扫描图书二维码，获取学习权限，然后扫描各章节知识点、案例旁的二维码，观看对应的视频讲解。

☑ 拓展学习资源

读者可扫码登录清大文森学堂，获取本书的源代码、微课视频、开发资源库等资源，可参加辅导答疑直播课。同时，还可以获得更多的C语言进阶学习资源、职业成长知识图谱等，技术上释疑解惑，职业上交流成长。



清大文森学堂

致读者

本书由明日科技C语言程序开发团队组织编写。明日科技是一家专业从事软件开发、教育培训及软件开发教育资源整合的高科技公司，其编写的教材既注重选取软件开发中的必需、常用内容，又注重内容的易学、方便及相关知识的拓展，深受读者喜爱。其编写的教材

多次荣获“全行业优秀畅销品种”“中国大学出版社优秀畅销书”等奖项，多个品种长期位居同类图书销售排行榜的前列。

在本书编写的过程中，我们以科学、严谨的态度，力求精益求精，但疏漏之处在所难免，敬请广大读者批评指正。

感谢您购买本书，希望本书能成为您编程路上的领航者。

“零门槛”学编程，一切皆有可能。

祝读书快乐！

编　者

2021年7月

目 录

前言　Preface

第1篇　基础知识

第1章　C语言概述

1.1　C语言的发展史

1.1.1　程序语言简述

1.1.2　C语言的历史

1.2　C语言的特点

1.3　一个简单的C程序

1.4　一个完整的C程序

1.5　C语言程序的格式

1.6　C语言开发环境

1.6.1　Visual C++ 6.0

1.6.2　Visual Studio 2019

1.7　实践与练习

第2章　算法

2.1　算法的基本概念

2.1.1　算法的特性

2.1.2　算法的优劣

2.2　算法描述

2.2.1　自然语言

2.2.2　流程图

2.2.3　N-S流程图

2.3　实践与练习

第3章　数据类型

3.1　编程规范

3.2　关键字

3.3　标识符
3.4　数据类型
3.5　常量
3.5.1　整型常量
3.5.2　实型常量
3.5.3　字符型常量
3.5.4　转义字符
3.5.5　符号常量
3.6　变量
3.6.1　整型变量
3.6.2　实型变量
3.6.3　字符型变量
3.7　变量的存储类别
3.7.1　auto变量
3.7.2　static变量
3.7.3　register变量
3.7.4　extern变量
3.8　混合运算
3.9　实践与练习
第4章　运算符与表达式
4.1　表达式
4.2　赋值运算符与赋值表达式
4.2.1　变量赋初值
4.2.2　自动类型转换
4.2.3　强制类型转换
4.3　算术运算符与算术表达式
4.3.1　算术运算符
4.3.2　算术表达式
4.3.3　优先级与结合性
4.3.4　自增／自减运算符
4.4　关系运算符与关系表达式

4.4.1　关系运算符
4.4.2　关系表达式
4.4.3　优先级与结合性
4.5　逻辑运算符与逻辑表达式
4.5.1　逻辑运算符
4.5.2　逻辑表达式
4.5.3　优先级与结合性
4.6　逗号运算符与逗号表达式
4.7　复合赋值运算符
4.8　运算符的优先级总结
4.9　实践与练习
第5章　常用的数据输入／输出函数
5.1　语句
5.2　字符数据的输入／输出
5.2.1　字符输出函数
5.2.2　字符输入函数
5.3　字符串输入／输出
5.3.1　字符串输出函数
5.3.2　字符串输入函数
5.4　格式输出函数
5.5　格式输入函数
5.6　顺序程序设计应用
5.7　实践与练习
第6章　选择结构程序设计
6.1　if语句
6.2　if语句的基本形式
6.2.1　if语句形式
6.2.2　if…else语句形式
6.2.3　else if语句形式
6.3　if的嵌套形式
6.4　条件运算符

6.5　switch语句
6.5.1　switch语句的基本形式
6.5.2　多路开关模式的switch语句
6.6　if…else语句和switch语句的区别
6.7　实践与练习
第7章　循环控制
7.1　循环语句
7.2　while语句
7.3　do…while语句
7.4　for语句
7.4.1　for循环语句
7.4.2　for循环的变体
7.4.3　for循环中的逗号应用
7.5　3种循环语句的比较
7.6　循环嵌套
7.6.1　循环嵌套的结构
7.6.2　循环嵌套实例
7.7　转移语句
7.7.1　goto语句
7.7.2　break语句
7.7.3　continue语句
7.8　实践与练习
第2篇　核心技术
第8章　数组
8.1　一维数组
8.1.1　一维数组的定义和引用
8.1.2　一维数组的初始化
8.1.3　一维数组的应用
8.2　二维数组
8.2.1　二维数组的定义和引用

8.2.2　二维数组的初始化
8.2.3　二维数组的应用
8.3　字符数组
8.3.1　字符数组的定义和引用
8.3.2　字符数组的初始化
8.3.3　字符数组的结束标志
8.3.4　字符数组的输入和输出
8.3.5　字符数组的应用
8.4　多维数组
8.5　数组的排序算法
8.5.1　选择法排序
8.5.2　冒泡法排序
8.5.3　交换法排序
8.5.4　插入法排序
8.5.5　折半法排序
8.5.6　排序算法的比较
8.6　数组应用
8.6.1　反转输出字符串
8.6.2　输出系统日期和时间
8.6.3　字符串的加密和解密
8.7　实践与练习
第9章　函数
9.1　函数概述
9.2　函数的定义和声明
9.2.1　函数的定义
9.2.2　函数的声明
9.3　函数参数
9.3.1　形式参数与实际参数
9.3.2　数组作函数参数
9.4　函数的调用和返回
9.4.1　函数的调用

9.4.2　函数的返回
9.4.3　函数的嵌套调用
9.4.4　函数的递归调用
9.5　内部函数和外部函数
9.5.1　内部函数
9.5.2　外部函数
9.6　局部变量和全局变量
9.6.1　局部变量
9.6.2　全局变量
9.7　函数应用
9.7.1　数学函数
9.7.2　字符函数
9.7.3　字符串处理函数
9.8　实践与练习
第10章　指针
10.1　指针的相关概念
10.1.1　地址与指针
10.1.2　变量与指针
10.1.3　指针变量
10.1.4　指针的自增、自减运算
10.2　数组与指针
10.2.1　一维数组与指针
10.2.2　二维数组与指针
10.2.3　字符串与指针
10.2.4　指针数组
10.3　指向指针的指针
10.4　指针变量作函数参数
10.5　返回指针值的函数
10.6　指针数组作main函数的参数
10.7　实践与练习

第3篇　高级应用
第11章　结构体和共用体
11.1　结构体
11.1.1　结构体类型的声明
11.1.2　结构体变量的定义
11.1.3　结构体变量的引用
11.1.4　结构体类型的初始化
11.2　结构体数组
11.2.1　定义结构体数组
11.2.2　初始化结构体数组
11.3　结构体指针
11.3.1　指向结构体变量的指针
11.3.2　指向结构体数组的指针
11.3.3　结构体作为函数参数
11.4　嵌套的结构体
11.5　链表
11.5.1　链表概述
11.5.2　创建动态链表
11.5.3　输出链表
11.6　链表的相关操作
11.6.1　插入结点
11.6.2　删除结点
11.7　共用体
11.7.1　共用体的概念
11.7.2　共用体变量的引用
11.7.3　共用体变量的初始化
11.7.4　共用体类型的数据特点
11.8　枚举类型
11.9　实践与练习
第12章　位运算

12.1　位与字节
12.2　位运算操作符
12.2.1　按位与运算符
12.2.2　按位或运算符
12.2.3　按位取反运算符
12.2.4　按位异或运算符
12.2.5　左移运算符
12.2.6　右移运算符
12.3　循环移位
12.4　位段
12.4.1　位段的概念与定义
12.4.2　位段相关说明
12.5　实践与练习
第13章　预处理
13.1　宏定义
13.1.1　不带参数的宏定义
13.1.2　带参数的宏定义
13.2　#include命令
13.3　条件编译
13.3.1　#if命令
13.3.2　#ifdef及#ifndef命令
13.3.3　#undef命令
13.3.4　#line命令
13.3.5　#pragma命令
13.4　实践与练习
第14章　文件
14.1　文件概述
14.2　文件基本操作
14.2.1　文件指针
14.2.2　文件的打开
14.2.3　文件的关闭

14.3　文件的读写
14.3.1　fputc函数
14.3.2　fgetc函数
14.3.3　fputs函数
14.3.4　fgets函数
14.3.5　fprintf函数
14.3.6　fscanf函数
14.3.7　fread和fwrite函数
14.4　文件的定位
14.4.1　fseek函数
14.4.2　rewind函数
14.4.3　ftell函数
14.5　实践与练习
第15章　存储管理
15.1　内存组织方式
15.1.1　数据的存放方式
15.1.2　堆与栈
15.2　动态管理
15.2.1　malloc函数
15.2.2　calloc函数
15.2.3　realloc函数
15.2.4　free函数
15.3　内存泄露与内存丢失
15.4　实践与练习
第16章　网络套接字编程
16.1　计算机网络基础
16.1.1　IP地址
16.1.2　OSI七层参考模型
16.1.3　地址解析
16.1.4　域名解析
16.1.5　TCP/IP协议

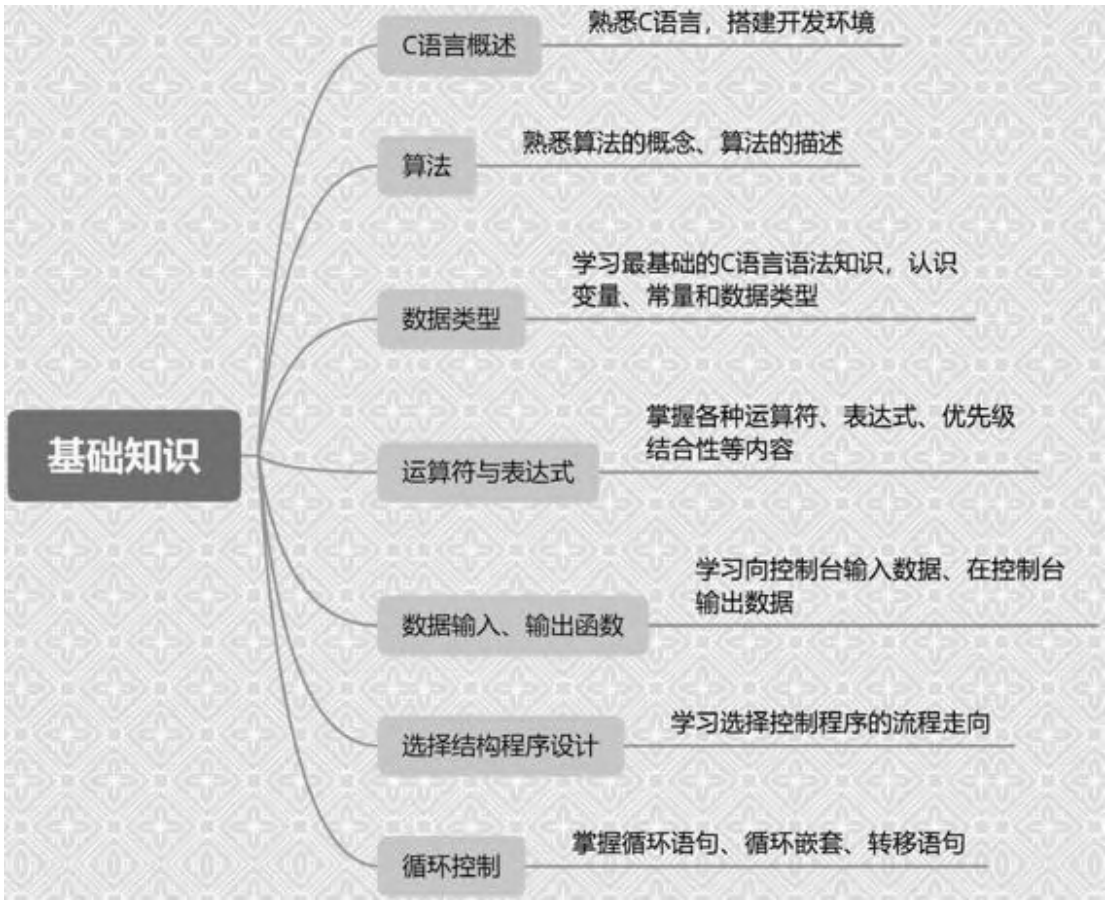
16.1.6　端口
16.1.7　套接字的引入
16.1.8　网络字节顺序
16.2　套接字基础
16.2.1　套接字概述
16.2.2　TCP套接字的socket编程
16.2.3　UDP套接字的socket编程
16.3　套接字函数
16.3.1　套接字常用函数介绍
16.3.2　基于TCP的网络聊天程序
16.4　实践与练习
第4篇　项目实战
第17章　单词背记闯关游戏
17.1　需求分析
17.2　系统设计
17.2.1　系统目标
17.2.2　构建开发环境
17.2.3　系统功能结构
17.2.4　业务流程图
17.2.5　系统预览
17.3　技术准备
17.3.1　文件的基本操作
17.3.2　文件的读写操作
17.4　公共类设计
17.5　预处理模块设计
17.5.1　模块概述
17.5.2　代码实现
17.6　欢迎界面设计
17.6.1　模块概述
17.6.2　代码实现

17.7　积分规则界面设计
17.7.1　模块概述
17.7.2　代码实现
17.8　显示游戏最高分设计
17.8.1　模块概述
17.8.2　代码实现
17.9　系统逻辑设计
17.9.1　模块概述
17.9.2　代码实现
17.10　结果显示界面设计
17.10.1　模块概述
17.10.2　代码实现
第18章　防空大战游戏
18.1　需求分析
18.2　系统设计
18.2.1　系统目标
18.2.2　构建开发环境
18.2.3　系统功能结构
18.2.4　业务流程图
18.2.5　系统预览
18.3　技术准备
18.3.1　EasyX图形库的下载与安装
18.3.2　EasyX库的主要函数
18.4　公共类设计
18.4.1　创建graphics.h头文件
18.4.2　源文件公设计
18.5　游戏主窗体设计
18.5.1　模块概述
18.5.2　创建防空导弹
18.5.3　创建炸弹
18.5.4　创建飞机

18.5.5　加载图片
18.5.6　绘图设计
18.6　碰撞检测设计
18.6.1　模块概述
18.6.2　代码实现
18.7　开始游戏设计
18.7.1　模块概述
18.7.2　加载音乐
18.7.3　飞机随机出现
18.7.4　按键设计
18.7.5　生命值和得分统计
18.7.6　游戏结束界面
18.7.7　重新开始游戏
18.8　为游戏应用添加图标
18.8.1　添加图标概述
18.8.2　实现步骤
附录　ASCII码对照表

第1篇
基础知识

只有具备扎实的基础知识，才能更快地掌握高级技术。本篇讲解了C语言的基础知识，包括C语言的历史和特性、C语言的开发环境、算法、C语言的数据类型、运算符与表达式、常用的数据输入／输出函数、选择结构程序设计和循环控制等。在学习过程中结合流程图和实例，并通过观看教学视频，读者可为今后的编程奠定坚实的基础。



第1章
C语言概述

在诸多的高级语言中，C语言是不折不扣的元老级语言。它是怎么发展而来的？有哪些特性？怎样才能使用它开发一个C程序？本章就带你来了解这些信息。除此以外，本章还会介绍Visual C++ 6.0和Visual Studio 2019开发环境，让你快速对C语言有一个全面的了解。深刻了解C语言，你会对编程产生浓厚的兴趣。

本章的知识架构及重难点如下：



1.1 C语言的发展史



1.1.1 程序语言简述

在介绍C语言之前，我们先来了解一下程序语言的发展历程。

1．机器语言

机器语言是低级语言，也称为二进制代码语言，是一种使用0、1表示的二进制代码编写指令以执行计算机操作的语言。机器语言的特点是计算机可以直接识别，不需要进行任何翻译。

2．汇编语言

汇编语言是面向机器的程序设计语言。为了减轻开发者使用二进制代码编程的痛苦，用英文字母或符号串来替代机器语言的二进制码，这样就把不易理解和使用的机器语言变成了汇编语言。因此，汇编语言要比机器语言更便于阅读和理解。

3．高级语言

由于汇编语言依赖于硬件体系，并且助记符号数量比较多，所以其运用起来仍然不够方便。为了使程序语言能更贴近人类的自然语言，同时又不依赖于计算机硬件，于是产生了高级语言。这种语言，其语法形式类似于英文，并且因为不需要对硬件进行直接操作，因此易于被普通人所理解与使用。其中影响较大、使用普遍的高级语言有Fortran、ALGOL、Basic、COBOL、LISP、Pascal、PROLOG、C、C++、VC、VB、Delphi、Java等。

1.1.2 C语言的历史

早期操作系统都是用汇编语言编写的，由于汇编语言依赖于计算机硬件，程序的可读性和可移植性都不是很好。为了提高可读性和可移植性，

人们开始寻找一种语言，这种语言既要具有高级语言的特性，又不能失去低级语言的优点。于是，C语言产生了。

C语言是在BCPL语言（简称B语言）的基础上发展和完善起来的，而B语言是由UNIX的研制者丹尼斯·里奇（Dennis Ritchie）和肯·汤普逊（Ken Thompson）于1970年研制出来的。20世纪70年代初期，AT&T Bell实验室的程序员丹尼斯·里奇第一次把B语言改为C语言。

最初，C语言运行于AT&T的多用户、多任务的UNIX操作系统上。后来，丹尼斯·里奇用C语言改写了UNIX C的编译程序，UNIX操作系统的开发者肯·汤普逊又用C语言成功地改写了UNIX，从此开创了编程史上的新篇章。UNIX成为第一个不是用汇编语言编写的主流操作系统。

1983年，美国国家标准委员会（ANSI）对C语言进行了标准化，于1983年颁布了第一个C语言草案（83ANSI C），后来于1987年又颁布了另一个C语言标准草案（87ANSI C），最新的C语言标准C99于1999年颁布，并在2000年3月被ANSI采用。但是由于未得到主流编译器厂家的支持，C99并未得到广泛使用。

尽管C语言是在大型商业机构和学术界的研究实验室中研发的，但是当开发者们为第一台个人计算机提供C编译系统之后，C语言就得以广泛传播，并为大多数程序员所接受。对MS-DOS操作系统来说，系统软件和实用程序都是用C语言编写的。Windows操作系统大部分也是用C语言编写的。

C语言是一种面向过程的语言，同时具有高级语言和汇编语言的优点。C语言可以广泛应用于不同的操作系统，如UNIX、MS-DOS、Microsoft Windows及Linux等。

在C语言基础上发展起来的有支持多种程序设计风格的C++语言，网络上广泛使用的Java和JavaScript，以及微软的C#语言等。也就是说，学好C语言之后，再学习其他语言就会比较轻松。

说明

目前流行的C语言版本有以下几个。

- ☑ Microsoft C或称MS C。
- ☑ Borland Turbo C或称Turbo C。
- ☑ AT&T C。

1.2 C语言的特点



C语言是一种通用的程序设计语言，主要用来进行系统程序设计，具有如下特点。

1．高效性

从C语言的发展历程可以看到，它继承了低级语言的优点，代码运行效率高，并具有良好的可读性和编写性。一般情况下，C语言生成的目标代码的执行效率只比汇编程序低10%~20%。

2．灵活性

C语言中的语法不拘一格，可在原有语法基础上进行创造、复合，从而给程序员更多想象和发挥的空间。

3．功能丰富

除C语言中所具有的类型外，还可以使用丰富的运算符和自定义结构类型来表达复杂的数据类型，完成所需要的功能。

4．表达力强

C语言的语法形式与人们惯用的自然语言相似，书写形式自由，结构规范，通过简单的控制语句即可轻松控制程序流程，完成烦琐的程序要求。

5．移植性好

作为高级语言，C语言具有良好的可移植性。在不同的操作系统下，只需要简单地修改甚至不用修改，即可进行跨平台的程序开发操作。

正是由于C语言拥有上述优点，因此它在程序开发中备受青睐。

1.3 一个简单的C程序



下面通过一个简单的实例来看看C语言程序是什么样的。为了便于讲解，实例代码中专门标注了行号（真实程序中这些行号是不存在的）。

【例1.1】输出一行语句（**实例位置：资源包\TM\sl\01\01**）

本实例程序的功能是显示一条信息“Welcome to MingRi”，通过这个程序可以初窥C程序的样貌。虽然这个简单的小程序只有7行，却充分说明了C程序是由什么位置开始、什么位置结束的。

```
1  #include<stdio.h>                    /*包含头文件 stdio.h*/
2
3  int main()                          /*主函数 main，这里是程序入口*/
4  {
5      printf("Welcome to MingRi\n");  /*输出要显示的字符串*/
6      return 0;                       /*程序返回 0*/
7  }
```

运行程序，显示效果如图1.1所示。

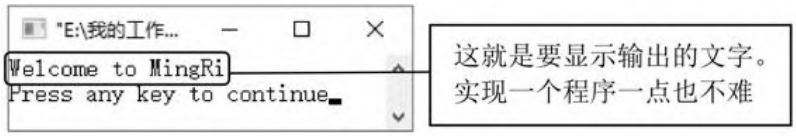


图1.1　一个简单的C程序

1．#include指令

实例代码的第1行是一个编译预处理命令，声明了在程序编译之前要预先处理的内容。

```
#include<stdio.h>                    /*包含头文件 stdio.h*/
```

其中，include称为文件包含命令，后面尖括号中的内容称为头文件或首文件。本行代码的作用是声明本段代码已包含标准输入／输出库函数stdio，后续可直接调用，不需要用户再进行定义。有关预处理的内容，将会在第13章中进行详细讲解，在此读者只需对相关概念有所了解即可。

2．空行

实例代码的第2行是一个空行。

C语言是一个相对灵活的语言，格式并不是固定不变的。也就是说，空格、空行、跳格并不会影响程序的运行。有的读者会问：为什么要有这么多空格和空行呢？其实这就像生活中在纸上写字一样，虽然拿来一张白纸就可以在上面写字，但是通常还会在纸上印上一行一行的方格或段落，

隔开每一段文字，使书写更加美观和规范。合理、恰当地使用这些空格、空行，可以使编写出来的程序更加规范、干净和易于阅读。

注意

> 并不是所有的空格都没用。比如，两个关键字之间要用空格隔开（else if），这种情况下如果将空格去掉，程序就不能通过编译。这里大家有个感性认识就好，在后续的学习中可以慢慢领悟。

3．main函数声明

实例代码的第3行如下：

```
int main()                                  /*主函数 main，这里是程序入口*/
```

本行代码的作用是声明main函数有一个整型返回值。其中的int称为关键字，该关键字代表的类型是整型。关于数据类型的内容将在第3章中讲解，关于函数的内容将在第9章中讲解。

函数中，这一部分称为函数头部分。每个C程序都必须有一个main函数，表示程序的入口部分。也就是说，程序都是从main函数头开始执行的，然后进入main函数体中，执行其中的代码块。

4．函数体

实例代码中的第4~7行如下：

```
{
    printf("Welcome to MingRi\n");          /*输出要显示的字符串*/
    return 0;                               /*程序返回 0*/
}
```

上面介绍main函数时，提到了一个名词“函数头”。既然有函数头，是不是也应该有函数体呢？没错，一个函数通常就分为两个部分：一

是函数头，二是函数体。

程序代码的第4行和第7行是两个大括号，这对大括号以及其中的语句块就构成了函数体。第5行和第6行语句就是函数体中要执行的内容。其中，printf函数就包含在标准输入／输出库文件stdio中，因为前面做了包含声明，所以这里开发者可以直接调用，进行字符串的输出操作。

5．执行语句

函数体中的第5行代码如下：

```
printf("Welcome to MingRi\n");                    /*输出要显示的字符串*/
```

执行语句就是函数体中要执行的动作内容。本行代码是这个简单例子中最复杂的语句，但其实不难理解。printf是格式化输出函数，用于向控制台输出文字或符号。括号中的内容称为函数参数，这里可以看到要输出的字符串为“Welcome to MingRi”。其中还可以看到“\n”这样一个符号，称之为转义字符，其作用是实现换行。关于转义字符的内容将第3章中介绍。

特别需要注意的是，语句的结尾一定要添加英文分号。

6．return语句

函数体中的第6行代码：

```
return 0;                                    /*程序返回 0*/
```

本行语句使main函数终止运行，并向操作系统返回一个整型常量0。前面介绍main函数时，说过它需要返回一个整型值，这里0就是要返回的整型值。可以将return语句理解成main函数的结束标志。

7．代码注释

在多句程序代码后，均可以看到一段由“/*”“*/”括起来的文字描述。例如：

```
printf("Welcome to MingRi\n");          /*输出要显示的字符串*/
return 0;                               /*程序返回 0*/
```

这段对代码的解释描述性文字称为代码注释，作用是为了他人能快速理解程序代码的含义和设计思想。注释内容包含在“/*”和“*/”之间，其语法格式如下：

```
/*其中为注释内容*/
```

注意

编写规范的代码格式和添加详细的注释，是一个优秀程序员应该具备的好习惯。

编程训练（答案位置：资源包\TM\sl\01\编程训练\）

训练1：打印经典语录　在控制台中输出马云在阿里巴巴上市时说的一句经典语录“梦想还是要有的，万一实现了呢！”。运行结果如下：

```
梦想还是要有的，万一实现了呢！
```

训练2：输出田字格图案　使用C程序在控制台中输出一个田字格图案。运行效果如下：（提示：可以使用搜狗输入法中的字符画）



1.4　一个完整的C程序



例1.1展现了一个简单的C程序，通过7行代码实现了显示一行字符串的功能。本节将在例1.1的基础上，对其内容进行扩充，使读者对C程序有一个更完整的认识。

【例1.2】计算长方形的体积（实例位置：资源包\TM\sl\01\02）

某长方体，其高已经给出，为10。要求从控制台输入长方体的长和宽，通过输入的长、宽以及给定的高度，计算出长方体的体积。

```
1   #include<stdio.h>                          /*包含头文件 stdio.h */
2   #define Height 10                          /*定义常量 Height*/
3   int calculate(int Long, int Width);        /*用户自定义函数 calculate 声明*/
4   int main()                                 /*主函数 main*/
5   {
6       int m_Long;                            /*定义整型变量 m_Long，表示长度*/
7       int m_Width;                           /*定义整型变量 m_Width，表示宽度*/
8       int result;                            /*定义整型变量 result，表示体积*/
9
10      printf("长方形的高度为：%d\n",Height);  /*显示提示*/
11
12      printf("请输入长度\n");                 /*显示提示*/
13      scanf("%d",&m_Long);                   /*输入长方体的长度*/
14
15      printf("请输入宽度\n");                 /*显示提示*/
16      scanf("%d",&m_Width);                  /*输入长方体的宽度*/
17
18      result=calculate(m_Long,m_Width);      /*调用 calculate 函数，计算体积*/
19      printf("长方体的体积是：");             /*显示提示*/
20      printf("%d\n",result);                 /*输出体积大小*/
21      return 0;                              /*程序返回 0*/
22  }
23
24  int calculate(int Long, int Width)         /*自定义计算体积函数 calculate */
25  {
26      int result =Long*Width*Height;         /*计算体积*/
27      return result;                         /*将计算的体积结果返回*/
28  }
```

运行程序，显示效果如图1.2所示。

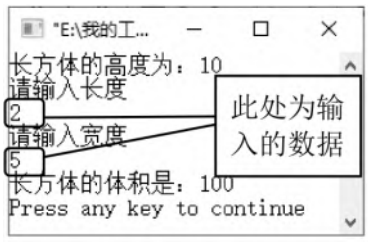


图1.2　一个完整的C程序

本例的程序执行流程如图1.3所示，从中可以观察出整个程序的运行过程。

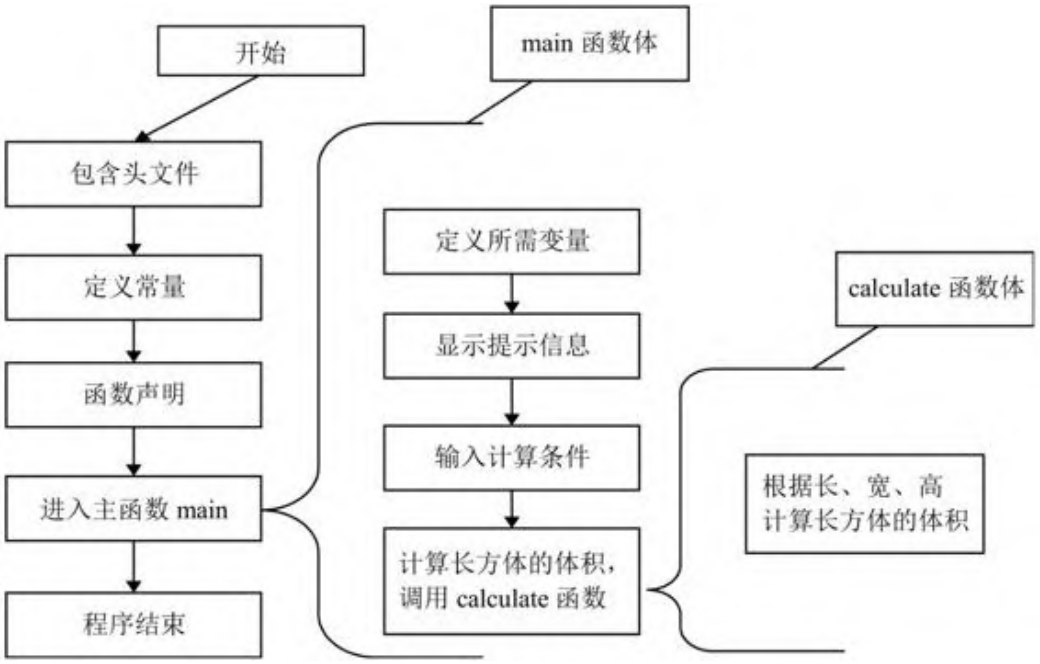


图1.3　程序流程分析

1.3节中已经介绍过#include包含命令、main函数、函数体、return返回语句、代码注释等内容，这里不再进行重复说明。下面介绍程序中新出现的一些内容。

说明

给出例1.1和例1.2，并不是为了讲解具体的知识点，而是为了将C程序的概貌展示出来，使读者对其有一个快速了解。还记得我们小时候初学加减法时的情况吗？老师只会教授1+1=2，却不会教授1+1为什么等于2，以及如何证明1+1=2这样的问题。学习C语言也应该如此，开始阶段大家不要过于深究为什么，有个简单的了解即可，在后续的不断接触中你会逐渐对其熟悉，直到能熟练应用。

1．定义常量

实例代码的第2行：

```
#define Height 10                    /*定义常量 Height */
```

本行代码使用#define定义了一个符号常量。#define在这里的功能是设定用符号Height代表数值10。这样在程序编译时，只要遇见Height标识符，就会被替换为数值10。从而满足题设中“其高已给出”这个条件。

2．函数声明

实例代码的第3行：

```
int calculate(int Long, int Width);          /*用户自定义函数 calculate 声明*/
```

本行代码的作用是对calculate函数进行声明。什么是函数声明呢？举例说明，两个公司进行合作，A公司要派一个经理到B公司进行业务洽谈。A公司会发送一个通知给B公司，告诉B公司会派一个经理过去，请B公司在机场接一下这位洽谈业务的经理。A公司会将这位经理的名字和大概的体貌特征告诉B公司的有关迎接人员，这样当这位经理下飞机之后，B公司就可以将他的名字写在纸上做成接机牌，然后找到这位经理。声明函数的作用，就类似于A公司告诉B公司前去谈判的经理信息的过程。也

就是说，此处声明calculate函数，表示后面将给出calculate函数的具体定义内容。

3．定义变量

实例代码的第6~8行：

```
int m_Long;                    /*定义整型变量 m_Long，表示长度*/
int m_Width;                   /*定义整型变量 m_Width，表示宽度*/
int result;                    /*定义整型变量 result，表示体积*/
```

这3行语句都是定义变量的语句。变量的作用就是存储数值，使得程序可利用变量进行计算。这就像在二元一次方程中，X和Y就是变量，当为其赋值后，如X为5，Y为10，这样X+Y的结果就等于15。C语言中，使用变量前必须先进行定义，编译器会根据变量类型为其分配内存空间。

4．输入语句

实例代码的第13行：

```
scanf("%d",&m_Long);           /*输入长方体的长度*/
```

例1.1中曾介绍过格式化输出函数printf，既然有输出函数，就一定有输入函数。C语言中，scanf函数用来接收键盘输入的内容，并将输入的内容保存在相应变量中。可以看到，在scanf函数的参数中，m_Long就是之前定义的整型变量，作用是存储输入的信息内容；“&”符号是取地址运算符，其具体内容会在后续章节中介绍。

5．自定义函数

实例代码的第24~28行：

```
int calculate(int Long, int Width)          /*自定义计算体积函数 calculate */
{
    int result =Long*Width*Height;          /*计算体积*/
    return result;                          /*将计算的体积结果返回*/
}
```

本段代码定义了一个calculate函数，作用是计算长方体的体积。前面介绍过main、printf、scanf函数，它们都属于C语言标准库函数，由系统提前定义好，读者直接调用就行。但calculate函数是用户根据需要自行定义的函数，因此必须给出定义的函数体。定义之后，读者一样可随时调用它。有关函数的具体内容将在第9章中介绍。

6．数学运算语句

实例代码的第26行：

```
int result =Long*Width*Height;                    /*计算体积*/
```

本行代码在calculate函数体内，其功能是将变量Long、Width、Height三者相乘得到的结果保存在result变量中。其中的“*”代表乘法运算符。

关于C语言程序，相信读者此时已经有了一定了解。下面再将例1.2的运行过程总结一下。

（1）包含程序所需要的头文件。

（2）定义一个常量Height，代表长方体的高度10。

（3）对calculate函数进行声明。

（4）进入main函数，程序开始执行。

（5）在main函数中，首先定义3个整型变量，分别代表长方体的长度、宽度和体积。

（6）显示提示文字，然后根据显示的文字输入有关的数据。

（7）当长方体的长度和宽度都输入之后，会调用calculate函数，计算长方体的体积。

（8）定义calculate函数的位置在main函数的下面，在calculate函数体内将计算长方体体积的结果进行返回。

（9）在main函数中，result变量得到了calculate函数返回的结果。

（10）通过输出语句将其中长方体的体积显示出来。

（11）程序结束。

编程训练（答案位置：资源包\TM\sl\01\编程训练\）

训练3：预测孩子身高　输入父亲和母亲的身高，通过计算公式“孩子身高＝（父亲身高＋母亲身高）*0.54”，预测孩子的身高。运行结果如下：

```
请输入父亲的身高：
1.8
请输入母亲的身高：
1.68
预测孩子的身高：1.88
```

训练4：米尺转化　在商场买裤子时，常常会听到“这条裤子腰围是2尺5”等类似的话。那么2尺5到底是多少米呢？试着编写程序实现米和尺之间的单位转换。运行结果如下：

```
请输入尺寸：
2.5
将尺转换成米是：0.83
```

1.5　C语言程序的格式



通过前面两节的学习，可以看出使用C语言编写程序有一定的格式特点。

☑ main函数是C程序的入口。

C程序都是从main函数开始执行的。也就是说，main函数不论放在什么位置都没有关系。

☑ C程序是由多个函数构成的。

除main函数外，还可以在程序中定义多个其他函数，并在这些自定义函数中完成特定的功能。虽然将所有程序代码全部放入main函数中也是可行的，但整个程序的可读性和扩展性将会变得很差。将一个C程序按功能分成多个块，每块使用一个函数来表示，整个程序将会具有结构性，既易于观察，又利于修改。

☑ 函数体的内容放在“{}”中。

每个函数都可以执行特定的功能，那么如何才能看出一个函数的具体操作范围呢？答案就是寻找“{”和“}”这两个大括号。C语言使用一对大括号来表示程序的结构层次，需要注意的就是左右大括号要成对使用。

技巧

编写程序时，为了防止对应大括号的遗漏，每次都可以先将两个对应的大括号写出来，再向括号中添加代码。

☑ 每个执行语句都以“;”结尾。

如果注意观察前面的两个实例，就会发现在每个执行语句后面都会有一个“;”（英文分号）作为语句结束的标志。

☑ 英文字符的大小写不通用。

同一个字母，其大小写的意义是不同的。注意，关键字和标准库函数名必须用小写。

☑ 要注意空格、空行的合理使用。

空格和空行的作用是增加程序的可读性，使得程序代码位置合理、美观。例如，如下代码就非常不利于观察：

```
int Add(int Num1, int Num2) /*定义计算加法函数*/
{/*将两个数相加的结果保存在result中*/
int result =Num1+Num2;
return result;/*将计算的结果返回*/}
```

将其中的执行语句按层级依次进行缩进，并排列代码注释，就会很有层次感。

```
int Add(int Num1, int Num2)          /*定义计算加法函数*/
{
    int result =Num1+Num2;           /*将两个数相加的结果保存在 result 中*/
    return result;                   /*将计算的结果返回*/
}
```

1.6 C语言开发环境

欲善工事，先利其器。本节将详细介绍两个C语言程序开发的常用工具，一个是Visual C++ 6.0，另一个是Visual Studio 2019。读者可根据需要选择使用其中一种开发环境。

1.6.1 Visual C++ 6.0



Visual C++ 6.0是一个功能强大的可视化软件开发工具，它将程序的代码编辑、程序编译、链接和调试等功能集于一身。其环境简单，界面友好，软件占用空间小，代码执行效率高，因此非常适合初学者学习使用。

1．Visual C++6.0的安装

微软公司已经停止了对Visual C++6.0的技术支持，并且也不提供下载。由于兼容性的问题，Windows 10系统需要安装Visual C++6.0的英文版。笔者采用的是在Windows 10系统下安装Visual C++6.0英文版，读者可以在网上搜索、下载合适的安装包并解压缩。下面介绍其安装过程。

（1）双击打开Visual C++6.0安装文件夹中的SETUP.EXE文件，如图1.4所示。

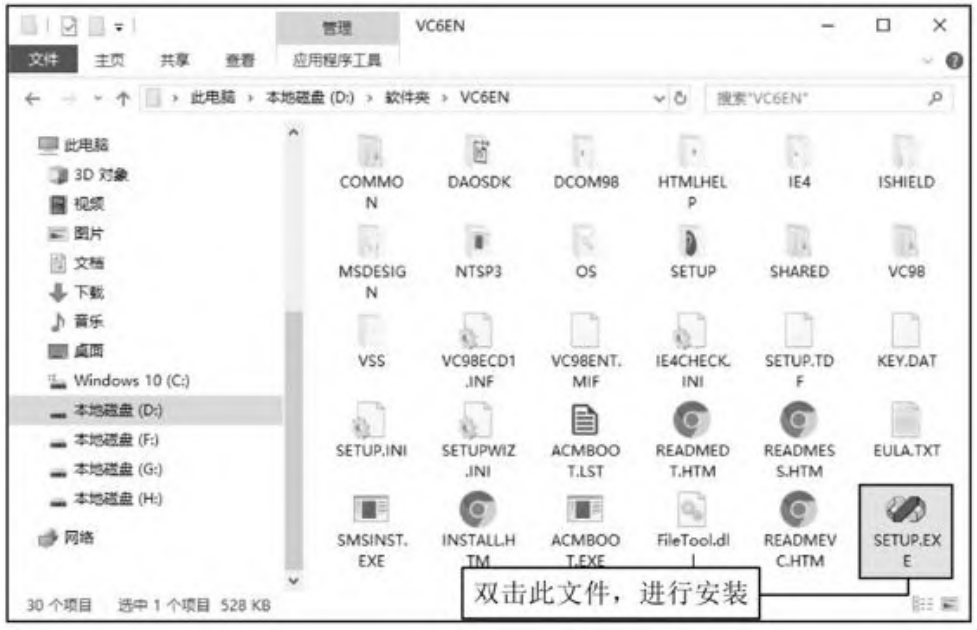


图1.4　双击SETUP.EXE安装文件

（2）进入安装向导界面，如图1.5所示，单击Next按钮。进入End User License Agreement（最终用户许可协议）界面，如图1.6所示，首先选中I accept the agreement单选按钮，然后单击Next按钮。

（3）进入Product Number and User ID（产品号和用户ID）界面，如图1.7所示。在安装包内找到CDKEY.txt文件，填写产品ID。姓名和公司名称根据情况填写，可以采用默认设置，不对其修改，单击Next按钮。



图1.5　安装向导界面

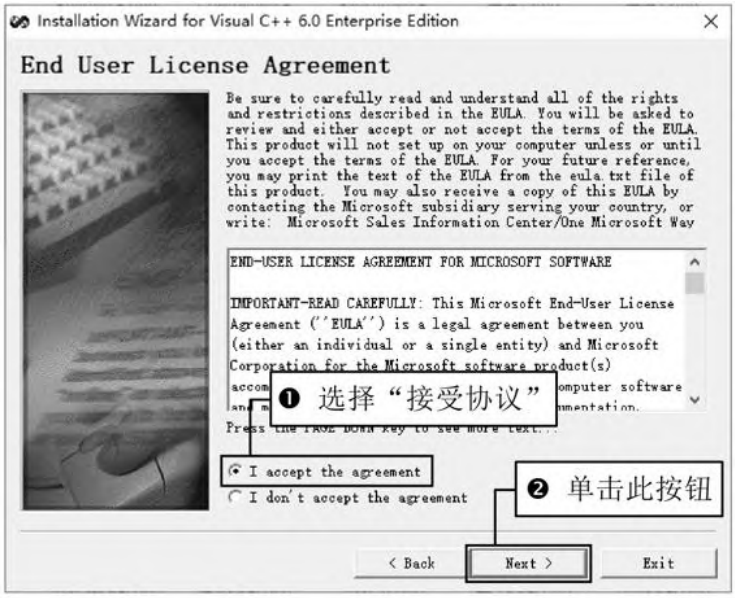


图1.6　“最终用户许可协议”界面

（4）进入Visual C++ 6.0 Enterprise Edition（Visual C++ 6.0英文企业版）界面，如图1.8所示。选中Install Visual C++ 6.0 Enterprise Edition单选按钮，然后单击Next按钮。

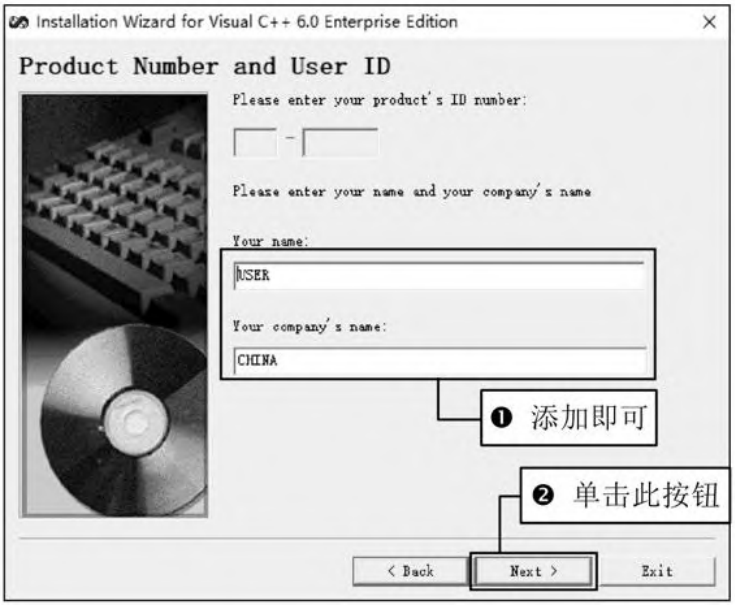


图1.7 “产品号和用户ID”界面

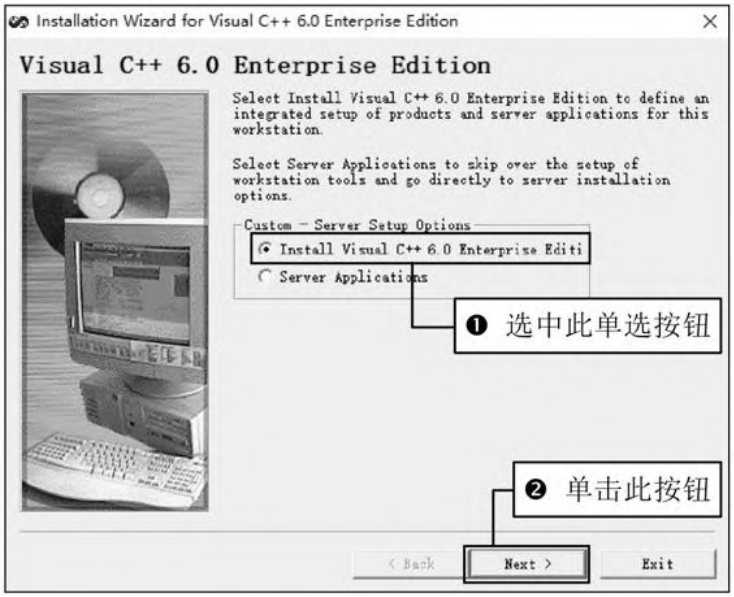


图1.8 “Visual C++ 6.0英文企业版”界面

（5）进入Choose Common Install Folder（选择公用安装文件夹）界面，如图1.9所示。公用文件默认存储在C盘中，单击Browse按钮，选择安装路径，这里建议安装在磁盘空间剩余比较大的磁盘中，以节约系统盘（C盘）空间。设置安装路径后，单击Next按钮。

（6）进入安装程序的欢迎界面中，如图1.10所示，单击“继续”按钮。

（7）进入产品ID确认界面，如图1.11所示，在此界面中，显示要安装的Visual C++6.0软件的产品ID，在向Microsoft请求技术支持时，需要提供此产品ID，单击“确定”按钮。

（8）如果读者计算机中安装过Visual C++6.0，即便已经卸载了，重新安装时还是会提示如图1.12所示的信息。安装软件检测到系统之前安装过Visual C++6.0，如果想要覆盖安装，单击“是”按钮；如果要将Visual C++6.0安装在其他位置，单击“否”按钮。这里单击“是”按钮，继续安装。

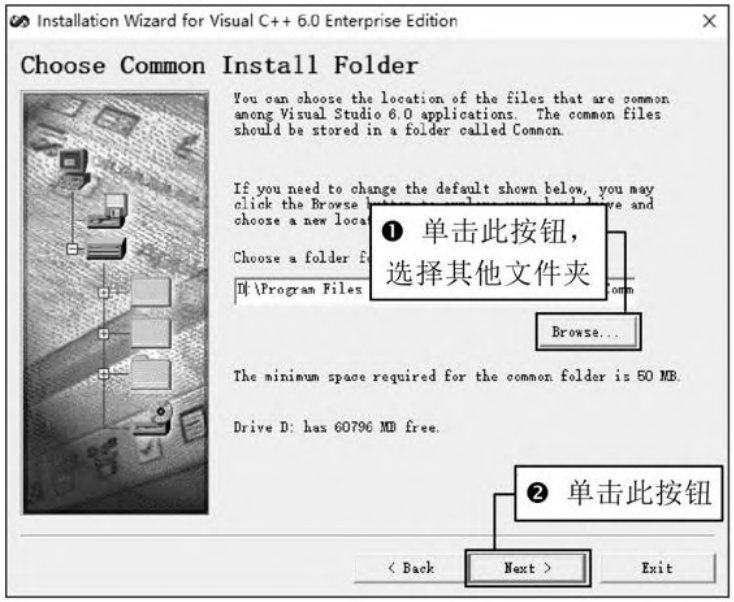


图1.9 “选择公用安装文件夹”对话框

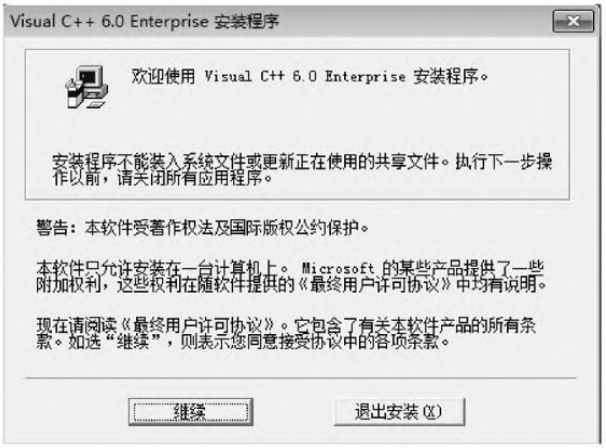


图1.10　安装程序的欢迎界面



图1.11　产品ID确认界面



图1.12　覆盖以前的安装

（9）进入选择安装类型界面，如图1.13所示。第一项Typical表示传统安装，第二项Custom表示自定义安装，这里选择Typical安装类型。

（10）进入注册环境变量界面，如图1.14所示，在此界面中，选中Register Environment Variables复选框，注册环境变量，单击OK按钮。



图1.13　选择安装类型界面

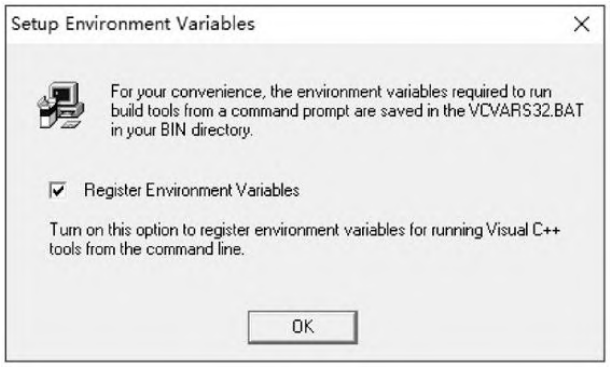


图1.14　注册环境变量界面

（11）弹出数据更新提示，如图1.15所示，单击“是”按钮，开始安装Visual C++6.0，如图1.16所示。

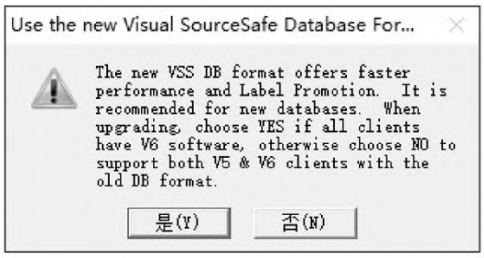


图1.15　数据更新界面

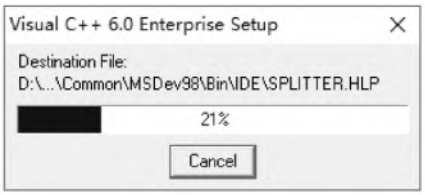


图1.16　安装进度条

（12）当进度条达到100%时，会进入如图1.17所示的界面，提示系统NT错误标志，这是由于Windows 10系统兼容性问题导致的，单击“确定”按钮即可。

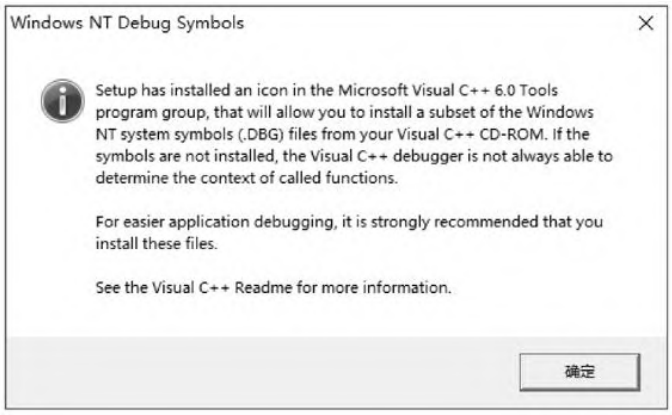


图1.17　系统NT错误界面

（13）完成以上操作后，会进入如图1.18所示的未响应界面。出现这种情况，同样是由于Windows 10系统兼容性问题导致的。这时单击鼠标，会弹出如图1.19所示的提示框，单击“关闭程序”选项，然后在“开始”菜单中找到Visual C++ 6.0图标并单击，就可以正常使用Visual C++ 6.0了。



图1.18　未响应界面



图1.19　关闭程序界面

2．Visual C++ 6.0的使用

Visual C++ 6.0程序的主界面如图1.20所示，下面来创建一个C语言源程序文件。

（1）在菜单栏中选择File→New命令，如图1.21所示，或者按Ctrl+N快捷键。

（2）打开New对话框，选择Files选项卡，在左侧的列表框中选择C++ Source File选项，右侧的File文本框中输入文件名称，Location文本框中设置文件保存地址，如图1.22所示，单击OK按钮，即可创建一个C源程序。

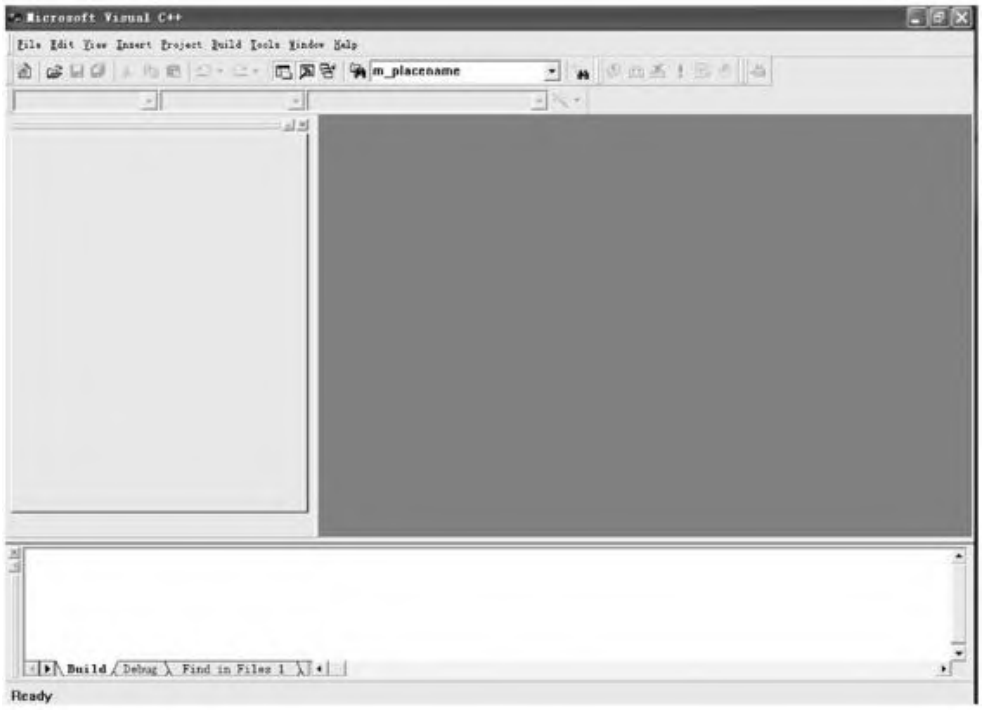


图1.20　Visual C++ 6.0界面

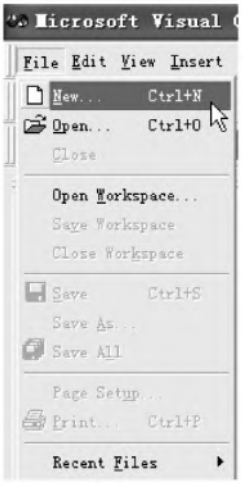


图1.21　创建一个新文件

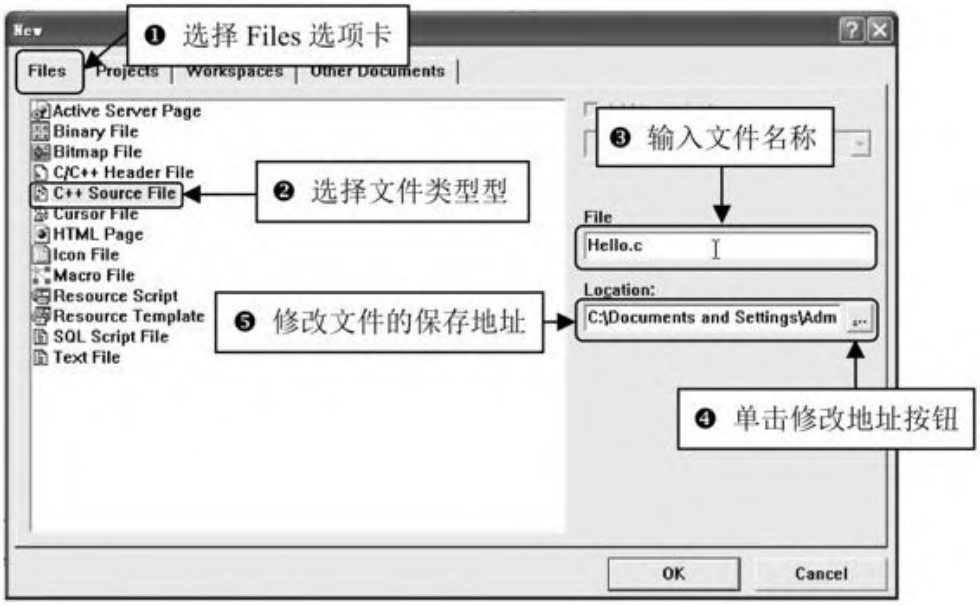


图1.22　创建C源文件

注意

因为创建的是C源文件，所以File文本框中要将C源文件的扩展名一起输入。例如，创建名称为Hello的C源文件，文本框中应输入“Hello.c”。

（3）此时可在开发环境中看到刚才创建的Hello.c源文件，如图1.23所示。

（4）将例1.1的程序代码输入其中，如图1.24所示。

（5）程序编写完成后，需要进行编译。选择Build→Compile命令，如图1.25所示。

（6）出现如图1.26所示的对话框，询问是否创建一个默认项目工作环境。单击“是”按钮，此时会询问是否要改动源文件的保存地址，如图1.27所示。



图1.23　新创建的文件

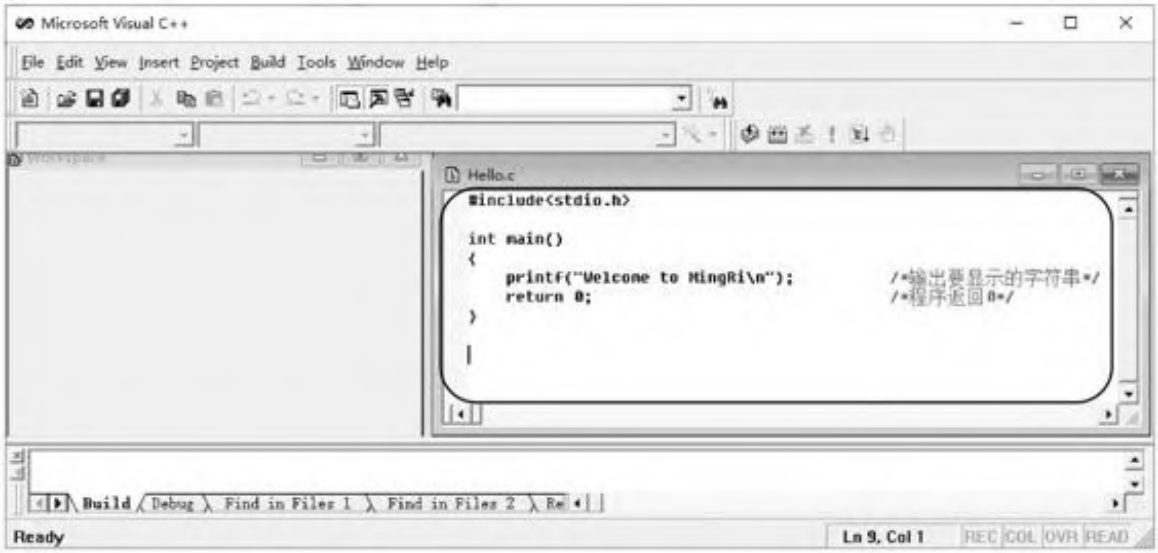


图1.24　输入程序代码

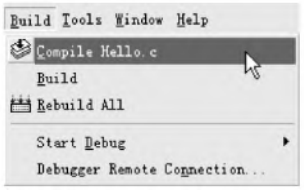


图1.25　选择Compile命令



图1.26　询问是否创建工作环境

（7）单击“是”按钮后，编译程序。如果程序没有错误，即可被成功编译。虽然此时代码已经被编译，但是还没有链接生成.exe可执行文件，因此如果此时要执行程序，会出现如图1.28所示的提示对话框，询问是否要创建.exe可执行文件。单击“是”按钮，则会链接生成.exe文件，即可执行程序。

（8）当然，也有直接创建.exe文件的操作选项。可以选择Build→Build命令，执行创建.exe文件操作，如图1.29所示。



图1.27　询问是否要改动源文件的保存地址

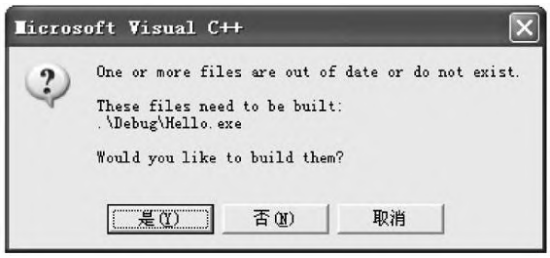


图1.28　询问是否要创建.exe文件

注意

在编译程序时可以直接选择Build命令进行编译、链接。这样就不用进行上面步骤（5）中的Compile操作，而可以直接将编译和链接操作一起执行。

（9）只有执行程序才可以看到有关程序执行的结果显示，可以选择Build→Execute命令进行执行程序操作，即可观察到程序的运行结果，如图1.30所示。



图1.29　选择Build命令

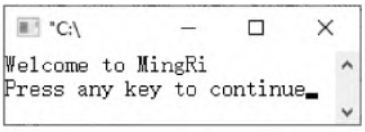


图1.30　程序运行结果显示

Visual C++ 6.0集成开发环境提供了一些有用的工具栏按钮。例如，代表Compile操作，代表Build操作，代表Execute操作。

编写程序时，使用快捷键会加快程序的编写进度。建议读者熟记如下常用操作的快捷键。

- ☑ Ctrl+N：创建一个新文件。
- ☑ Ctrl+]：检测程序中的括号是否匹配。
- ☑ F7：Build操作。
- ☑ Ctrl+F5：Execute（执行）操作。
- ☑ Alt+F8：整理多段不整齐的源代码。
- ☑ F5：调试操作。

为了便于读者阅读代码，可将程序运行结果的显示底色和文字进行修改。修改过程如下。

（1）按Ctrl+F5快捷键执行一个C程序，在程序标题栏上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，如图1.31所示。

（2）此时弹出“属性”对话框，在“颜色”选项卡中对“屏幕文字”和“屏幕背景”进行修改，如图1.32所示。在此读者可以根据自己的喜好设置颜色并显示。



图1.31　选择“属性”命令



图1.32　“颜色”选项卡

1.6.2　Visual Studio 2019

Microsoft Visual Studio（简称VS）是微软公司推出的开发工具集，也是目前最流行的Windows平台应用程序的集成开发环境之一。Visual Studio不仅可以编写C语言程序，还可以开发C++、C#、ASP.NET等，用其编写的代码适用于微软支持的所有平台。下面以Visual Studio 2019社区版的安装为例，讲解具体的安装过程和使用方法。

1．Visual Studio 2019的安装

Visual Studio 2019社区版是完全免费的，其下载地址为https://www.visualstudio.com/zh-hans/downloads/。安装文件是exe可执行文件，其命名格式为“vs_community__编译版本号.exe”。

（1）笔者下载的安装文件名为vs_community__1230733315.1531385802.exe文件，双击该文件开始安装。

说明

安装Visual Studio 2019开发环境前，需要先安装.NET Framework 4.6框架。如果没有安装，请先到微软官方网站下载并安装，下载地址为https://www.microsoft.com/zh-CN/download/details.aspx?id=48130。

（2）在安装程序界面中单击“继续”按钮，跳转到安装选项界面，如图1.33所示。选中“使用C++的桌面开发”复选框（其他复选框可根据

个人需要确定是否安装），在“位置”文本框中设置安装路径，建议安装在D盘或其他磁盘，最后单击“安装”按钮。

注意

在安装Visual Studio 2019开发环境时，一定要确保计算机处于联网状态，否则无法正常安装。

（3）跳转到安装进度界面，显示安装进度条，如图1.34所示。



图1.33 Visual Studio 2019安装界面

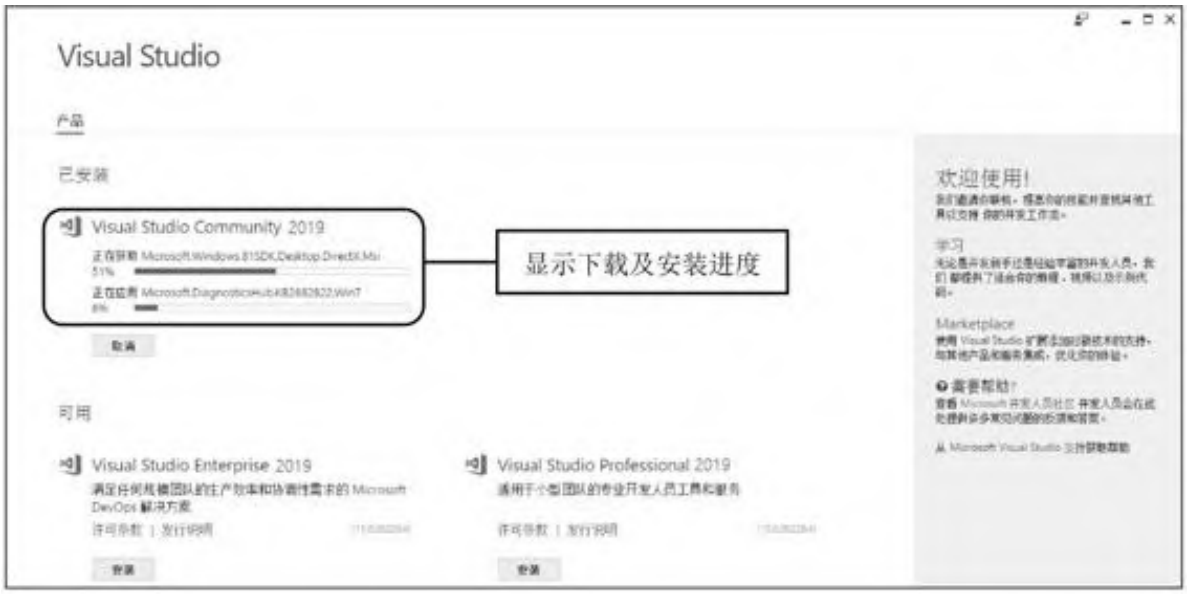


图1.34 Visual Studio 2019安装进度

（4）安装完成后，也就是进度条为100%时，会出现如图1.35所示的界面。单击“重启”按钮，完成Visual Studio 2019的安装。



图1.35 重启提示框

（5）重启计算机，在“开始”菜单中选择Visual Studio 2019命令，打开Visual Studio 2019。如果是第一次打开，会出现欢迎界面，单击“以后再说”按钮跳过。

（6）进入Visual Studio 2019环境的开发设置界面，如图1.36所示，在“开发设置”下拉列表框中选择Visual C++选项，颜色根据自己的喜好来选择，最后单击“启动Visual Studio”按钮。等待几秒钟后，即可进入开发欢迎界面。

2．Visual Studio 2019的使用

（1）打开Visual Studio 2019开发环境，单击“创建新项目”选项，如图1.37所示，然后在打开的界面中选择“空项目”选项，并单击

“下一步”按钮，如图1.38所示。

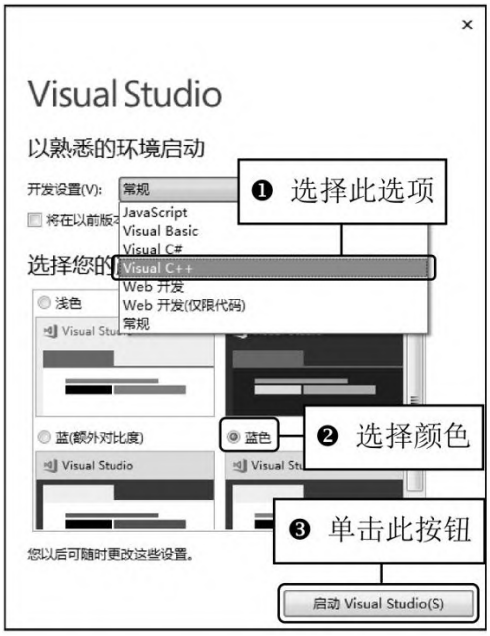


图1.36　开发设置界面



图1.37　创建新项目



图1.38　创建新文件

（2）打开“配置新项目”对话框，在“项目名称”文本框中输入文件名称，在“位置”下拉列表框中设置文件保存地址（可通过 ... 按钮修改文件存储位置），如图1.39所示。最后单击“创建”按钮，打开如图1.40所示的窗口。

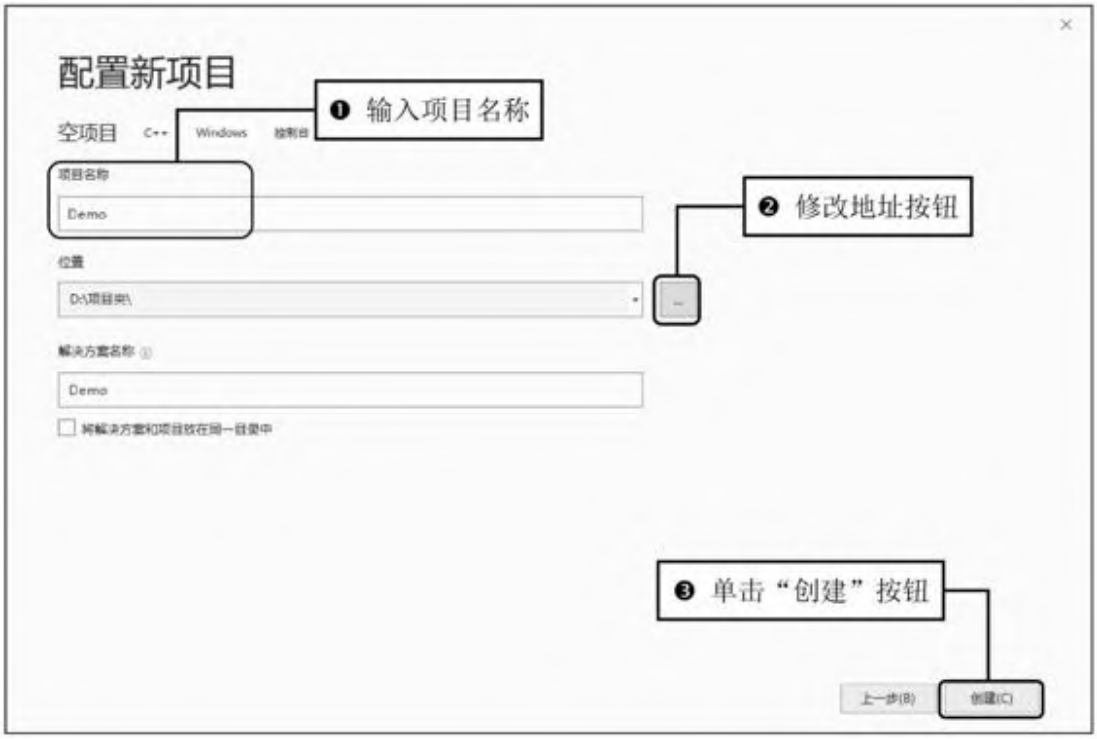


图1.39　创建C源文件

（3）选择“解决方案资源管理器”中的Demo，右击“源文件”选项，在弹出的快捷菜单中选择“添加”→“新建项”命令，如图1.41所示，或者按Shift+Ctrl+A组合键，打开“添加新项”对话框。

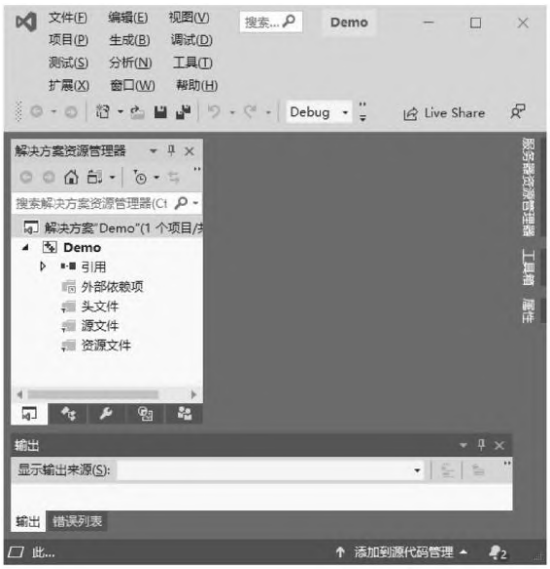


图1.40　创建项目界面



图1.41　添加项目界面

（4）如图1.42所示，左侧选择Visual C++选项，右侧列表框中将显示可创建的不同文件。因为要创建C源文件，因此这里选择 C++ 文件(.cpp) 选项，在下方的“名称”文本框中输入要创建的C文件名称，如demo.c，在“位置”下拉列表框中设置文件的保存地址，最后单击“添加”按钮。



图1.42　添加新项

注意

因为要创建C源文件，所以在文本框中将默认扩展名.cpp改为.c。例如，创建名称为demo的C源文件，文本框中应改为demo.c。

（5）此时已添加了一个C文件demo.c，如图1.43所示。

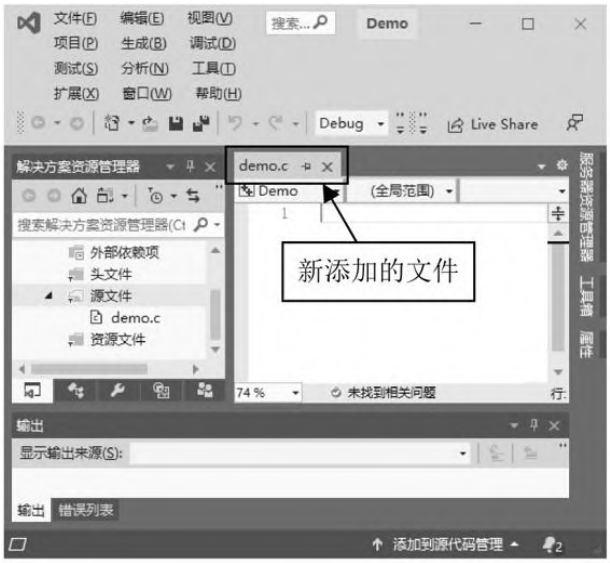


图1.43　完成添加C文件

（6）将代码写入demo.c文件中，如图1.44所示。

（7）代码编写完之后，接下来就要编译程序了。在菜单栏中选择“生成”→“编译”命令，如图1.45所示，或者按Ctrl+F7快捷键，编译程序。

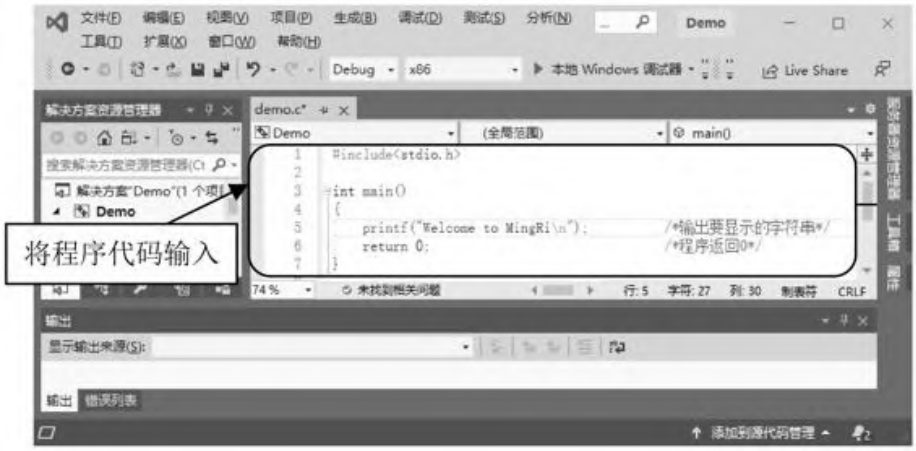


图1.44　输入代码



图1.45　编译程序

（8）编译结束后，如在输出工作空间的位置输出“生成：成功1个，失败0个，最新0个，跳过0个”，则表示编译成功。此时将得到一个可执行文件。

（9）在菜单栏中选择“调试”→“开始执行（不调试）”命令，如图1.46所示，或者按Ctrl+F5快捷键，运行程序，结果如图1.47所示。

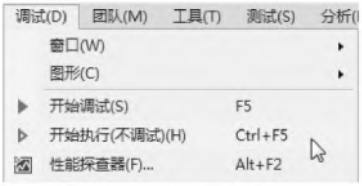


图1.46　运行程序

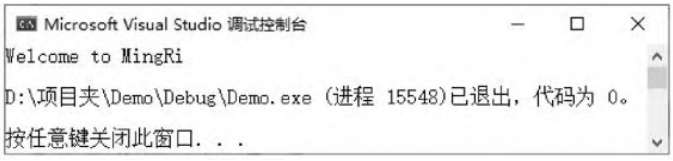
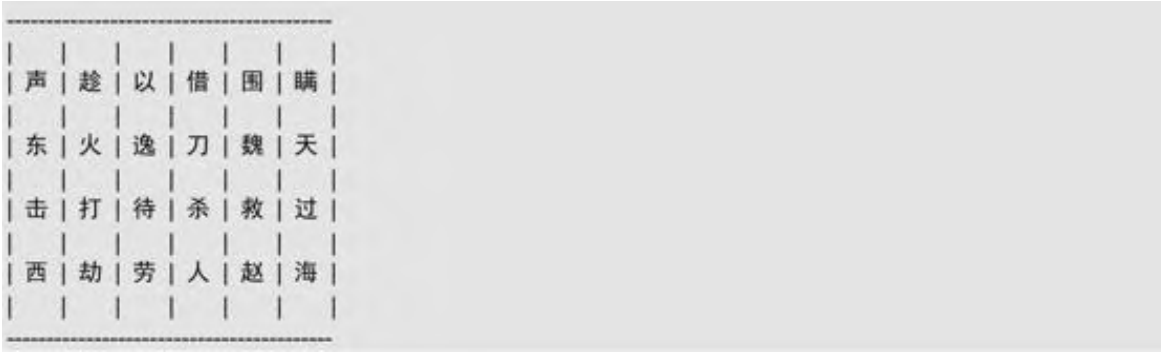


图1.47　运行结果

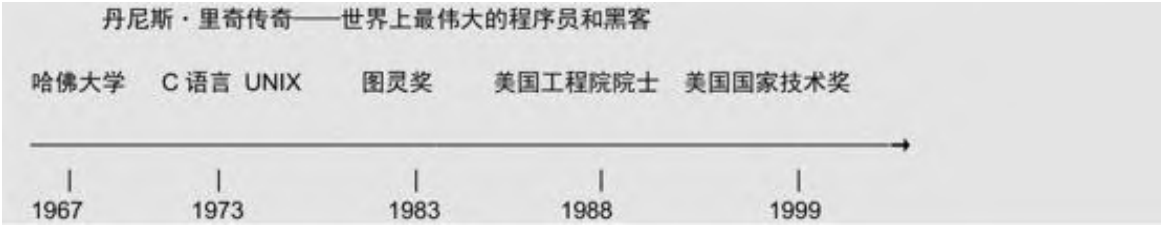
1.7　实践与练习

（答案位置：资源包\TM\sl\01\实践与练习\）

综合练习1：输出三十六计　编写程序，输出三十六计中的若干计策。实现效果如下：



综合练习2：输出丹尼斯·里奇的传奇人生　丹尼斯·里奇是现代计算机及网络技术的奠定者，也是C语言和UNIX之父。1973年，丹尼斯·里奇和肯·汤姆森在玩模拟太阳系航行的Space Trave游戏时发现机器中没有操作系统，于是联合开发了C语言，并用C语言开发了UNIX操作系统，从而拉开了程序开发时代的序幕。编写程序，输出丹尼斯·里奇的传奇人生经历。实现效果如下：



综合练习3：输出搜狐邮箱登录界面　参照图1.48，编写程序，输出模拟的搜狐邮箱登录界面，效果如图1.49所示。

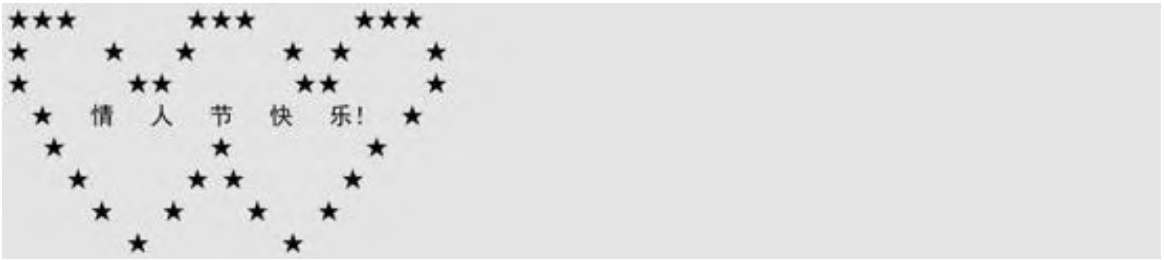


图1.48　参考图片



图1.49　实现效果

综合练习4：输出“情人节快乐”　每年的阳历2月14日和阴历7月7日是中西方的情人节，这是一个关于爱、浪漫以及花、巧克力、贺卡的节日，男女在这一天互换礼物，以表达爱意或友好。编写程序，打印输出如下图形。



综合练习5：输出微信个性签名　微信提供了个性签名功能，每个人都可以设置属于自己的个性签名。应用printf函数在IDLE窗口中输出如下个性签名。

```
-----------------------------------------------
|    再小的努力，乘以 365 都很明显    |
-----------------------------------------------
```

综合练习6：输出图书音像勋章　某网站设置了如图1.50所示的图书音像勋章，编写程序，模拟输出类似的效果。图标可以采用特殊符号“★、▲、◆、●”代替，也可以选用其他自己喜欢的小图标加以装饰，文字颜色用紫色输出，实现效果如图1.51所示。

综合练习7：输出京东秒杀界面　京东秒杀是京东商城旗下一款品牌特卖购物的应用，其倒计时效果如图1.52所示。编写程序，输出如图1.53所示的京东秒杀的文字版。

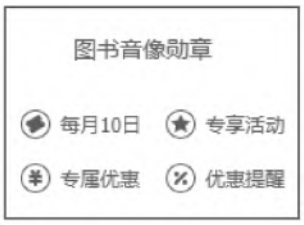


图1.50　参考图片



图1.51　实现效果



图1.52　参考图片



图1.53　实现效果

综合练习8：输出4399游戏登录界面　参照图1.54，输出模拟的4399游戏登录界面，效果如图1.55所示。



图1.54　参考图片

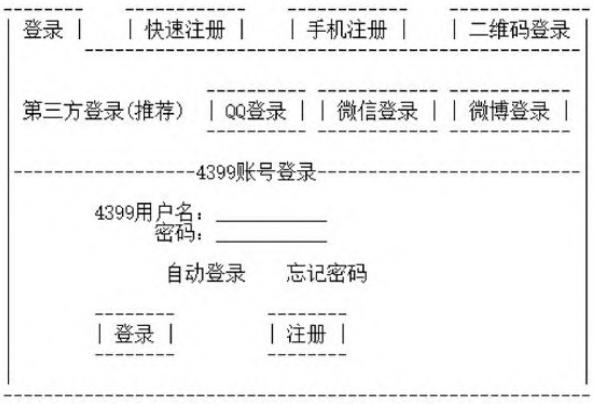


图1.55　实现效果

第2章
算法

开发一个程序通常要解决算法、数据结构、程序设计方法以及语言工具和环境这4个问题。其中，算法是核心，解决的是“做什么”和“如何做”的问题。正是因为算法非常重要，所以这里单独列出一章来介绍算法的基本知识。

本章的知识架构及重难点如下：



2.1 算法的基本概念

算法与程序设计及数据结构密切相关，是解决一个问题的完整的步骤描述，更是解决这个问题的策略、规则和方法。算法的描述形式有很多

种，如传统流程图、结构化流程图及计算机程序语言等，下面就介绍算法的一些相关内容。

2.1.1 算法的特性



算法是为解决某一特定类型的问题而制定的一个实现过程，它具有下列特性。

1．有穷性

一个算法必须在执行有穷步之后结束，且每一步都可在有穷时间内完成，不能无限地执行下去。如要编写一个由小到大整数累加的程序，就需要给出整数的上限，也就是加到哪个数为止。若没有上限，那么程序将无终止地运行下去，进入死循环。

2．确定性

算法的每一个步骤都应当有确切定义，每一个过程都不能有二义性，必须对将要执行的每个动作做出严格而清楚的规定。

3．可行性

算法中的每一步都应当能有效地运行，也就是说算法是可执行的，并能够最终得到正确的结果。如下面一段程序：

```
int x,y,z;
scanf("%d,%d,%d",&x,&y,&z);
if(y==0)
    z=x/y;
```

在这段代码中，“z=x/y;”就是一个无效的语句，因为0不可以做分母。

4．输入

一个算法应有零个或多个输入。输入就是执行算法时需要从外界取得的一些必要的（如算法所需的初始量等）信息。例如：

```
int a,b,c;
scanf("%d,%d,%d",&a,&b,&c);
```

上面的代码中有3个输入。又如：

```
main()
{
    printf("hello world!");
}
```

上面的代码需要零个输入。

5．输出

一个算法应有一个或多个输出。什么是输出？输出就是算法最终所求的结果。编写程序的目的就是要得到一个结果，如果一个程序运行下来没有任何结果，那么这个程序本身也就失去了意义。

误区警示

需要注意的是：一个程序，可能存在输入，也可能不存在输入，但一定存在输出。也就是说，至少存在一个输出。

2.1.2　算法的优劣



衡量一个算法的好坏，通常要从以下几个方面来分析。

1．正确性

正确性是指所写的算法应能满足具体问题的要求，即对任何合法的输入，算法都会得出正确的结果。

2．可读性

可读性是指算法被写好之后，该算法被理解的难易程度。一个算法可读性的好坏十分重要，如果一个算法比较抽象，难以理解，那么这个算法就不易于进行交流和推广使用，其后续修改、扩展、维护都十分不方便。因此在写一个算法时，要尽量将该算法写得简明、易懂。

3．健壮性

一个程序完成后，运行该程序的用户对程序的理解各有不同，并不能保证每一个人都能按照要求进行输入。健壮性就是指当输入的数据非法时，算法也会做出相应判断，而不会因为输入的错误造成瘫痪。

4．时间复杂度与空间复杂度

简单地说，时间复杂度就是算法运行所需要的时间。不同的算法具有不同的时间复杂度，当一个程序较小时，会感觉不到时间复杂度的重要性；但当一个程序特别大时，时间复杂度实际上是十分重要的。因此，如何写出更高速的算法一直是算法优化的目标。空间复杂度是指算法运行时所需的存储空间的大小。随着计算机硬件的发展，空间复杂度已经不再显得那么重要。

2.2 算法描述

算法包含算法设计和算法分析两个方面。算法设计主要研究怎样针对某一特定类型的问题设计出求解步骤，算法分析则要讨论所设计出来的算法步骤的正确性和复杂性。

对于一些问题的求解步骤，需要一种表达方式，即算法描述。他人可以通过算法描述来了解算法设计者的思路。表示一个算法，可以用不同的方法，常用的有自然语言法、流程图法、N-S流程图法等。

2.2.1 自然语言



自然语言就是人们日常所用的语言，这种表述方式通俗易懂，下面通过实例具体介绍。

【例2.1】求n！。

算法描述步骤如下:

（1）定义3个变量i、n及mul，为i和mul均赋初值为1。

（2）从键盘中输入一个数，赋给n。

（3）将mul乘以i的结果赋给mul。

（4）i的值加1，判断i的值是否大于n，如果大于n，则执行步骤（5），否则执行步骤（3）。

（5）将mul的结果输出。

【例2.2】农夫、羊、狼及白菜过河。

一名农夫要将一只狼、一只羊和一袋白菜运到河对岸。农夫的船很小，每次只能载下农夫本人以及狼、羊、白菜中的一个。但是，他不能把羊和白菜留在岸边，因为羊会把白菜吃掉；也不能把狼和羊留在岸边，因为狼会吃掉羊。那么，农夫该怎样将这3样东西送过河呢?

算法描述步骤如下:

（1）先把羊运过去。

（2）回来运狼。

（3）把狼运到对岸后，把羊装上船运回来。

（4）把羊放到开始的地方，把白菜运过去。

（5）再把羊运过去。

例2.1和例2.2的算法实现过程就是采用自然语言来描述的。从上面的描述中可以发现，自然语言描述的好处就是易懂，弊端是容易产生歧义。例如，将例2.1步骤（3）中的“将mul乘以i的结果赋给mul”改为“mul等于i乘以mul”，这样就产生了歧义。并且，用自然语言来描述较为复杂的算法时，会显得不是很方便，因此一般情况下不采用自然语言来描述。

2.2.2 流程图



流程图是一种传统的算法表示法，它用一些图框来代表各种不同性质的操作，用流程线来指示算法的执行方向。由于它直观形象，易于理解，所以应用广泛。特别是在语言发展的早期阶段，只有通过流程图才能简明地表述算法。

1．流程图符号

流程图使用一些图框来表示各种操作。如图2.1所示为一些常见的流程图符号，其中，起止框用来标识算法的开始和结束；判断框用于对一个给定的条件进行判断，根据条件成立与否来决定如何执行后续操作；输入／输出框用来表示输入／输出；处理框用来表示变量的计算或赋值；流程线用于表示算法的流向；注释框用于表示算法的注释；连接点用于将画在不同地方的流程线连接起来。

2．3种基本结构

Bohra和Jacopini为了提高算法的质量，提出了3种基本结构，即顺序结构、选择结构和循环结构，因为任何一个算法都可由这3种基本结构组成。这3种基本结构之间可以并列，可以相互包含，但不允许交叉，不允许从一个结构直接转到另一个结构的内部去。

任何算法都是由3种基本结构组成的，所以只要规定好3种基本结构的流程图的画法，就可以画出任何算法的流程图。

1）顺序结构

顺序结构是最简单的线性结构。在顺序结构的程序中，各操作按照它们出现的先后顺序执行。如图2.2所示，在执行完A框所指定的操作后，接着执行B框所指定的操作。这个结构中只有一个入口点A和一个出口点B。

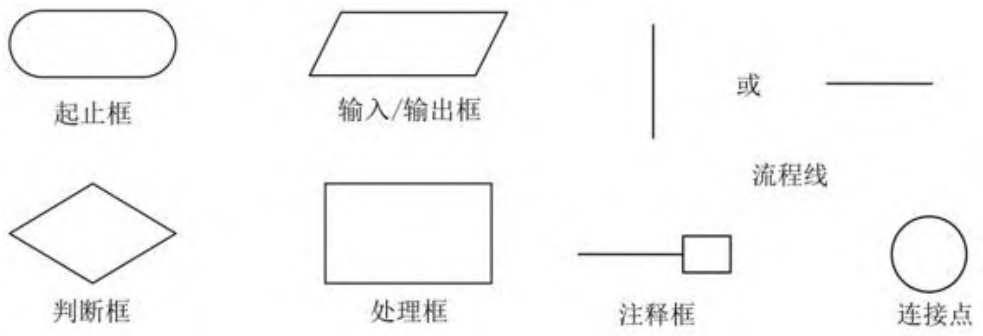


图2.1　流程图符号

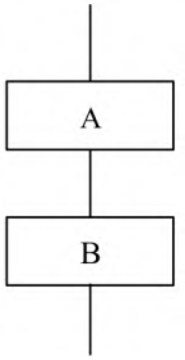


图2.2　顺序结构

【例2.3】输出数学、语文成绩。

输入两个数，分别代表数学、语文成绩，并分别赋给变量math和chinese，再将这两个数分别输出。

本实例的流程图可以采用顺序结构来实现，如图2.3所示。

2）选择结构

选择结构也称为分支结构，其常见形式有两种，如图2.4和图2.5所示。

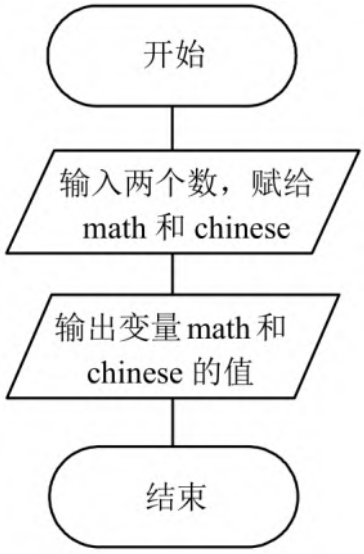


图2.3　输出数学、语文成绩

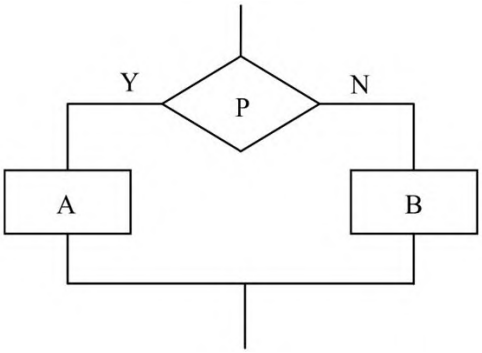


图2.4　选择结构1

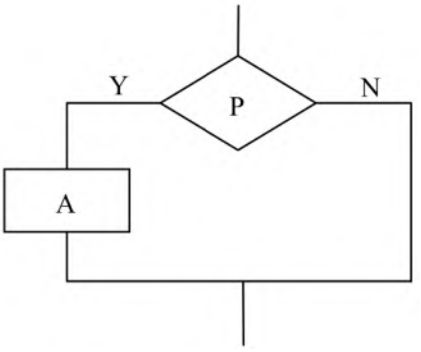


图2.5　选择结构2

选择结构中必须包含一个判断框。图2.4所代表的含义是根据给定的条件P是否成立选择执行A框还是B框。图2.5所代表的含义是根据给定的条件P进行判断，如果条件成立则执行A框，否则什么也不做。

【例2.4】判断是否为偶数。

输入一个数，判断该数是否为偶数，并给出相应提示。

本实例的流程图可以采用选择结构来实现，如图2.6所示。

3）循环结构

在循环结构中，反复地执行一系列操作，直到条件不成立时才终止循环。按照判断条件出现的位置，可将循环结构分为当型循环结构和直到型循环结构。

当型循环的流程结构如图2.7所示。先判断条件P是否成立，如果成立，则执行A框；执行完A框后，再判断条件P是否成立，如果成立，接着

再执行A框；如此反复，直到条件P不成立为止，此时不执行A框，跳出循环。

直到型循环的流程结构如图2.8所示。先执行A框，然后判断条件P是否成立，如果条件P成立则再执行A；然后判断条件P是否成立，如果成立，接着再执行A框；如此反复，直到条件P不成立，此时不执行A框，跳出循环。

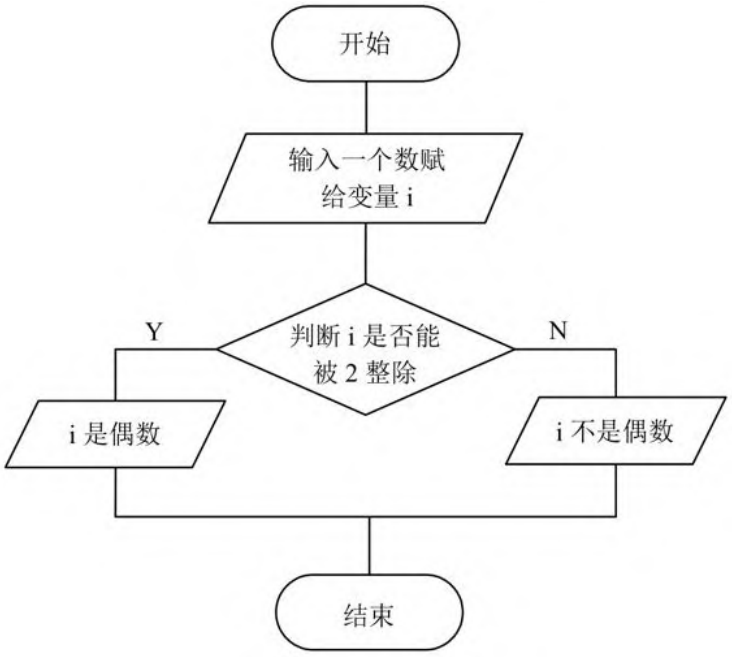


图2.6　判断一个数是否为偶数

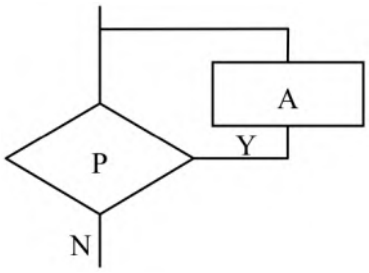


图2.7　当型循环

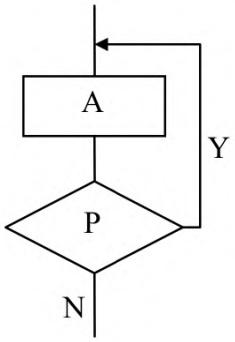


图2.8　直到型循环

【例2.5】计算1+2+3+…+100的结果。

求1和100之间（包括1和100）所有整数之和。

本实例的流程图可以用当型循环结构来表示，如图2.9所示。也可以用直到型循环结构来表示，如图2.10所示。

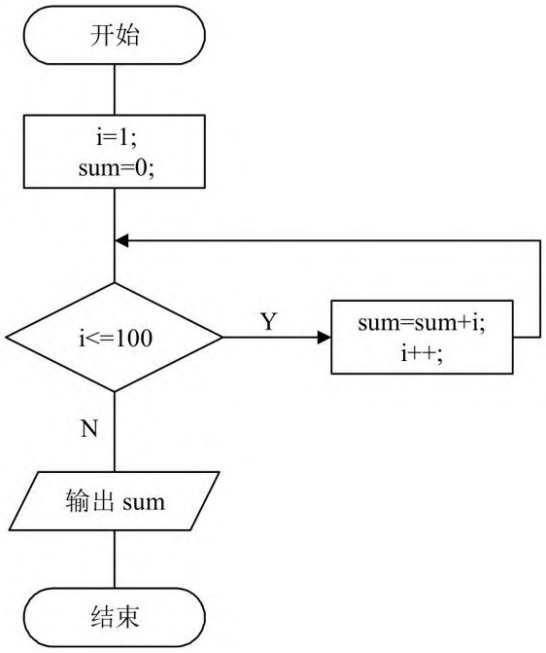


图2.9　当型循环结构求和

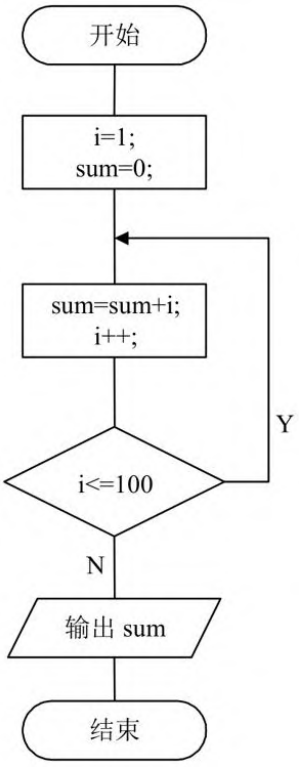


图2.10　直到型循环结构求和

2.2.3　N-S流程图



N-S流程图是另一种算法表示法，是由美国人I.Nassi和B.Shneiderman提出的。其根据是：既然任何算法都可以由顺序、选择和循环3种结构组成，则各基本结构之间的流程线就是多余的，因此可以去掉所有流程线，将全部的算法写在一个矩形框内。N-S图也是算法的一种结构化描述方法，同样也有3种基本结构，下面分别进行介绍。

1．顺序结构

顺序结构的N-S流程图如图2.11所示。例2.3的N-S流程图如图2.12所示。

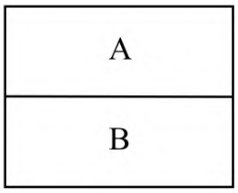


图2.11　顺序结构

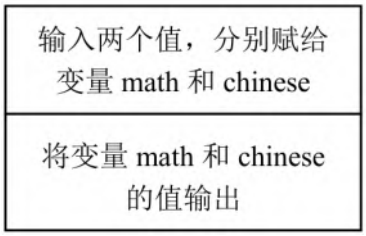


图2.12　输出数学、语文成绩

2．选择结构

选择结构的N-S流程图如图2.13所示。例2.4的N-S流程图如图2.14所示。

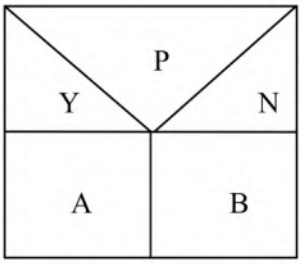


图2.13　选择结构

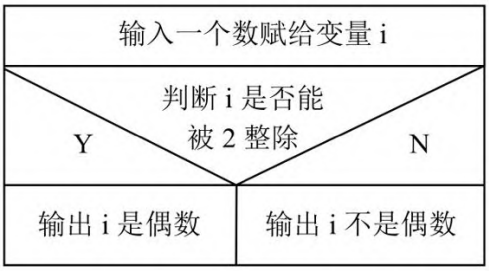


图2.14　判断一个数是否为偶数

3．循环结构

当型循环的N-S流程图如图2.15所示。例2.5的当型循环的N-S流程图如图2.16所示。

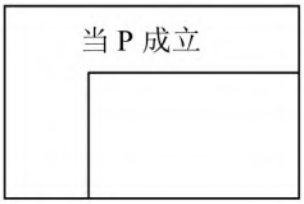


图2.15　当型循环

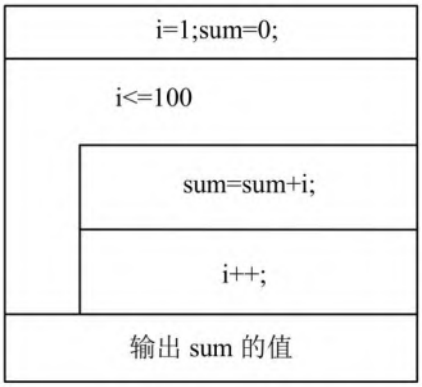


图2.16　当型循环求和

直到型循环的N-S图如图2.17所示。例2.5的直到型循环的N-S流程图如图2.18所示。

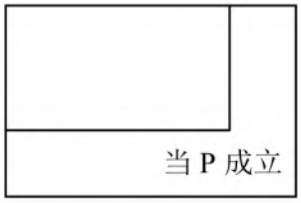


图2.17　直到型循环

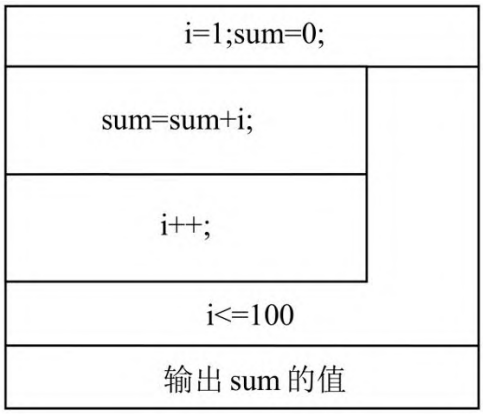


图2.18　直到型循环求和

说明

顺序、选择、循环3种基本结构都只有一个入口和一个出口，结构内的每一部分都有可能被执行，且不会出现无终止循环的情况。

【例2.6】计算n!。

从键盘中输入一个数n，求n!（提示：n!=1*2*3*4*…n）。分别用流程图和N-S图绘制其算法描述。

本实例的流程图如图2.19所示，N-S流程图如图2.20所示。

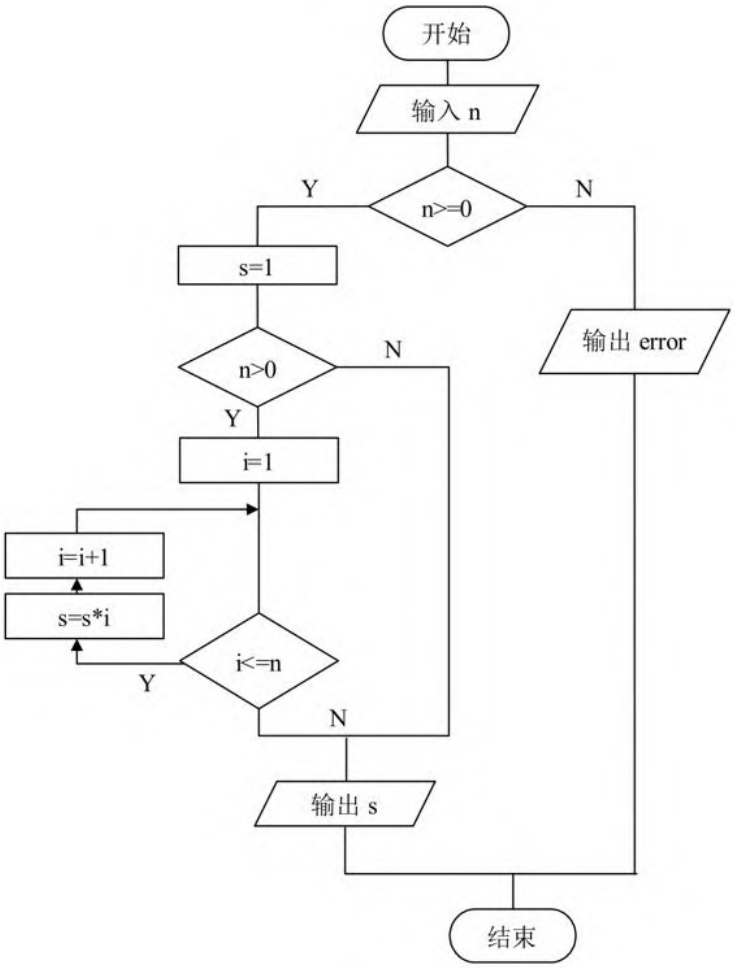


图2.19　求n!的流程图

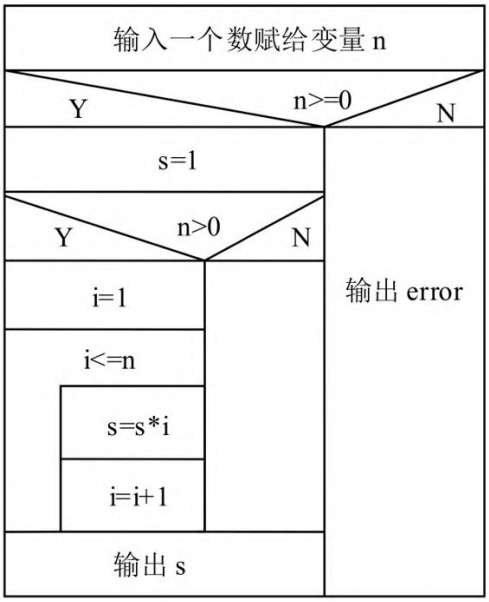


图2.20　求n!的N-S流程图

【例2.7】求a和b的最大公约数。

任意输入a、b值，利用顺序结构、选择结构、循环结构求解a、b的最大公约数。分别用流程图和N-S图绘制其算法描述。

本实例的流程图如图2.21所示，N-S流程图如图2.22所示。

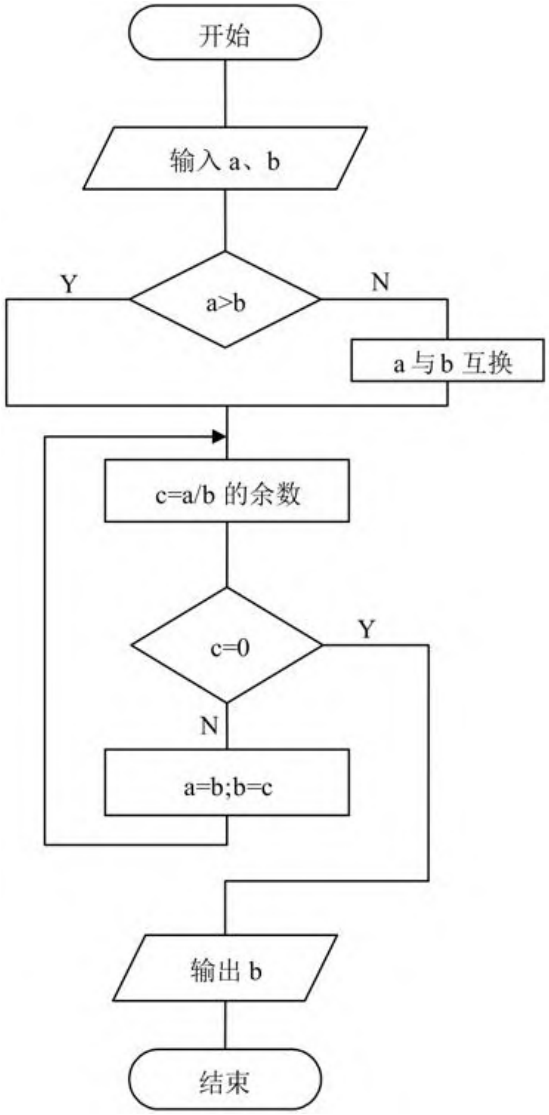


图2.21　求最大公约数的流程图

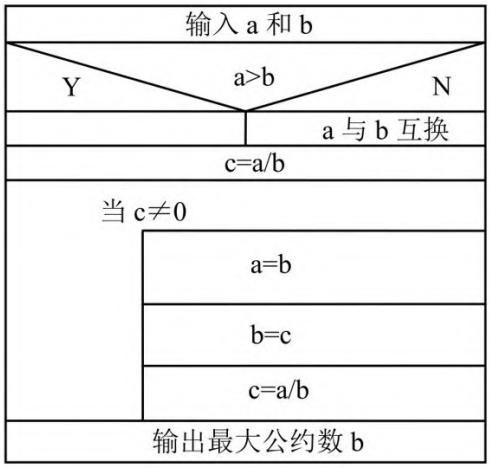


图2.22　求最大公约数的N-S流程图

2.3　实践与练习

（答案位置：资源包\TM\sl\02\实践与练习\）

综合练习1：把大象装进冰箱里　把大象装进冰箱里是比较经典的段子，大家都知道，总共分为3步：打开冰箱→把大象放进去→关上冰箱。用自然语言和N-S流程图描述把大象装进冰箱里的算法实现过程。

综合练习2：评定成绩等级　考试之后，老师通常会把学生的分数进行等级分类，例如等级A是90~100分；等级B是80~89分；等级C是60~79分；等级D是60分以下。利用流程图对学生成绩进行等级评定。

综合练习3：面积之争　使用关系运算符来比较扇形和圆锥谁的面积大（半径相等），用N-S流程图来表示面积之争的算法实现。

综合练习4：可乐还是咖啡　数字1代表选择喝CocaCola，否则就代表选择喝coffee，用流程图来表示要选择喝的东西。

综合练习5：健身房的邂逅　男同学在健身房跑步，本打算跑20分钟，但在10分钟的时候遇到了自己比较心仪的女同学，就停下跑步，去和女同学约会。用流程图来表示此过程的算法。（提示：用循环和break语句画流程图）

综合练习6：德邦物流承载量问题　已知德邦物流车长4.2米，宽1.9米，高1.9米，快递箱子长0.5米，宽0.5米，高0.3米，计算这个物流车能装多少个这样规格的箱子。计算公式为：箱子总数（取整数）=（物流车宽／快递箱宽）*（物流车长／快递箱长）*（物流车高／快递箱高），请绘制题目的流程图。

综合练习7：放假去哪儿嗨　使用if嵌套实现行程判断。外层if语句判断是否为周末，里层if语句判断周六、周日每天的活动，如果是周六就去长城，如果是周日就去故宫，如果是周一就开会，如果是周二到周五某一天，就正常工作。用流程图来模拟此场景的算法实现。

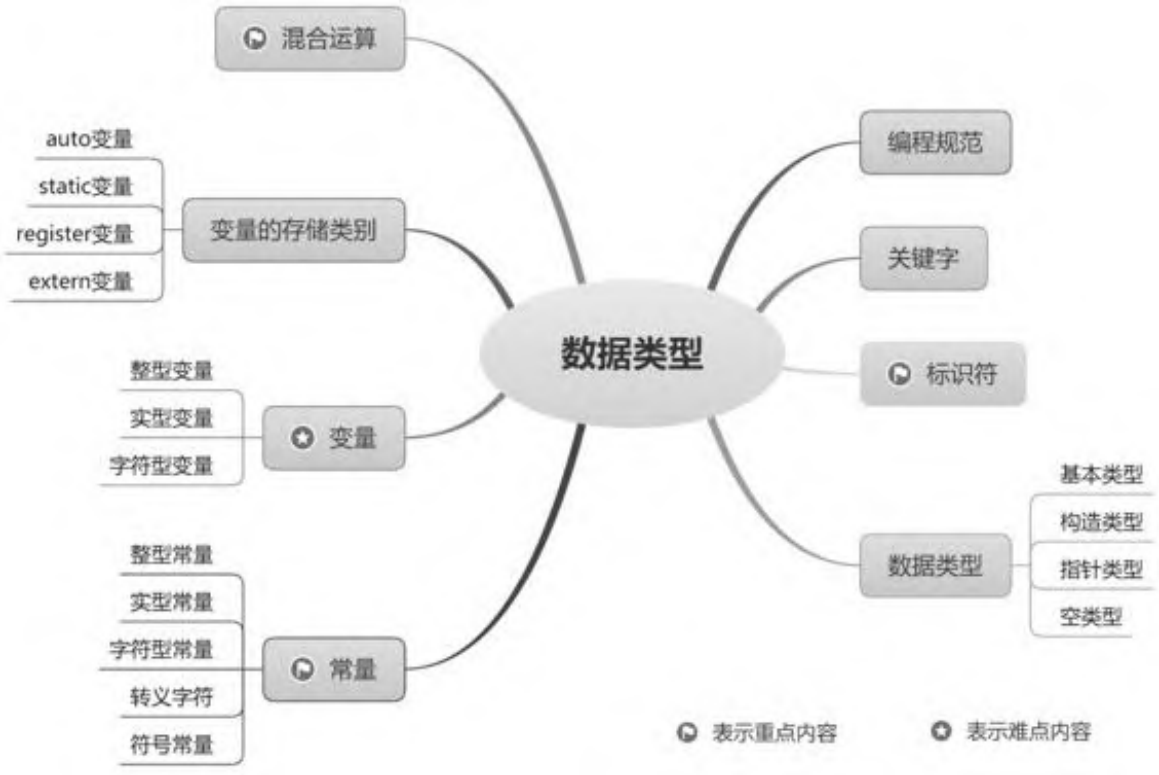
综合练习8：农夫卖西瓜　农夫一共有1020个西瓜，第一天卖掉一半多2个，第二天卖掉剩下的一半多2个，如此循环下去，需要卖几天才能卖完。用流程图模拟此问题的算法实现。

第3章
数据类型

学好了C语言，会很容易掌握其他编程语言，这是因为不同的编程语言有着一些共性存在。同时，一个好的程序员在编写代码时，一定要有规范性，清晰、整洁的代码才是有价值的。

本章致力于使读者掌握C语言中非常重要的一部分知识——数据类型、常量与变量。只有学懂了这些知识，才可以开始编写程序。

本章的知识架构及重难点如下：



3.1 编程规范



俗话说：没有规矩，不成方圆。虽然C语言中的代码书写形式相对自由，但是为了使编写的代码具有通用、友好的可读性，应尽量遵守一定的规范。

1．代码缩进

C语言中，代码按递进层次依次向右缩进，每次缩进4个字符。不采用空格，而采用Tab键制表位，以保证缩进的对齐。

```
#include<stdio.h>
int main()                              /*主函数 main*/
{
    int iResult=0;                      /*定义变量*/
    int i;
    printf("由 1 加到 100 的结果是：");   /*输出语句*/
    for(i=1;i<=100;i++)
    {
        iResult=i+iResult;
    }
    printf("%d\n",iResult);             /*输出结果*/
    return 0;                           /*结束返回*/
}
```

不同层次的代码缩进

2．变量、常量命名规范

常量命名统一为大写格式。如果是成员变量，均以m_开始。如果是普通变量，取与实际意义相关的名称，并且名称的首字母要大写，再在前面添加类型的首字母。如果是指针，则在其标识符前添加p字符，并且名称首字母要大写。例如：

```
#define   AGE  28                     /*定义常量*/
int m_iAge;                           /*定义整型成员变量*/
int iNumber;                          /*定义普通整型变量*/
int * pAge;                           /*定义指针变量*/
```

3．函数的命名规范

定义函数时，函数名的首字母要大写，其后的字母根据含义大小写混合。例如：

```
int AddTwoNum(int num1,int num2);
```

4．注释

尽量采用行注释。如果行注释与代码处于一行，则注释应位于代码右方。如果连续出现多个行注释，并且代码较短，则应对齐注释。例如：

```
int iLong;                    /*长度*/
int iWidth;                   /*宽度*/
int iHeight                   /*高度*/
```

3.2　关键字



关键字（Keywords），又称为保留字，是C语言规定的具有特定意义的字符串。C语言中有32个关键字，如表3.1所示。用户定义的常量、变量、函数等名称不能与关键字相同，否则会出现错误。

表3.1　C语言中的关键字

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	union	return
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	while	static	if

注意

这32个关键字不需要背诵。在C语言开发环境中编写代码时，关键字会自动显示为特殊字体（如变成蓝色）。今后的学习中会逐渐接触到这些关键字的具体使用方法，这里不需要死记硬背。

3.3 标识符



C语言中，为了在程序的运行过程中可以使用变量、常量、函数、数组等，就要为这些形式设定一个名称，而设定的名称就是所谓的标识符。

标识符的名称设定比较自由，开发者可以设定自己喜欢且容易理解的各类名称。但这并不意味着可以随意自由发挥，下面介绍设定C语言标识符时应该遵守的一些基本命名规则。

（1）标识符必须以字母或下画线开头，而不能以数字或者符号开头。例如：

```
int $num;                    /*错误，标识符第一个字符不能为符号*/
int 2hao;                    /*错误，标识符第一个字符不能为数字*/
int num;                     /*正确，标识符第一个字符为字母*/
int _hao;                    /*正确，标识符第一个字符为下画线*/
```

（2）标识符中，除开头外的其他字符可以由字母、下画线或数字组成。例如：

```
int good_way;                /*正确，标识符中可以有下画线*/
int bus7;                    /*正确，标识符中可以有数字*/
int car6V;                   /*正确*/
```

（3）C语言区分大小写，也就是说，大写或小写的英文字母代表的是不同的标识符。例如，下面的代码给出了3个不同的标识符。

```
int mingri;                    /*全部是小写*/
int MINGRI;                    /*全部是大写*/
int MingRi;                    /*一部分是小写，一部分是大写*/
```

可以看出，标识符中只要有一个字符不同，其代表的就是一个新的名称。

（4）标识符不能是关键字。例如，系统使用int关键字表示整型数据类型，因此用户定义的标识符就不能再使用int，否则会提示编译错误。但将其中的某个字母改写成大写字母后，就可以通过编译。

```
int int;                       /*错误，int 为系统关键字*/
int Int;                       /*正确，Int 不再是关键字*/
```

（5）标识符应体现一定的功能含义，便于理解。通常采用所表述内容的英文、拼音等，这样即便是没有注释，只直观地查看代码，也能快速了解到其表述的作用及功能。

例如，定义一个长方体的长、宽和高时，如果将其简单定义为a、b、c，则没有后面的注释就很难理解。相比之下，iLong、iWidth、iHeight这样的标识符更清晰、明了，推荐大家采用。

```
int a;                         /*长方体的长*/
int b;                         /*长方体的宽*/
int c;                         /*长方体的高*/
int iLong;                     /*长方体的长*/
int iWidth;                    /*长方体的宽*/
int iHeight;                   /*长方体的高*/
```

误区警示

标识符必须使用英文，是一个不精准的说法。开发环境不同，要求也不同。Visual C++ 6.0中，使用中文标识符会出现错误，但Visual Studio 2109中使用中文标识符编译器并不会报错。综合考

量，最好不要使用中文写代码，以免出现不必要的错误。另外，写代码时要多写注释，养成好习惯。

（6）ANSI标准规定，标识符可以为任意长度，但外部名必须保证仅前6个字符就能唯一地进行区分。这是因为某些编译程序（如IBM PC的MS C）仅能识别前6个字符。

注意

编写代码和定义标识符时需要注意以下方面。

（1）书写标识符时，要注意大小写的区分。

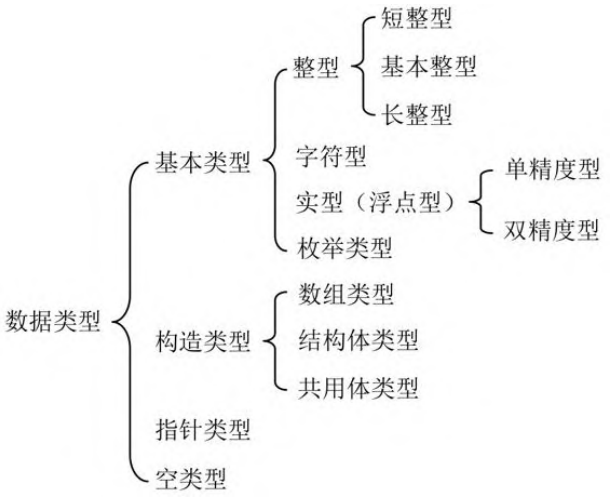
（2）书写代码时，应先调整输入状态为“英文半角输入法”。

3.4 数据类型



程序要解决复杂的问题，就要处理不同的数据。不同的数据类型，如整型、字符型、实型等，在内存中占用的存储空间大小也不同。

C语言中的数据类型包括基本类型、构造类型、指针类型和空类型等，如图3.1所示。



1．基本类型

图3.1　数据类型

基本类型是C语言中的基础类型，包括整型、字符型、实型（浮点型）、枚举类型4类。

2．构造类型

当基本类型不能满足需要时，开发者可以自行构造数据类型。即使用基本类型数据，或已构造好的数据类型，通过添加、设计、组合，形成能满足待解决问题需要的新的数据类型。

构造类型是由多种类型组合而成的新类型，其每个组成部分都称为成员。构造类型包括数组类型、结构体类型和共用体类型3类。

3．指针类型

指针是C语言的精华。指针类型不同于其他类型，因为其值表示的是某个内存地址。

4．空类型

空类型的关键字是void，主要用于对函数返回值和函数参数进行限定。

函数一般具有一个返回值（返给调用者），这个返回值应该具有特定的类型，如整型int。当函数不需要返回值时，就可以使用空类型进行设定。

3.5　常量

常量就是其值在程序运行过程中不可以改变的量，包括3类：数值型常量（包括整型常量和实型常量）、字符型常量和符号常量。下面分别进行说明。

3.5.1 整型常量



整型常量就是整型常数（没有小数部分），如123、-456等。有十进制、八进制和十六进制3种表示形式。

- ☑ 十进制：我们日常所见到的整数都是十进制形式的，如365、75等。十进制数使用0~9来表示所有数，递进关系为逢十进一。注意，除了0本身，其他十进制数不能以0开头。
- ☑ 八进制：使用0~7来表示所有的数，递进关系为逢八进一。八进制常量前要加上0作为前缀，如0123、0432等。
- ☑ 十六进制：使用0~9和字母A~F（或a~f）来表示所有的数，递进关系为逢十六进一。十六进制常量前要加上0x作为前缀，如0x123、0x3ba4。

整型常量可以是基本整型（int）、长整型（long int）或短整型（short int），还可以是无符号整型（unsigned）或有符号（signed）整型。在常量后加上L（或l），表示该常量是长整型，如1314L；加上U（或u），表示该常量为无符号整型，如520U。

注意

整型常量所占的内存大小和数值范围取决于编译器。字长为16位的计算机中，基本整型为16位；字长为32位的计算机中，基本整型为32位。这里，8位等于1字节。

无符号整型可以存放的正数范围比有符号整型大一倍，因为有符号整型的最高位用于储存符号，而无符号整型所有位全部同来存储数

字。

- ☑ 短整型是16位的，无符号短整型的取值范围是0~65535，有符号短整型的取值范围是-32768~+32767。
- ☑ 长整型是32位的，无符号长整型的取值范围是0~4294967295，有符号长整型的取值范围是-2147483648~+2147483647。
- ☑ 16位计算机中，基本整型的取值范围和短整型相同；32位及以上计算机中，基本整型的取值范围和长整型相同。

技巧

Windows 10系统中，通过内置的计算器可进行八进制、十进制和十六进制之间的转换。打开计算器后，默认显示的是标准模式计算器，只能进行十进制运算。单击左上角的 ☰ 按钮，在菜单中选择“程序员”命令，可切换至程序员计算器模式，如图3.2所示。

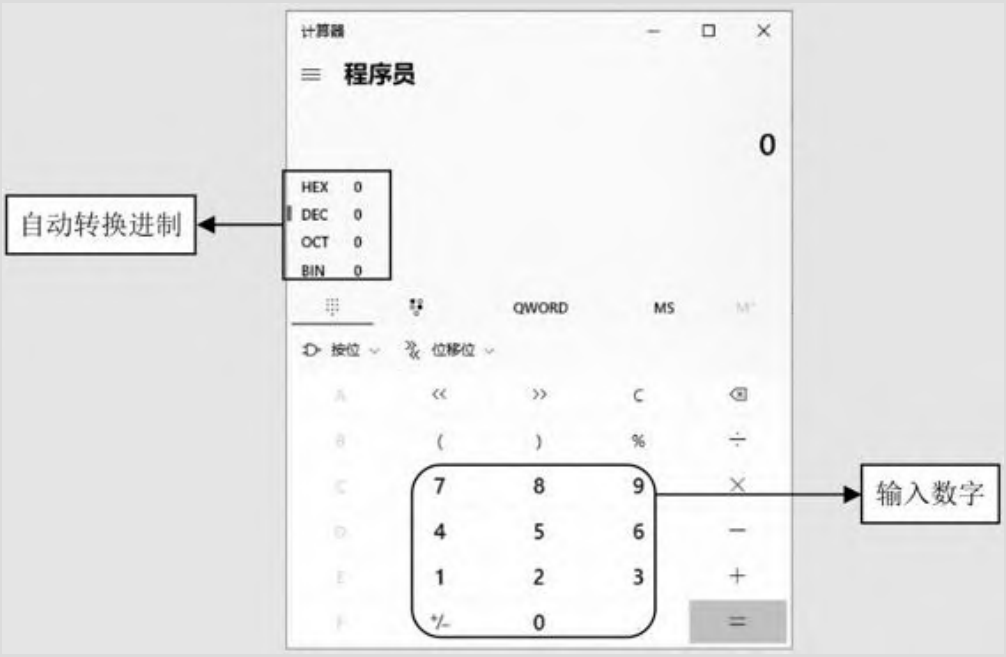


图3.2　程序员计算器

3.5.2　实型常量



实型也称为浮点型，由整数和小数两部分组成，并用十进制的小数点进行分隔。表示实数的方式有小数和指数两种。

1．小数形式

小数表示方法就是使用十进制的小数方法描述实型。例如：

```
123.45;                          /*小数表示方法*/
0.5458;
```

2．指数形式（科学计数形式）

当实型数非常大或非常小时，可使用指数形式来表示。使用字母e或者E表示指数形式，如45e2表示的是45×10^{2}，即4500，45e-2表示的是45×10^{-2}，即0.45。上面示例的两个小数用指数形式表示如下：

```
1.2345e2;                        /*指数表示方法*/
5.458e-1;
```

注意

实型常量中可以使用下画线进行分隔，这些下画线不会影响数字的值，仅提供分隔作用，以方便阅读。需要注意的是，下画线不能位

于数字开头和结尾，不可以写在二进制和十六进制前缀字母旁，也不可以写在小数点旁。

书写实型常量时，可以在常量后加上F或L后缀（大小写通用）。F表示该常量是float单精度类型，L表示该常量为long double长双精度类型。例如：

```
1.2345e2F;                    /*单精度类型*/
5.458e-1L;                    /*长双精度类型*/
```

如果不加后缀，则默认实型常量为double双精度类型。例如：

```
1.2345e2;                     /*默认为双精度类型*/
```

3.5.3 字符型常量



字符型常量与整型、实型有所不同，需要使用定界符进行限制。字符型常量包括两种：字符常量和字符串常量。

1．字符常量

使用一对单直撇引号（' '）括起来的字符就是字符常量，如'A'、'#'、'b'等。

- ☑ 字符常量只能包括一个字符，不能是字符串。例如，'A'是正确的，但用'AB'来表示字符常量就是错误的。
- ☑ 字符常量是区分大小写的。例如，'A'和'a'代表的是不同的字符常量。
- ☑ ' '代表的是定界符，不属于字符常量的一部分。

【例3.1】输出“Find Day!”（实例位置：资源包\TM\sl\03\01）

在本实例中，使用putchar函数将多个字符常量依次输出，在控制台中显示“Find Day!”。

```
#include <stdio.h>
int main()
{
    putchar('F');                    /*输出字符常量 F*/
    putchar('i');                    /*输出字符常量 i*/
    putchar('n');                    /*输出字符常量 n*/
    putchar('d');                    /*输出字符常量 d*/
    putchar('\0');                   /*字符结束标志*/
    putchar('D');                    /*输出字符常量 D*/
    putchar('a');                    /*输出字符常量 a*/
    putchar('y');                    /*输出字符常量 y*/
    putchar('!');                    /*输出字符常量!*/
    putchar('\n');                   /*进行换行*/
    return 0;
}
```

运行程序，显示效果如图3.3所示。

注意

给char型变量赋值时不可以使用3个单引号。这样写编译器会不知道从哪里开始，到哪里结束，进而产生错误。例如，下面的赋值语句会提示“error C2001:newline in constant”错误。

```
char cChar='A";                  /*使用 3 个单引号为字符型赋值*/
```

2．字符串常量

字符串常量是用一对双直撇引号（" "）括起来的若干字符序列。例如，"Have a good day! "和"beautiful day"即为字符串常量。如果字符串中一个字符都没有，将其称作空串，此时字符串的长度为0。

在C语言中存储字符串常量时，系统会在字符串的末尾自动添加一个“\0”，作为字符串的结束标志。例如，字符串"welcome"在内存中的存储形式如图3.4所示。

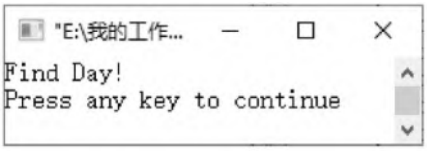


图3.3 输出“Find Day!”

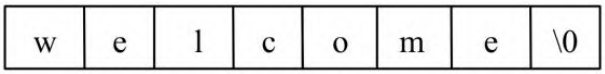


图3.4 “\0”为系统所加

【例3.2】打印《静夜思》诗句（实例位置：资源包\TM\sl\03\02）

在本实例中，使用printf函数将字符串常量《静夜思》诗句在控制台输出显示。

```
#include <stdio.h>                    /*包含头文件*/
int main()
{
    printf("窗前明月光\n");           /*输出字符串*/
    printf("疑是地上霜\n");
    printf("举头望明月\n");
    printf("低头思故乡\n");
    return 0;                         /*程序结束*/
}
```

运行程序，显示效果如图3.5所示。

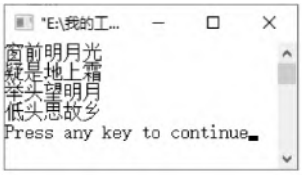


图3.5 输出字符串

字符常量和字符串常量的差别主要体现在以下几个方面。

☑ 定界符不同。字符常量使用的是' '，而字符串常量使用的是" "。

☑ 长度不同。字符常量只能有一个字符，长度为1。字符串常量的长度可以是0，也可以是任意值。即使字符串常量中的字符只有1个，长度也不是1，而是2。例如，字符串常量"F"，其在内存中的存储情况如图3.6所示（因为系统自动添加了结束字符“\0”，所以其长度为2）。

☑ 存储方式不同。字符常量中存储的是字符的ASCII码值；字符串常量不仅要存储有效的字符，还要存储结尾处的结束标志“\0”。

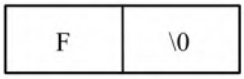


图3.6 字符串“F”

ASCII码到底是什么呢？在C语言中，所使用的字符被一一映射到一个表中，这个表称为ASCII码表，如表3.2所示。

表3.2 ASCII码表

ASCII值	缩写/字符	解　释	ASCII值	缩写/字符	解　释
0	NUL（null）	空字符（\0）	17	DC1（device control1）	设备控制1
1	SOH（star to fhanding）	标题开始	18	DC2（device control2）	设备控制2
2	STX（star to ftext）	正文开始	19	DC3（device control3）	设备控制3
3	ETX（end of text）	正文结束	20	DC4（device control4）	设备控制4
4	EOT（end of transmission）	传输结束	21	NAK（negative acknowledge）	拒绝接收
5	ENQ（enquiry）	请求	22	SYN（synchronousidle）	同步空闲
6	ACK（acknowledge）	收到通知	23	ETB（end of trans.block）	传输块结束
7	BEL（bell）	响铃（\a）	24	CAN（cancel）	取消
8	BS（backspace）	退格（\b）	25	EM（end of medium）	介质中断
9	HT（horizontal tab）	水平制表符（\t）	26	SUB（substitute）	替补
10	LF（NL）（linefeed,newline）	换行键（\n）	27	ESC（escape）	溢出
11	VT（verticaltab）	垂直制表符	28	FS（file separator）	文件分割符
12	FF（NP）（formfeed,newpage）	换页键（\f）	29	GS（group separator）	分组符
13	CR（carriagereturn）	回车键（\r）	30	RS（record separator）	记录分离符
14	SO（shift out）	不用切换	31	US（unit separator）	单元分隔符
15	SI（shift in）	启用切换	32	……	（完整表参见附录A）
16	DLE（data link escape）	数据链路转义			

3.5.4 转义字符



在例3.1和例3.2中能看到“\n”符号，但输出结果中却不显示该符号，只是进行了换行操作，这种符号称为转义符号。

转义符号在字符常量中是一种特殊的字符，通常以反斜杠“\”开头，后面跟一个或几个字符。常用的转义字符及其含义如表3.3所示。

表3.3　常用的转义字符表

转义字符	意　义	转义字符	意　义
\n	回车换行	\\	反斜杠“\”
\t	横向跳到下一制表位置	\'	单引号符
\v	竖向跳格	\a	鸣铃
\b	退格	\ddd	1～3位八进制数所代表的字符
\r	回车	\xhh	1～2位十六进制数所代表的字符
\f	走纸换页		

3.5.5　符号常量



在例1.2中，程序的功能是求解一个长方体的体积。因为题设中长方体的高度是固定的，所以使用符号名Height来代替固定的常量值10，这里使用的符号名就称之为符号常量。使用符号常量的好处在于可以为编程和阅读带来方便。

【例3.3】小时和年之间的转换（实例位置：资源包\TM\sl\03\03）

本实例使用符号常量来表示一年有多少个小时（一年按365天计算）。在控制台上显示文字，提示用户输入一个表示小时数量的值。用户输入小时数后，经过计算得到对应的年数，并输出显示。

```
#include <stdio.h>
#define HOUR 365*24                    /*定义符号常量 HOUR*/
int main()
{
    int h,result;                      /*定义变量*/
    printf("请输入小时：");             /*提示输入信息*/
    scanf("%d",&h);                    /*输入数据*/
    result=h/(HOUR);                   /*进行计算*/
    printf("一共有%d 年\n",result);     /*显示结果*/
    return 0;                          /*程序结束*/
}
```

运行程序，显示效果如图3.7所示。

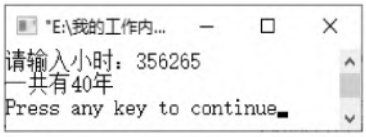


图3.7　符号常量的使用

编程训练（答案位置：资源包\TM\sl\03\编程训练\）

训练1：输出猪猪表情　使用字符型常量'（'、'）'、'-'、'o'组合成猪猪表情，用putchar函数输出。输出结果如下：

```
(-(○○)-)
```

训练2：打印绕口令《白石塔》　使用字符串常量，用printf函数输出绕口令《白石塔》内容，输出结果如下：

```
白石塔，白石搭，
白石搭白塔，白塔白石搭，
搭好白石塔，白塔白又大。
```

3.6　变量

在前面的学习中，我们已经多次接触过变量。变量就是在程序运行期间其值可以变化的量，每个变量都属于某种类型，如整型、实型、字符型

等，该类型定义了变量的格式和行为。一个变量应该有属于自己的名称，并且在内存中占有一定的存储空间，其所占空间大小取决于类型。

3.6.1 整型变量



整型变量是用来存储整型数值的变量。整型变量可以分为如表3.4所示的6种类型，其中基本类型的符号使用int关键字，在此基础上可以根据需要加上一些符号进行修饰，如关键字short或long。

表3.4 整型变量的分类

类 型	关 键 字	类 型	关 键 字
有符号基本整型	[signed] int	无符号短整型	unsigned short [int]
无符号基本整型	unsigned [int]	有符号长整型	[signed] long [int]
有符号短整型	[signed] short [int]	无符号长整型	unsigned long [int]

说明

表格中的[]为可选部分。例如[signed] int，在书写时可以省略signed关键字。

1．有符号基本整型

有符号基本整型是指signed int型，其值是基本的整型常数。编写时，常将其关键字signed进行省略。有符号基本整型在内存中占4个字节，取值范围是-2147483648~2147483647。

说明

通常说到的整型，都是指有符号基本整型int。

定义一个有符号整型变量的方法是在变量前使用关键字int。例如，定义一个整型变量iNumber，并为其赋值-10的方法如下：

```
int iNumber;                                    /*定义有符号基本整型变量*/
iNumber=-10;                                    /*为变量赋值*/
```

或者在定义变量的同时为其赋值：

```
int iNumber=-10;                                /*定义有符号基本整型变量的同时进行赋值*/
```

注意

程序中用到的变量应集中定义，并放在变量赋值之前，否则会产生错误。例如：

```
/*正确的写法：*/
int iNumber1;                                    /*先定义变量*/
int iNumber2;
iNumber1=6;                                      /*再对变量赋值*/
iNumber2=7;
/*错误的写法：*/
int iNumber1;                                   /*定义变量*/
iNumber1=6;                                     /*为变量赋值，错误！！因为赋值语句在定义变量语句之
int iNumber2;                                   /*定义变量*/
iNumber2=7;                                     /*为变量赋值*/
```

【例3.4】输出有符号int类型取值范围（**实例位置：资源包\TM\sl\03\04**）

本实例练习对有符号基本整型变量的使用，读者可直观地看到其作用。

```
#include <stdio.h>
int main()
{
    signed int i,j;                          /*定义有符号基本整型变量*/
    i=-2147483648;                           /*为变量赋值*/
    j=2147483647;                            /*为变量赋值*/
    printf("有符号整型变量最小值是%d\n",i);    /*显示整型变量*/
    printf("有符号整型变量最大值是%d\n",j);    /*显示整型变量*/
     return 0;                               /*程序结束*/
}
```

运行程序，显示效果如图3.8所示。

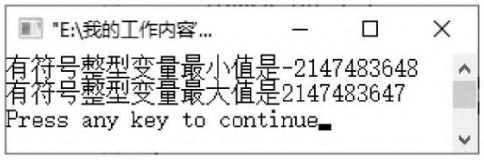


图3.8　有符号基本整型

2．无符号基本整型

无符号基本整型使用的关键字是unsigned int，其中的关键字int在编写时可以省略。无符号基本整型在内存中占4个字节，取值范围是0~4294967295。

定义一个无符号基本整型变量的方法是在变量前使用关键字unsigned。例如，要定义一个无符号基本整型变量iUnsignedNum，并为其赋值111的方法如下：

```
unsigned iUnsignedNum;                   /*定义无符号基本整型变量*/
iUnsignedNum=111;                        /*为变量赋值*/
```

3．有符号短整型

有符号短整型使用的关键字是signed short int，其中的关键字signed和int在编写时可以省略。有符号短整型在内存中占两个字节，取值范围是-32768~32767。

定义一个有符号短整型变量的方法是在变量前使用关键字short。例如，要定义一个有符号短整型变量iShortNum，并为其赋值-521的方法如下：

```
short iShortNum;                         /*定义有符号短整型变量*/
iShortNum=-521;                          /*为变量赋值*/
```

4．无符号短整型

无符号短整型使用的关键字是unsigned short int，其中的关键字int在编写时可以省略。无符号短整型在内存中占两个字节，取值范围是0~65535。

定义一个无符号短整型变量的方法是在变量前使用关键字unsigned short。例如，要定义一个无符号短整型变量iUnsignedShtNum，并为其赋值222的方法如下：

```
unsigned short iUnsignedShtNum;          /*定义无符号短整型变量*/
iUnsignedShtNum=222;                     /*为变量赋值*/
```

5．有符号长整型

有符号长整型使用的关键字是signed long int，其中的关键字signed和int在编写时可以省略。有符号长整型在内存中占4个字节，取值范围是-2147483648~2147483647。

定义一个有符号长整型变量的方法是在变量前使用关键字long。例如，要定义一个有符号长整型变量iLongNum，并为其赋值-3333333的方法如下：

```
long iLongNum;                           /*定义有符号长整型变量*/
iLongNum=-3333333L;                      /*为变量赋值*/
```

6．无符号长整型

无符号长整型使用的关键字是unsigned long int，其中的关键字int在编写时可以省略。无符号长整型在内存中占4个字节，取值范围是0~4294967295。

定义一个无符号长整型变量的方法是在变量前使用关键字unsigned long。例如，要定义一个无符号长整型变量iUnsignedLongNum，并为其赋值6666666的方法如下：

```
unsigned long iUnsignedLongNum;                /*定义无符号长整型变量*/
iUnsignedLongNum=6666666L;                     /*为变量赋值*/
```

3.6.2 实型变量



实型变量也称为浮点型变量，是用来存储实型数值的变量，其中实型数值由整数和小数两部分组成。实型变量根据精度可以分为单精度类型、双精度类型和长双精度类型3种。

1．单精度类型

单精度类型使用的关键字是float，它在内存中占4个字节，取值范围是-3.4×10^{-38}~3.4×10^{38}。

定义一个单精度类型变量的方法是在变量前使用关键字float。例如，要定义一个变量fFloatStyle，并为其赋值3.14的方法如下：

```
float fFloatStyle;                             /*定义单精度类型变量*/
fFloatStyle=3.14f;                             /*为变量赋值*/
```

【例3.5】存款一年后能取出多少钱（实例位置：资源包\TM\sl\03\05）

已知银行的年利率是2.95%，如果在银行中存入1000元，一年后可以取出多少钱？（提示：总额＝本金＋年利率*本金）。具体代码如下：

```
#include<stdio.h>
int main()
{
    float result;                              /*定义单精度类型变量*/
    result=1000+0.0295*1000;                   /*计算一年后可取出的本金+利息总额*/
    printf("result=%f\n",result);              /*显示结果*/
    return 0;                                  /*程序结束*/
}
```

运行程序，显示效果如图3.9所示。

2．双精度类型

双精度类型使用的关键字是double，它在内存中占8个字节，取值范围是-1.7×10^{-308}~1.7×10^{308}。

定义一个双精度类型变量的方法是在变量前使用关键字double。例如，要定义一个变量dDoubleStyle，并为其赋值5.321的方法如下：

```
double dDoubleStyle;                    /*定义双精度类型变量*/
dDoubleStyle=5.321;                     /*为变量赋值*/
```

【例3.6】计算圆柱体的粮仓体积和屯粮量（实例位置：资源包\TM\sl\03\06）

一个圆柱体的粮仓，底面直径为50米，高为15米，该粮仓体积为多少立方米？如果每立方米能屯粮896千克，该粮仓一共可存储多少千克粮食？（提示：体积=3.14*半径^2*高）。具体代码如下：

```
#include<stdio.h>
int main()
{
    double d,h,v,res;                   /*定义 4 个双精度类型变量*/
    d=50;                               /*对变量赋值，d 表示直径，h 表示高，v 表示体积*/
    h=15;
    v=3.14*(d/2)*(d/2)*h;               /*根据体积公式计算粮仓体积*/
    printf("该粮仓体积为%f 立方米\n",v);   /*显示结果*/
    res=896*v;                          /*计算屯粮量*/
    printf("该粮仓一共可储存%f 千克粮食\n",res); /*显示结果*/
    return 0;                           /*程序结束*/
}
```

运行程序，显示效果如图3.10所示。

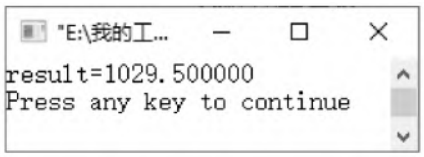


图3.9　使用单精度类型变量



图3.10　使用双精度类型变量

3．长双精度类型

长双精度类型使用的关键字是long double，它在内存中占8个字节，取值范围是-1.7×10^{-308}~1.7×10^{308}。

定义一个双精度类型变量的方法是在变量前使用关键字long double。例如，要定义一个变量fLongDouble，并为其赋值46.257的方法如下：

```
long double fLongDouble;                    /*定义长双精度类型变量*/
fLongDouble=46.257;                         /*为变量赋值*/
```

【例3.7】记录太阳和火星之间的距离（实例位置：资源包\TM\sl\03\07）

在本实例中，定义一个长双精度类型变量表示太阳和火星之间的距离，然后为其赋值22.794，最后通过输出语句将其显示在控制台。

```
#include<stdio.h>
int main()
{
    long double fLongDouble;                        /*定义长双精度变量*/
    fLongDouble=22.794;                             /*为变量赋值*/
    printf("火星距太阳有%f 亿千米\n",fLongDouble);    /*将变量值进行输出*/
    return 0;                                       /*程序结束*/
}
```

运行程序，显示效果如图3.11所示。

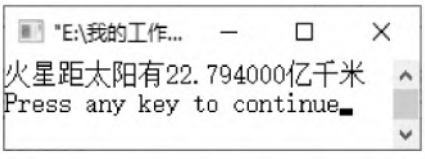


图3.11　使用长双精度类型变量

3.6.3　字符型变量



字符型变量是用来存储字符常量的变量。将一个字符常量存储到一个字符变量中，实际上是将该字符的ASCII码值（无符号整数）存储到内存单元中。

字符型变量在内存空间中占一个字节，取值范围是-128~127。

定义一个字符型变量的方法是使用关键字char。例如，要定义一个字符型变量cChar，为其赋值'a'的方法如下：

```
char cChar;                    /*定义字符型变量*/
cChar= 'a';                    /*为变量赋值*/
```

说明

字符型变量在内存中存储的是字符的ASCII码，即一个无符号整数，其形式与整数的存储形式一样，因此C语言允许字符型数据与整型数据之间互相转换。

【例3.8】字符型和整型间的互换（实例位置：资源包\TM\sl\03\08）

本实例为定义的字符型变量和整型变量进行不同的赋值，然后通过输出结果观察整型变量和字符型变量之间的转换。

```
#include<stdio.h>
int main()
{
    char cChar1;                          /*定义字符型变量 cChar1*/
    char cChar2;                          /*定义字符型变量 cChar2*/
    int iInt1;                            /*定义整型变量 iInt1*/
    int iInt2;                            /*定义整型变量 iInt2*/

    cChar1='a';                           /*为变量赋值*/
    cChar2=97;
    iInt1='a';
    iInt2=97;

    printf("%c\n",cChar1);          /*显示结果为 a*/
    printf("%d\n",cChar2);          /*显示结果为 97*/
    printf("%c\n",iInt1);           /*显示结果为 a*/
    printf("%d\n",iInt2);           /*显示结果为 97*/
    return 0;                       /*程序结束*/
}
```

上述代码首先定义了4个变量，两个字符型，两个整型，然后分别赋值为'a'和97。因为字符'a'在ASCII码中的值正好是97，因此两者可以互相转换输出。运行程序，显示效果如图3.12所示。

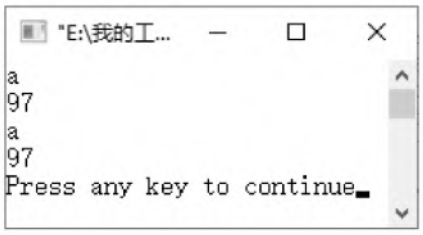


图3.12　使用字符型变量

下面对目前学过的所有数据类型进行一下概括总结，如表3.5所示。

表3.5　数值型和字符型数据的字节数和数值范围

类　型	关 键 字	字　节	数 值 范 围
整型	[signed] int	4	-2147483648~2147483647
无符号整型	unsigned [int]	4	0~4294967295
短整型	short [int]	2	-32768~32767
无符号短整型	unsigned short [int]	2	0~65535
长整型	long [int]	4	-2147483648~2147483647
无符号长整型	unsigned long [int]	4	0~4294967295
字符型	[signed] char	1	-128~127
无符号字符型	unsigned char	1	0~255
单精度型	float	4	-3.4×10^{-38}~3.4×10^{38}
双精度型	double	8	-1.7×10^{-308}~1.7×10^{308}
长双精度型	long double	8	-1.7×10^{-308}~1.7×10^{308}

编程训练（答案位置：资源包\TM\sl\03\编程训练\）

训练3：记录大毛的基本信息　使用整型变量保存大毛的年龄，使用字符型变量保存大毛的性别（W表示女，M表示男），用实型变量保存大毛的身高、体重，最后用printf函数输出。结果如下：

```
大毛基本信息如下：
年龄是：31岁,性别是：M
身高是：1.680000米,体重是：59.400000千克
```

训练4：预测快递到货时长　假设某商品从广州市发货，预测其到达北京、上海、浙江、江苏、山东、湖南、辽宁、吉林、黑龙江等地的时长，使用整型变量存储预测的时长（即天数），用printf函数输出预测结果。输出结果如下：

```
快递预测时间：
从广州到湖南省、浙江省是2天
从广州到上海、北京、江苏省、山东省是3天
从广州到辽宁省、吉林省、黑龙江省是4天
```

3.7　变量的存储类别

根据变量的产生时间，其在C程序中有两种不同的存储形式——静态存储和动态存储。静态存储是程序运行时为其分配固定的存储空间，动态存储则是在程序运行期间根据需要动态地分配存储空间。

可通过存储类修饰符来告诉编译器要处理什么样的类型变量，包括自动变量（auto）、静态变量（static）、寄存器变量（register）和外部变量（extern）4种。

3.7.1　auto变量

auto关键字用于定义一个局部变量为自动变量，该变量存储在内存的动态存储区中，这意味着每次执行到该变量定义语句时，都会产生一个新的变量，并且重新对其初始化。

【例3.9】 auto变量两次加1结果（**实例位置：资源包\TM\sl\03\09**）

在AddOne函数中定义一个auto型的整型变量iInt，在其中对变量执行加1操作。之后在主函数main中两次调用AddOne函数。从结果可以看到，在AddOne函数中定义整型变量时，系统会为其分配内存空间，在函数调用结束时，自动释放这些存储空间。

```
#include<stdio.h>

void AddOne()
{
    auto int iInt=1;                    /*定义 auto 整型变量*/
    iInt=iInt+1;                        /*变量加 1*/
    printf("%d\n",iInt);                /*显示结果*/
}
int main()
{
    printf("第一次调用：");              /*显示提示信息*/
    AddOne();                           /*调用 AddOne 函数*/
    printf("第二次调用：");              /*显示提示信息*/
    AddOne();                           /*调用 AddOne 函数*/
    return 0;                           /*程序结束*/
}
```

运行程序，显示效果如图3.13所示。

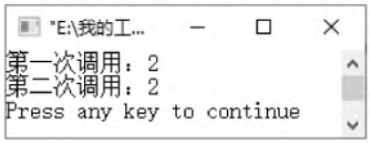


图3.13　使用auto变量

注意

关键字auto可以省略，不特别指定情况下，局部变量的存储方式默认为都是auto型。

3.7.2 static变量



static变量表示静态变量，存储在内存的静态存储区中。

对于局部变量来说，尽管static变量和auto变量的作用域都仅限于声明变量的函数之中，但是在语句块执行期间，static变量将始终保持它的值，并且初始化操作只在第一次执行时起作用，在随后的运行过程中变量将保持语句块上一次执行时的值。

【例3.10】使用static变量记录点击量（实例位置：资源包\TM\sl\03\10）

创建click函数，用于记录用户的点击量。在函数体中定义static变量sum=0，每次调用此方法，sum的值都会加1，并输出此时sum的值。调用5次click函数，查看此时的点击量是多少。代码如下:

```
#include <stdio.h>
void click()
{
    static int sum = 0;                    /*定义 static 整型变量*/
    sum = sum +1;                          /*变量加 1*/
    printf("此时点击量：%d\n",sum);          /*显示点击量*/
}

int main()
{
    click();                               /*5 次调用 click 函数*/
    click();
    click();
    click();
    click();
    return 0;
}
```

运行程序，显示效果如图3.14所示。

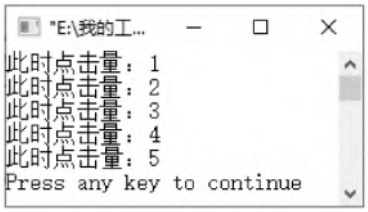


图3.14　使用static变量

3.7.3　register变量



register变量又称为寄存器存储类变量。通过它，程序员可以请求编译器把某个需要频繁访问的局部变量存放在计算机的硬件寄存器中，而不是内存中。这样做的好处是可以提高程序的运行速度。不过这只是反映了程序员的主观意愿，由于cpu的寄存器数量有限，实际上编译器可以忽略register对变量的修饰。

用户无法获得寄存器变量的地址，因为绝大多数计算机的硬件寄存器都不占用内存地址。而且，即使编译器忽略register，而把变量存放在可设定的内存中，也是无法获取变量的地址的。

要想有效地利用寄存器register关键字，必须像汇编语言程序员那样了解处理器的内部结构，知道可用于存放变量的寄存器的数量、种类以及工作方式。由于不同计算机对于这些细节可能是不同的，因此对于一个具备可移植性的程序来说，register的作用并不大，读者了解即可。

【例3.11】加快计算机运行速度（实例位置：资源包\TM\sl\03\11）

使用register变量修饰整型变量。代码如下：

```
#include<stdio.h>
int main()
{
    register int iInt;                  /*定义寄存器整型变量*/
    iInt = 100;
    printf("%d\n",iInt);                /*显示结果*/
    return 0;                           /*程序结束*/
}
```

运行程序，显示效果如图3.15所示。

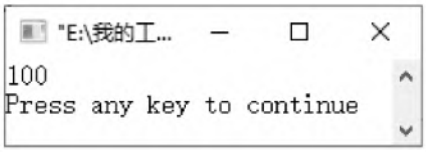


图3.15　使用register变量

3.7.4　extern变量



一个工程（Project，即项目）是由多个C文件组成的，这些源代码文件会分别进行编译，然后链接成一个可执行模块。有时我们需要用到其他

C文件中的变量，这些变量就需要声明为extern型。

extern变量又称为外部存储变量，常用于声明程序需要使用且已在其他程序文件中定义过的变量。

【例3.12】打破常规输出100（实例位置：资源包\TM\sl\03\12）

在本实例中，首先在Extern1文件中声明一个外部整型变量iExtern，然后在Extern2文件中定义iExtern变量，并进行赋值，最后在extern1中将变量值显示到控制台。

```
/*////////////////////////////////////////////////////////////////////////////////*/
/*                                在 Extern1 文件中                               */
/*////////////////////////////////////////////////////////////////////////////////*/
#include<stdio.h>

int main()
{
    extern int iExtern;                 /*声明外部整型变量*/
    printf("%d\n",iExtern);             /*显示变量值*/
    return 0;                           /*程序结束*/
}

/*////////////////////////////////////////////////////////////////////////////////*/
/*                                在 Extern2 文件中                               */
/*////////////////////////////////////////////////////////////////////////////////*/

#include<stdio.h>

int iExtern=100;                        /*定义一个整型变量，为其赋值 100*/
```

运行程序，显示效果如图3.16所示。

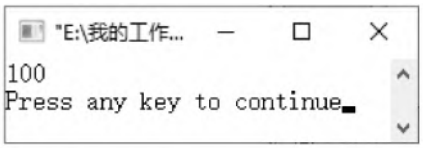


图3.16 使用extern变量

编程训练（答案位置：资源包\TM\sl\03\编程训练\）

训练5：还剩多少停车位 记录停车场的车位数量，每进一辆车，车位数就会减少一个。模拟此场景，创建函数park用于记录停车场的车位数量，函数中定义static变量sum=3625，每次调用此方法，sum的值都会减1，并输出此时sum的值。调用5次park函数，查看此时车位数量是多少。输出结果如下：

```
进入一辆车，此时车位剩：3624个
进入一辆车，此时车位剩：3623个
进入一辆车，此时车位剩：3622个
进入一辆车，此时车位剩：3621个
进入一辆车，此时车位剩：3620个
```

训练6：数据清洗　定义一个mul函数，在函数中使用auto修饰变量a，并对此变量执行乘以2操作。主函数中调用5次mul函数，查看结果，发现每次都得到同一个结果，说明每次都将数据进行了清洗和重置。结果如下：

```
第1次调用：6
第2次调用：6
第3次调用：6
第4次调用：6
第5次调用：6
```

3.8　混合运算



不同类型的数据之间可以进行混合运算，如10+'a'-1.5+3.2*6。在进行这样的运算时，不同的数据类型要先转换成同一类型，然后再进行运算。类型之间的转换规律如图3.17所示。

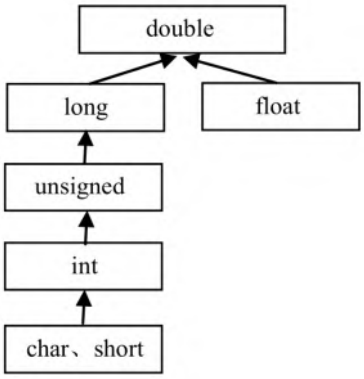


图3.17　不同类型之间的转换规律

【例3.13】不同数据类型间的混合运算（实例位置：资源包\TM\sl\03\13）

在本实例中，计算（10+'a'）-1.5+3.2*6的结果，最后使用printf函数将其输出。

```
#include<stdio.h>

int main()
{
    int iInt1=10;                                   /*定义整型变量*/
    int iInt2=6;
    char cChar='a';                                 /*定义字符变量，a 的 ASCII 码值为 97*/
    float fFloat1=1.5f;                             /*定义单精度型变量*/
    float fFloat2=3.2f;
    double result=(iInt1+cChar)-fFloat1+fFloat2*iInt2 ;   /*得到混合运算的结果*/
    printf("%f\n",result);                          /*输出变量值*/
    return 0;                                       /*程序结束*/
}
```

运行程序，显示效果如图3.18所示。

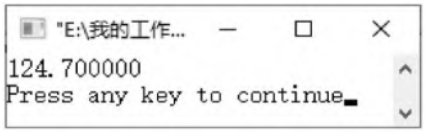


图3.18　混合运算

编程训练（答案位置：资源包\TM\sl\03\编程训练\）

训练7：混合运算1　编写程序，计算混合运算1.68+'s'+8-9*9+8+'d'的结果，最后使用printf函数将其输出。

训练8：混合运算2　编写程序，计算混合运算（1.68+'s'+8-9）*（9+8+'d'）的结果，最后使用printf函数将其输出。

3.9　实践与练习

（答案位置：资源包\TM\sl\03\实践与练习\）

综合练习1：设置百度地图常用地点　使用百度地图时，经常会弹出设置常用地点的对话框。编写程序，输出家庭住址和公司地址。输出结果如下：

```
==============设置常用地点==============
我的家：  北京市海淀区上地十街 9 号
我的单位：北京市朝阳区定福街 1220 号
```

综合练习2：模拟联通流量提醒　定义两个实型变量，表示已用流量（3.592）和剩余流量（3.408）。定义一个字符型变量，表示流量查询网址（http://u.10010.cn/tAE3v）。输出结果如下：

```
中国联通流量提醒：
截至10月21日24时，
您当月共享国内通用流量已用3.592GB，剩余3.408GB；
其他流量使用情况请点击进入http://u.10010.cn/tAE3v
```

综合练习3：计算3的幂　在自定义函数中使用static静态局部整型变量，计算3的幂。输出结果如下：

```
结果为：9
结果为：27
结果为：81
结果为：243
结果为：729
结果为：2187
```

综合练习4：抓鸭子　模拟抖音上非常火的抓鸭子游戏。游戏开始后，大家轮番说话，第一个人说“抓鸭子”，第二个人问“抓几只”，第三个说“抓X只”（X代表鸭子数量），其后的人可以说“抓住了”或“没抓着”。如果说“没抓着”，后面的人就重复上述过程；如果说“抓住了”，后面的人就依次模仿鸭子叫，“嘎”的次数要对应鸭子的数量。如果有人出错，就算输，要接受惩罚。输出结果如下：

```
---4个人，抓鸭子游戏开始了----
A说：抓鸭子
B说：抓几只
C说：抓2只
D说：抓住了
A说：嘎
B说：嘎
C说：抓鸭子
D说：抓几只
A说：抓一百只
B说：嘎
C说：嘎
GAGAGA......
```

综合练习5：一起来测试情商　编写程序，测试你的情商是多少。运行效果如图3.19所示。

综合练习6：输出饭店菜谱　编写程序，输出如图3.20所示的菜谱。

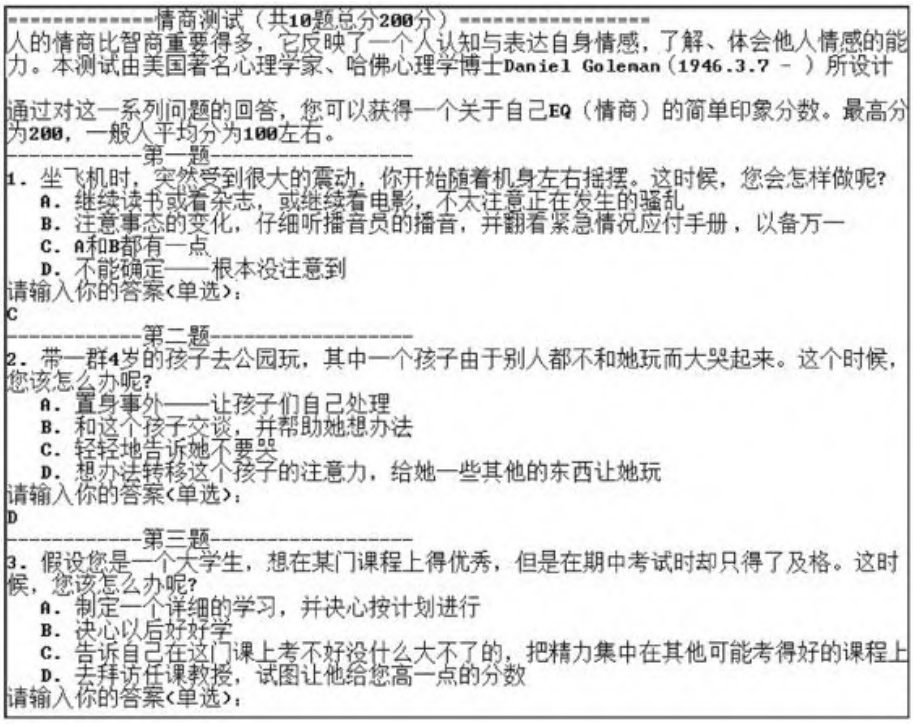


图3.19　测试情商题目

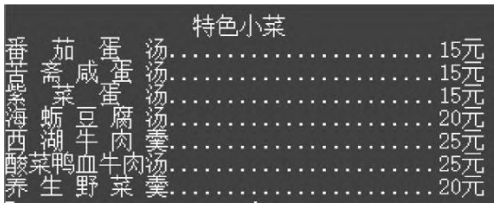


图3.20　输出菜谱

综合练习7：输出狙击枪形状　**利用字符串常量输出狙击枪形状，输出形状如下所示。（提示：利用搜狗特殊符号）**



综合练习8：模拟超市结账　**超市结账是我们经常会遇到的事情，假设某人去超市买商品的结果分别如下：56.75元、72.91元、88.50元、26.37元、68.51元。接下来模拟超市结账场景。运行结果如下所示。（提示：利用搜狗特殊符号画★ ☆）。**



第4章
运算符与表达式

了解了程序中常用的数据类型后，还应该懂得如何操作这些数据。因此，掌握C语言中各种运算符与表达式是必不可少的。

本章致力于使读者了解表达式的概念，掌握运算符及相关表达式的使用方法，其中包括赋值运算符、算术运算符、关系运算符、逻辑运算符、位逻辑运算符、逗号运算符和复合赋值运算符，并且通过实例进行相应的练习，加深印象。

本章的知识架构及重难点如下：



4.1 表达式



C语言中，表达式由操作符和操作数组成。简单的表达式只包含一个操作符，复杂的表达式包含两个或两个以上操作符。例如：

```
25+25
num+29
iBase+(iPay*iDay)
```

表达式可出现在赋值语句的右侧或函数的参数中。表达式可返回一个结果，其数据类型取决于组成表达式的变量和常量的类型。

说明

表达式的返回值具有逻辑特性：如果返回值非零，那么返回的是真值，否则返回的是假值。因此，可以将表达式放在流程控制语句中，形成条件表达式。

【例4.1】编程中的数学计算（**实例位置：资源包\TM\sl\04\01**）

本实例定义了3个整型变量，首先对两个变量赋值并输出，然后通过3次计算，将3个表达式的结果依次赋值给第3个变量，最后将变量的值显示在屏幕上。

```c
#include<stdio.h>
int main()
{
    int iNumber1,iNumber2,iNumber3;                    /*定义 3 个变量*/
    iNumber1=3;                                        /*为变量赋初值*/
    iNumber2=7;

    printf("the first number is :%d\n",iNumber1);      /*显示变量值*/
    printf("the second number is :%d\n",iNumber2);

    iNumber3=iNumber1+10;                              /*计算 iNumber1+10 并将结果赋给 iNumber3*/
    printf("the first number add 10 is :%d\n",iNumber3);  /*显示 iNumber3 的值*/

    iNumber3=iNumber2+10;                              /*计算 iNumber2+10 并将结果赋给 iNumber3*/
    printf("the second number add 10 is :%d\n",iNumber3); /*显示 iNumber3 的值*/

    iNumber3=iNumber1+iNumber2;                        /*计算 iNumber1+iNumber2 并赋给 iNumber3*
    printf("the result number of first add second is :%d\n",iNumber3);   /*将计算结果输出*/

    return 0;                                          /*程序结束*/
}
```

（1）主函数main中首先定义3个变量，然后为变量赋值。“iNumber1=3;”表示将3赋值给iNumber1，“iNumber2=7;”表示将7赋值给iNumber2，然后通过输出语句printf显示这两个变量的值。

（2）语句“iNumber3=iNumber1+10;”中，表达式将变量iNumber1与常量10相加，然后将返回的值赋给iNumber3变量，之后使用输出函数printf将iNumber3变量的值进行显示。接下来将变量iNumber2与常量10相加，进行相同的操作。

（3）语句“iNumber3=iNumber1+iNumber2;”中，iNumber1和iNumber2两个变量相加，将返回的和赋给变量iNumber3，最后输出显示结果。

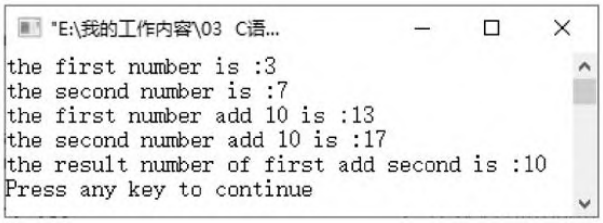


图4.1 程序输出结果

运行程序，显示效果如图4.1所示。

编程训练（答案位置：资源包\TM\sl\04\编程训练\）

训练1：计算两个数的和　定义两个整型变量，分别赋值为520和1314，然后输出两数相加的结果。输出结果如下：

```
加数一是：520
加数二是：1314
相加的和是：1834
```

训练2：求三角形第三边的长度范围　三角形中，任何一边的长度都大于其他两边之差，小于其他两边之和。一个三角形的两边分别是13、16，求第三边的取值范围。输出结果如下：

```
三角形的两边长分别为：13,16
第三边的取值范围是：3<第三边<29
```

4.2　赋值运算符与赋值表达式

程序中常常遇到的赋值符号“=”就是赋值运算符，其作用就是将一个数据赋给一个变量。例如：

```
iAge=520;
```

这就是一次赋值操作，将常量520赋给变量iAge。也可以将一个表达式的值赋给一个变量，例如：

```
Total=Counter*3;
```

4.2.1　变量赋初值



在声明变量时，可以为其赋一个初值，就是将一个常数或者一个表达式的结果赋值给一个变量，变量中保存的内容就是这个常量或者赋值语句中表达式的值。这就是为变量赋初值。

1．将常数赋值给变量

先来看一下将常数赋值给变量的情况。一般形式如下：

```
类型 变量名 = 常数;
```

其中的变量名也称为变量的标识符。例如：

```
char cChar ='A';
int iFirst=100;
float fPlace=1450.78f;
```

2．通过赋值表达式为变量赋初值

通过赋值语句，可把一个表达式的结果值赋给一个变量。一般形式如下：

```
类型 变量名 = 表达式;
```

可以看到，与常数赋值的一般形式是相似的。例如：

```
int iAmount= 12+26;
float fPrice= fBase+Day*3;
```

在上面的代码中，得到赋值的变量iAmount和fPrice称为左值，因为它出现在赋值语句的左侧。产生值的表达式称为右值，因为它出现在表达式的右侧。

在定义变量的同时直接为其赋值的操作称为赋初值，也就是变量的初始化。先定义变量，再进行变量的赋值操作也是可以的。例如：

```
int iMonth;                          /*定义变量*/
iMonth= 212;                         /*为变量赋值*/
```

【例4.2】计算乘坐出租车的费用（实例位置：资源包\TM\sl\04\02）

出租车每千米路程需要3元，李女士要到10千米远的地方，后来发现走错了，之后又坐出租车返回3千米，计算她需要花多少钱才能到达目的地。代码如下：

```
#include<stdio.h>
int main()
{
    int price=3;                     /*定义变量，表示每公里路程的费用*/
    int journey1=10;                 /*定义变量，表示目的地的距离*/
    int journey2=3;                  /*定义变量，表示返回路程的距离*/
    int money;                       /*定义变量，表示最终的费用*/
    money=journey1*price+journey2*price;  /*计算费用，将右侧表达式的计算结果赋给变量 money*/
    printf("总共花了%d 元\n",money);  /*输出结果*/
    return 0;                        /*程序结束*/
}
```

（1）出租车费用的计算公式是“每千米路程的费用×路程”。因路线错误，需要返回3千米，因此需要定义4个变量来计算乘出租车的费用。price表示每千米路程费用，journey1表示距离目的地的路程，journey2表示因走错返回的路程，money表示乘出租车的总费用。

（2）每千米路程的费用是固定的（3元），根据公式计算总车费，并将表达式的结果保存在money变量中。

（3）最后通过输出函数，将计算结果在屏幕上显示。

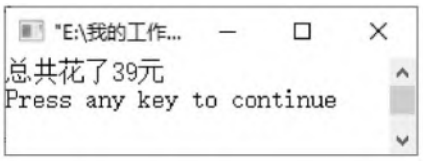


图4.2　为变量赋初值

运行程序，显示效果如图4.2所示。

4.2.2　自动类型转换

数值类型有多种，如字符型、整型、长整型、实型等，其变量、长度和符号特性都不同，取值范围也不同。3.8节中介绍了C语言中默认存在的一些自动类型转换规则，根据这些规则，不同的数值类型变量之间可以混合运算。除此之外，不同数据类型变量间相互赋值时也会进行自动类型转换。

例如，把较短的数值类型变量的值赋给较长的数值类型变量时，较短的数值类型变量其值会自动升级为较长的数值类型，数据信息不会丢失。但把较长的数值类型变量的值赋给较短的数值类型变量时，数据会降低级别显示。当数据大小超过较短的数值类型的可表示范围时，就会发生数据截断。

有些编译器遇到这种情况时会发出警告信息。例如，把一个实型变量赋值给一个整型变量时:

```
float i=10.1f;
int j=i;
```

此时编译器会发出警告，提示实型（float）转换为整型（int）时将发生数据截断，如图4.3所示。



图4.3　程序警告

误区警示

C语言中，整数的默认类型是int，浮点数的默认类型是double。如果一个表达式中数字都是int型，则表达式结果也默认为是int型。

例如，计算5除以2，如果这么写：

```
double b = 5 / 2;
```

计算后会发现b的值是2.0，而不是2.5。这是因为5和2都是int型，所以表达式5/2的结果默认也是int型，即2。赋值给b后，因为b是double型，所以得到的结果是2.0.

要想得到正确的结果2.5，应该这样修改：

```
double b = 5.0 / 2;                    /*将其中一个数改为实型*/
```

4.2.3 强制类型转换



通过自动类型转换的介绍得知，如果数据类型不同，系统会根据情况自动进行类型转换，但此时编译器会提示警告信息。如果使用强制类型转换告知编译器，就不会出现警告。

强制类型转换的一般形式如下：

```
(类型名) (表达式)
```

例如，在上述不同变量类型转换时使用强制类型转换：

```
float i=10.1f;
int j= (int)i;                         /*进行强制类型转换*/
```

在代码中可以看到，在变量前使用包含要转换类型的括号，就对变量进行了强制类型转换。

【例4.3】计算载货区摆放箱子的数量（实例位置：资源包\TM\sl\04\03）

一辆货车运输箱子，载货区宽2米，长4米，一个箱子宽1.5米，长1.5米，计算可以运输多少个箱子。注意，箱子数必须为整数，不存在半个箱子。实现代码如下：

```
#include<stdio.h>                                  /*包含头文件*/
int main()                                         /*主函数 main*/
{
    int width=2,length=4,num;                      /*定义变量，并赋初值，表示载货区规格*/
    float f=1.5f;                                  /*表示箱子规格*/
    num= (int)(width/f) * (int)(length/f);         /*通过强制类型转换，计算最大箱子承载数*/
    printf("载货区一层可以放%d 个箱子\n",num);      /*显示结果*/
    return 0;                                      /*程序结束*/
}
```

（1）程序中首先定义了3个整型变量，width表示载货区宽度，length表示载货区长度，num表示计算结果。然后定义了一个实型变量，f表示货物箱子的长、宽。最后用强制类型转换计算承载数。

（2）在强制类型转换代码中，用载货区的宽度除以箱子的宽度再强制转换成int类型，载货区的长度除以箱子的长度再强制转换成int类型，两者相乘，即可计算出最大能装载的箱子数量。

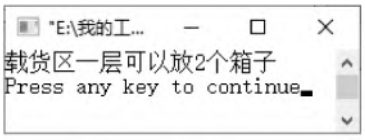


图4.4　强制类型转换的结果

（3）最后通过输出函数，将计算的结果在屏幕上显示。

运行程序，显示效果如图4.4所示。

误区警示

如果某个表达式要进行强制类型转换，需要将该表达式用括号括起来，否则将只对表达式中的第一个变量或常量进行强制类型转换。例如：

```
float x=2.5f,y=4.7f;          /*定义 2 个实型变量并赋初值*/
int z=(int)(x+y);             /*将表达式 x+y 的结果强制转换为整型*/
int g=(int)x+y;               /*将 x 强制转换为整型，再与实数 y 相加*/
```

编程训练（答案位置：资源包\TM\sl\04\编程训练\）

训练3：模拟超市抹零　购物时，如果买的东西总价多出1角、2角，收银员会抹掉，只收取整钱。模拟超市抹零结账场景，输出结果如下：

```
实际应付313.04元
抹零之后支付313元
```

训练4：换季买鞋　买鞋时，如果37码鞋小，38码鞋大，则说明脚的尺寸是37.5码。因为没有37.5的鞋码，所以我们一般买38码的鞋子。利用强制类型转换来模拟此场景，输出结果如下：

```
您的脚是37.5码的尺寸
您应该买38码的鞋子
```

4.3　算术运算符与算术表达式

C语言中，运算符分为单目运算符（使用一个操作数）和双目运算符（使用两个操作数）两种。其中，算术运算符用于实现正、负、加、减、乘、除、求余等运算。单目正和单目负运算符的优先级最高。双目运算符中，乘法、除法和取模运算符的优先级比加法和减法运算符高。

4.3.1　算术运算符



算术运算符包括两个单目运算符（正和负）和5个双目运算符（即乘法、除法、取模、加法和减法）。具体符号和对应的功能如表4.1所示。

表4.1　算术运算符

符　号	功　能	符　号	功　能
+	正	%	取模
-	负	+	加法
*	乘法	-	减法
/	除法		

其中，取模运算符“%”用于计算两个整数相除得到的余数，运算符的两侧均为整数，如7%4的结果是3。“-”作为减法运算符用时为双目运算符，如5-3；作负值运算符用时为单目运算符，如-5等。

说明

单目正运算符是冗余的，只是为了与单目负运算符构成一对才存在。单目正运算符不会改变任何数值，更不会将一个负值表达式改为正值。

4.3.2　算术表达式



使用算术运算符的表达式称为算术表达式，例如：

```
Number=(3+5)/Rate;
Height= Top-Bottom+1;
Area=Height * Width;
```

需要说明的是，两个整数相除的结果为整数，如7/4的结果为1，舍去的是小数部分。但是，如果其中的一个数是负数时会出现什么情况呢？此时机器会采取“向零取整”的方法，即为-1，取整后向0靠拢。

注意

参与算术运算的两个数中只要有一个为实型，整个表达式的结果就是double型，这是因为所有实数都是按double型进行设定的。

【例4.4】使用算术表达式计算加速度（实例位置：资源包\TM\sl\04\04）

平均加速度，即速度的变化量除以这个变化所用的时间。现有一辆轿车用了8.7秒从每小时0千米加速到每小时100千米，计算并输出这辆轿车的平均加速度。

在本实例中，在表达式中使用算术运算符完成加速度计算，然后显示结果。

```
#include<stdio.h>
int main()
{
    int V1=0/3600,V2=100000/3600;        /*定义两个变量，将速度单位 km/h 转换为 m/s*/
    float t=8.7f;                        /*定义变量，表示所用时间*/
    double a;                            /*定义变量，表示加速度*/
    a=(V2-V1)/t;                         /*计算加速度*/
    printf("加速度 a=%f\n",a);
    return 0;                            /*程序结束*/
}
```

（1）在主函数main中定义两个整型变量，V1表示起始速度，V2表示加速到最后的速度，并将单位km/h转换成m/s。

（2）再定义单精度类型变量t表示所用时间，双精度类型变量a表示加速度。

（3）利用公式“加速度＝速度的变化量／所用的时间”计算加速度，将计算结果赋给变量a。

（4）利用printf函数将最后的计算结果显示在控制台上。

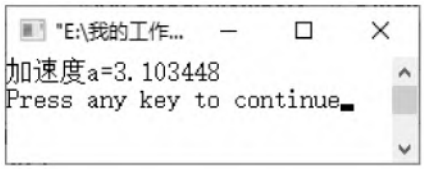


图4.5　使用算术表达式计算加速度

运行程序，显示效果如图4.5所示。

4.3.3　优先级与结合性



C语言中规定了各种运算符的优先级和结合性，首先来看一下有关算术运算符的优先级。

1．算术运算符的优先级

表达式求值时，通常会按照运算符的优先级别从高到低依次执行。算术运算符中，“*”“/”“%”的优先级别高于“+”“-”。如果同时出现“*”和“+”，应先运算乘法（这和数学中是一样的）。例如：

```
R=x+y*z;
```

上述表达式中，先进行y*z的运算，最后再加上x。使用括号将表达式改为（a+b）*c，可将“+”的运算级别提高，使其先进行运算。注意，括号在所有运算符中优先级别最高。

2．算术运算符的结合性

当算术运算符的优先级相同时，结合方向为“自左向右”。例如：

```
a-b+c
```

这里，b先与减号相结合，执行a-b的操作，然后执行加c的操作。这样的操作过程称为自左向右的结合性。

4.3.4 自增 / 自减运算符



C语言中还有两个特殊的运算符，即自增运算符“++”和自减运算符“--”。自增、自减运算符对变量的操作效果分别是使变量增加1和减少1。

自增、自减运算符可以放在变量的前面或者后面，放在变量前面称为前缀，放在后面称为后缀。使用的一般方法如下：

```
--Counter;          /*自减前缀符号*/
Grade--;            /*自减后缀符号*/
++Age;              /*自增前缀符号*/
Height++;           /*自增后缀符号*/
```

上述代码比较简单，只有自增、自减，没有其他的表达式运算，因此，运算符是前缀还是后缀不重要，得到的结果是一样的，自减就是变量减1，自增就是变量加1。

注意

当自增、自减运算符出现在表达式内部，作为运算的一部分时，前缀和后缀对运算结果的影响是不一样的。如果运算符放在变量前，那么变量将先完成自增或自减运算，再以增减后的结果参与表达式运

算；如果运算符放在变量后，那么变量将先参加表达式运算，之后再进行自增或自减。

【例4.5】千变万化的自增、自减运算（实例位置：资源包\TM\sl\04\05）

在本实例中定义一些变量，为变量赋相同的值，然后观察表达式中自增、自减运算符作为前缀和后缀时的不同结果。

```
#include<stdio.h>
int main()
{
    int iNumber1=3;                                  /*定义变量，赋值为 3*/
    int iNumber2=3;

    int iResultPreA,iResultLastA;                    /*定义变量，存储自增运算的前缀和后缀结果*/
    int iResultPreD,iResultLastD;                    /*定义变量，存储自减运算的前缀和后缀结果*/

    iResultPreA=++iNumber1;                          /*前缀自增运算*/
    iResultLastA=iNumber2++;                         /*后缀自增运算*/

    printf("The Addself ...\n");
    printf("the iNumber1 is :%d\n",iNumber1);        /*显示自增运算后自身的数值*/
    printf("the iResultPreA is :%d\n",iResultPreA);  /*得到自增表达式中的结果*/
    printf("the iNumber2 is :%d\n",iNumber2);        /*显示自增运算后自身的数值*/
    printf("the iResultLastA is :%d\n",iResultLastA);/*得到自增表达式中的结果*/

    iNumber1=3;                                      /*恢复变量的值为 3*/
    iNumber2=3;

    iResultPreD=--iNumber1;                          /*前缀自减运算*/
    iResultLastD=iNumber2--;                         /*后缀自减运算*/

    printf("The Deleteself ...\n");
    printf("the iNumber1 is :%d\n",iNumber1);        /*显示自减运算后自身的数值*/
    printf("the iResultPreD is :%d\n",iResultPreD);  /*得到自减表达式中的结果*/
    printf("the iNumber2 is :%d\n",iNumber2);        /*显示自减运算后自身的数值*/
    printf("the iResultLastD is :%d\n",iResultLastD) ;/*得到自减表达式中的结果*/

    return 0;                                        /*程序结束*/
}
```

（1）在程序代码中，定义iNumber1和iNumber2两个变量用来进行自增、自减运算。

（2）进行自增运算，分为前缀自增和后缀自增。通过程序最终的显示结果可以看到，自增变量iNumber1和iNumber2的结果同为4，但是得到表达式结果的两个变量iResultPreA和iResultLastA却不一样。iResultPreA的值为4，iResultLastA的值为3，因为前缀自增使得iResultPreA变量先进行自增操作，然后进行赋值操作；后缀自增操作是先进行赋值操作，然后进行自增操作。因此两个变量得到表达式的结果值是不一样的。

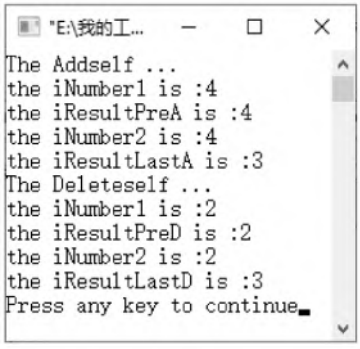


图4.6　自增、自减运算符前缀与后缀

（3）在自减运算中，前缀自减和后缀自减与自增运算方式是相同的，前缀自减是先进行减1操作，然后赋值操作；而后缀自减是先进行赋值操作，再进行自减操作。

运行程序，显示效果如图4.6所示。

注意

自增、自减运算符是单目运算符，因此表达式和常量不可以进行自增、自减运算。例如，5++，（a+5）++都是不合法的。

编程训练（答案位置：资源包\TM\sl\04\编程训练\）

训练5：统计得票数　利用自增运算符模拟统计综艺节目《演员的诞生》中丫头的得票数。每个导师投一票，得票数就会增加。输出的结果如下：

```
观众投票之后丫头得156票
导师一投给丫头之后得157票
导师二投给丫头之后得158票
导师三投给丫头之后得159票
```

训练6：统计剩余车位数量　一个新建小区内有70个停车位。现有一批新进住户购买车位，使用自减运算符在控制台中计算剩余的车位数。输出结果如下：

```
剩余停车位数：69
剩余停车位数：68
剩余停车位数：67
```

4.4　关系运算符与关系表达式

在数学中，经常需要比较两个数的大小。C语言中，关系运算符的作用就是判断两个操作数的大小关系。

4.4.1　关系运算符



关系运算符包括大于、大于等于、小于、小于等于、等于和不等于6种运算符，如表4.2所示。

表4.2　关系运算符

符　号	功　能	符　号	功　能
>	大于	<=	小于等于
>=	大于等于	==	等于
<	小于	!=	不等于

4.4.2　关系表达式



关系运算符可对两个值进行比较，返回一个真值或者假值。真值为1，表示指定的关系成立；假值为0，表示指定的关系不成立。例如：

```
7>5                    /*7 大于 5，该关系成立，表达式结果为真*/
416>=426               /*416 小于 426，该关系不成立，表达式结果为假*/
'a'<'b'                /*ASCII 码值'a'小于'b'，该关系成立，表达式结果为真*/
52.45<=45.5            /*52.45 大于 45.5，该关系不成立，表达式结果为假*/
'c'=='c1'              /*'c'不等于'c1'，该关系不成立，表达式结果为假*/
'y'!='t'               /*'y'不等于't'，该关系成立，表达式结果为真*/
```

误区警示

初学者要注意区分“=”和“==”，这两个运算符的含义截然不同。前者是赋值运算符，用于将等号右边的值或表达式赋给左边；后者是等于运算符，用于判断左右两侧的值是否相等。

例如，i==3中“==”是关系运算符，i=3中“=”是赋值运算符。

关系运算符常用来构造条件表达式，用在流程控制语句中。

例如，下方的if条件语句中，if（iCount<10）用于判断iCount小于10这个关系是否成立，如果成立，则为真，执行大括号中的语句块；如果不成立，则为假，不执行大括号中的语句块。。

```
if(Count<10)
{
    ...                    /*判断条件为真值时，执行此部分代码*/
}
```

注意

在if语句中进行条件判断时，如果误用了“=”，例如：

```
if(Amount=100)
{  …  }
```

上述代码希望能检验变量Amount是否等于常量100，但事实上没有起到检验效果。因为表达式使用的是赋值运算符“=”而不是等于运算符“==”。赋值表达式“Amount=100”本身也是表达式，其返回值是100。既然是100，说明是非零值也就是真值，则该表达式的值始终为真值，起不到条件判断的作用。如果赋值表达式右侧不是常量100，而是变量，则赋值表达式的真值或假值就由这个变量的值决定。

构造条件表达式时，这两个运算符很容易被混淆，初学者一定要格外注意。

4.4.3 优先级与结合性



关系运算符的结合性是自左向右的。使用关系运算符时常常会判断两个表达式的关系，但是由于运算符存在着优先级的问题，因此如果不小心处理则会出现错误。例如，先对一个变量进行赋值，然后判断这个赋值的变量是否不等于一个常数，代码如下：

```
if(Number=NewNum!=10)
{
    …
}
```

因为“!=”运算符比“=”的优先级要高，所以NewNum!=10的判断操作会在赋值之前实现，变量Number得到的就是关系表达式的真值或者

假值，这样并不会按照之前的意愿执行。

前文曾经介绍过括号运算符，其优先级具有最高性，因此应该使用括号来表示需要优先进行计算的表达式，例如：

```
if((Number=NewNum)!=10)
{
    …
}
```

这种写法比较清楚，不会产生混淆，没有人会对代码的含义产生误解。由于这种写法格式比较精确、简洁，因此被多数程序员所接受。

【例4.6】可乐vs矿泉水（实例位置：资源包\TM\sl\04\06）

一个小孩跑完步去买水，如果他手里的零钱多于2元，卖家会说：“你可以买可乐。”如果手里的零钱少于2元，卖家会说：“你可以买矿泉水。”试用if语句模拟两人对话的场景。

在本实例中，定义一个变量表示钱数，使用if语句判断钱数与2的大小，通过printf函数输出建议信息，得到比较的结果。

```
#include <stdio.h>
int main()
{
    int money;                          /*定义变量，用来保存钱数*/
    printf("输入手里的钱数: \n");          /*提示信息*/
    scanf("%d",&money);                 /*输入钱数*/
    if(money>=2)                        /*利用关系运算符判断钱数是否大于等于 2*/
        printf("你可以买可乐\n");          /*提示信息*/
    if(money<2)                         /*利用关系运算符判断钱数是否小于 2*/
        printf("你可以买矿泉水\n");        /*提示信息*/
    return 0;                           /*程序结束*/
}
```

（1）定义变量money，用scanf函数输入money的值。

（2）利用if语句进行判断，在判断条件中使用了关系表达式，判断钱数是否使得表达式成立。如果成立，则返回真值；如果不成立，则返

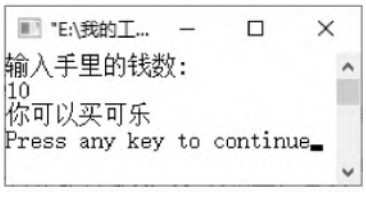


图4.7　关系运算符的使用

回假值。最后根据真值和假值选择不同的输出信息。

运行程序，显示效果如图4.7所示。

编程训练（答案位置：资源包\TM\sl\04\编程训练\）

训练7：买口红　某明星主播购买口红时，如果口红价格在500元以下，就会买下；超过500元，就会放弃。利用关系运算符模拟这一场景，输出结果如下：

结果一：

```
请输入口红的价格：
255
太便宜了，买它~~~~
```

结果二：

```
请输入口红的价格：
599
太贵了，买不起！！！
```

训练8：面积之争　使用关系运算符比较扇形和圆锥谁的面积大（半径相等）。要求定义几个变量，用来保存半径、高等数据，输出结果如下：

```
圆锥的面积比扇形的面积大
圆锥的侧面积是18.84平方厘米
扇形的面积是3.50平方厘米
```

4.5　逻辑运算符与逻辑表达式

逻辑运算符可根据表达式的真假属性返回真值或假值。在C语言中，表达式的值非零，那么其值为真。非零的值用于逻辑运算，则等价于1；假值总是为0。

4.5.1 逻辑运算符



逻辑运算符有3种，如表4.3所示。

表4.3 逻辑运算符

符　号	功　能	示　例	含　义
&&	逻辑与	A&&B	A、B 两个表达式都为真时，逻辑运算结果为真 A、B 两个表达式只要有一个为假，逻辑运算结果为假
||	逻辑或	A||B	A、B 两个表达式都为假时，逻辑运算结果为假 A、B 两个表达式只要有一个为真，逻辑运算结果为真
!	逻辑非	!A	A 表达式为真，逻辑运算结果为假 A 表达式为假，逻辑运算结果为真

4.5.2 逻辑表达式



前文介绍过，关系运算符可用于对两个操作数进行比较。使用逻辑运算符，可以将多个关系表达式的结果合并在一起进行判断。其一般形式如下：

```
表达式　逻辑运算符　表达式
```

例如，使用逻辑运算符的几个示例：

```
Result= Func1&&Func2;                    /*Func1 和 Func2 都为真时，Result 为真*/
Result= Func1||Func2;                    /*Func1、Func2 只要有一个为真，Result 就为真*/
Result= !Func2;                          /*如果 Func2 为真，则 Result 为假*/
```

逻辑与运算符和逻辑或运算符可用于复杂的表达式中。例如，在后面要学的if、for、while语句等流程控制语句中构造条件表达式。

在程序中，通常使用单目逻辑非运算符“!”把一个变量的数值转换为相应的逻辑真值或者假值，也就是1或0。例如：

```
Result= !Value;                                    /*转换成逻辑值*/
```

4.5.3 优先级与结合性



“&&”和“||”是双目运算符，要求有两个操作数，结合方向自左至右；“!”是单目运算符，要求有一个操作数，结合方向自右向左。逻辑运算符的优先级从高到低依次为“!”“&&”和“||”。

【例4.7】模特公司招聘（实例位置：资源包\TM\sl\04\07）

某模特公司招聘，要求年龄大于等于25岁且身高大于等于1.7米，才通知参加面试和笔试，否则就不能进行面试和笔试。模拟此场景，实现代码如下：

```
#include<stdio.h>
int main()
{
    float height;                                  /*定义变量，表示身高*/
    int age;                                       /*定义变量，表示年龄*/
    printf("请输入您的年龄和身高:\n");              /*输出提示信息*/
    scanf("%d %f", &age, &height);                 /*输入年龄、身高*/
    if (age>=25&&height>=1.7)                      /*如果年龄大于或等于 25 岁且身高大于或等于 1.7 米
    {
        printf("恭喜您通过了本公司的初选，请您及时到本公司参加面试和笔试。\n");  /*输出初选通过信息*
    }
    else                                           /*不符合条件*/
    {
        printf("对不起，您不符合我们公司的要求!\n");  /*输出婉拒信息*/
    }
    return 0;                                      /*程序结束*/
}
```

（1）先定义两个变量，height表示身高，age表示年龄。利用scanf函数输入两个变量的值。

（2）利用if语句判断年龄是否大于等于25岁，且身高大于等于1.7米。使用逻辑与运算符来保证两个条件同时成立。如果满足条件，输出初选通过提示；只要有一个条件不满足，输出婉拒信息。

运行程序，显示效果如图4.8所示。

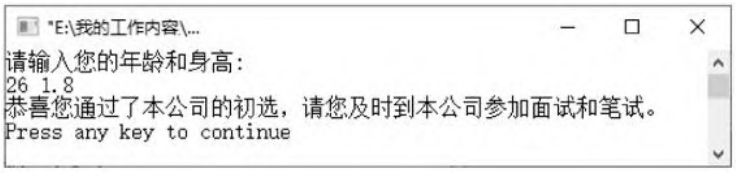


图4.8　逻辑运算符的应用

编程训练（答案位置：资源包\TM\sl\04\编程训练\）

训练9：判断用户是否可以登录网站　在明日学院网站首页中，可以使用账户名登录，也可以使用手机号登录，还可以使用电子邮箱地址登录。请判断某用户是否可以登录。已知服务器中有如下记录，账户名：张三，手机号：1234567890，电子邮箱：zhangsan@163.com。输出结果如下：

```
the 1 is true,0 is false
用户是否可以登录：0
```

训练10：模拟打折活动　某手机店在每周二上午10~11点、周五下午14~15点进行打折促销活动。使用逻辑运算符编写程序，模拟是否可参加手机店的打折活动。输出结果如下：

```
请输入星期几、时间：
2 11
恭喜你，可参加手机店的打折活动
```

4.6　逗号运算符与逗号表达式



C语言中，可用逗号运算符将多个表达式分隔开来。逗号表达式的一般形式如下：

```
表达式1,表达式2,…,表达式n
```

逗号表达式又称为顺序求值运算符，其求解过程是：先求解表达式1，再求解表达式2，一直求解到表达式n。整个逗号表达式的值是表达式n的值。

但观察下面的逗号表达式：

```
Value=2+5,1+2,5+7;
```

上述语句中，Value的值为7，而非12。为什么逗号表达式的值不是最后一个表达式的值12呢？问题在于优先级。赋值运算符的优先级比逗号运算符高，因此先执行赋值运算。如果要先执行逗号运算，则可以使用括号运算符，代码如下：

```
Value=(2+5,1+2,5+7);
```

使用括号之后，Value的值为12。

【例4.8】奇妙的逗号运算符（实例位置：资源包\TM\sl\04\08）

在本实例中，通过逗号运算符将其他运算符结合在一起形成表达式，再将表达式的最终结果赋值给变量。根据变量的值分析逗号运算符的计算过程。

```
#include<stdio.h>
int main()
{
    int iValue1,iValue2,iValue3,iResult;          /*定义 4 个变量*/
    iValue1=10;                                   /*为变量赋值*/
    iValue2=43;
    iValue3=26;
    iResult=0;

    iResult=iValue1++,--iValue2,iValue3+4;        /*计算第一个逗号表达式*/
    printf("the result is :%d\n",iResult);        /*将结果输出显示*/

    iResult=(iValue1++,--iValue2,iValue3+4);      /*计算第二个逗号表达式*/
    printf("the result is :%d\n",iResult);        /*将结果输出显示*/
    return 0;                                     /*程序结束*/
}
```

（1）代码开始处，定义变量时使用了逗号运算符，分隔多个变量，然后分别进行赋值。

（2）在第一个逗号表达式中，赋值的变量进行各自的计算，变量iResult得到表达式的结果。这里需要注意的是，通过输出可以看到iResult的值为10，从前面的讲解知道因为逗号表达式没有使用括号运算符，所以iResult得到第一个表达式的值。在第一个表达式中，iValue1变量进行的是后缀自加操作，于是iResult先得到iValue1的值，iValue1再进行自加操作。

（3）在第二个逗号表达式中，由于使用了括号运算符，因此iResult变量得到的是第三个表达式iValue3+4的值，iResult变量赋值为30。

运行程序，显示效果如图4.9所示。

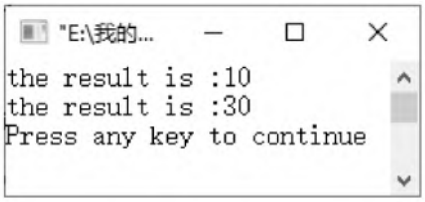


图4.9　奇妙的逗号运算符

编程训练（答案位置：资源包\TM\sl\04\编程训练\）

训练11：逗号运算1　定义3个变量x，y，z，分别赋值为3，3，1。思考语句prinf（"%d，%d，%d "，（++x，y++），z+x+y+2）的输出结果。输出结果如下：

```
结果为：
3,10,4198768
```

训练12：逗号运算2　编写程序，用逗号运算符计算n=（（1+2，3），9）。输出结果如下：

```
((1 + 2, 3), 9)=9
```

4.7　复合赋值运算符



复合赋值运算符是C语言所独有的，实际这种操作是一种缩写形式，可使变量操作的描述方式更为简洁。例如，为变量Value赋值，值为Value自身与3相加的结果，代码如下：

```
Value=Value+3;
```

使用复合赋值运算符，可以实现同样的操作，但描述更为简洁。代码如下：

```
Value+=3;
```

可见，使用复合赋值运算符不但可以简化程序，使代码更为精练，而且可以提高编译效率。

对于简单赋值运算符，如Func=Func+1中，表达式Func计算两次；对于复合赋值运算符，如Func+=1中，表达式Func仅计算一次。一般来说，这种区别对于程序的运行没有太大的影响，但如果表达式中存在某个函数的返回值，那么函数将被调用两次。

【例4.9】简化版赋值运算（实例位置：资源包\TM\sl\04\09）

在本实例中，定义一个整型变量iValue =7，计算iValue += iValue *= iValue /= iValue -5的值。代码如下：

```
#include<stdio.h>                          /*包含头文件*/
int main()                                 /*主函数 main*/
{
    int iValue;                            /*定义变量 iValue*/
    iValue=7;                              /*为变量 iValue 赋值*/
    iValue+=iValue*=iValue/=iValue-5;      /*计算得到 iValue 变量值*/
    printf("the result is %d\n",iValue);   /*将计算结果输出*/
    return 0;                              /*程序结束*/
}
```

（1）代码第6行中使用复合赋值运算符，运算顺序从右往左计算，其中iValue/=iValue-5表示的意思是iValue的值等于iValue/（iValue-5）的结果，即iValue=3。

（2）iValue*=iValue/=iValue-5表示的意思是iValue的值等于（1）中iValue的结果乘以iValue，即iValue=9。

（3）“iValue+=iValue*=iValue/=iValue-5;”表示的意思是iValue的值等于（2）中iValue的结果加上iValue，即最终结果iValue=18。利用printf函数将结果输出。

运行程序，显示效果如图4.10所示。

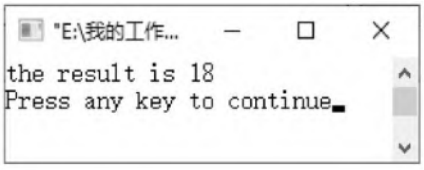


图4.10　复合赋值运算符

编程训练（答案位置：资源包\TM\sl\04\编程训练\）

训练13：求n的阶乘　阶乘的计算公式为n!=1×2×3×…×n。例如，4!=1×2×3×4=24。输入n的值，用程序计算n的阶乘。输出结果如下：

```
请输入n的值：
5
5的阶乘是120
```

训练14：计算a+=a*=a/=a-5的结果　定义一个变量a，将输入的年龄赋给a，再计算表达式a+=a*=a/=a-5的结果。输出结果如下：

```
请输入年龄值：
28
结果是 2
```

4.8　运算符的优先级总结

为了方便学习，下面按照优先级从小到大的排列顺序列出C语言中所有运算符的优先级和结合性，如表4.4所示。

表4.4　运算符的优先级和结合性

优 先 级	运 算 符	含 义	结 合 性
1	()	圆括号	自左向右
	[]	下标运算符	
	->	指向结构体成员运算符	
	.	结构体成员运算符	
2	!	逻辑非运算符（单目运算符）	自右向左
	~	按位取反运算符（单目运算符）	
	++	自增运算符（单目运算符）	
	—	自减运算符（单目运算符）	
	-	负号运算符（单目运算符）	
	*	指针运算符（单目运算符）	
	&	地址与运算符（单目运算符）	
	sizeof	长度运算符（单目运算符）	
3	*、/、%	乘法、除法、求余运算符	自左向右
4	+、-	加法、减法运算符	
5	<<、>>	左移、右移运算符	
6	<、<=、>、>=	小于、小于或等于、大于、大于或等于运算符	
7	==、!=	等于、不等于运算符	
8	&	按位与运算符	
9	^	按位异或运算符	
10	|	按位或运算符	
11	&&	逻辑与运算符	
12	||	逻辑或运算符	
13	?:	条件运算符（三目运算符）	自右向左
14	=、+=、-=、*=、/=、%=、>>=、<<=、&=、^=、|=	赋值运算符	
15	,	逗号运算符（顺序求值运算符）	自左向右

表4.4中，有些运算符本章中并未介绍，但在后续学习中会逐渐接触到。这里大家了解即可，在后续遇到的时候再仔细揣摩。

4.9 实践与练习

（答案位置：资源包\TM\sl\04\实践与练习\）

综合练习1：连加计算　编写程序，用户输入3个数字，输出连加结果。3个数字需要分别输入，不可以一起输。例如，输入一个数字后，按Enter键可输入第二个数字；3个数字输完后，按Enter键输出计算结果。输出结果如下：

```
请输入3个数，实现连加
3
5
12
三个数的和为：20
```

综合练习2：人生路程计算器　英国专家研究发现，居住在现代城市中的人一生中大约会步行80500千米。如果人的平均寿命为70岁，一年为365天，计算现代城市人每天需要走多少千米，每小时需要走多少米。输出结果如下：

```
每天要走3.150685千米
每小时要走0.131279千米
```

综合练习3：港珠澳大桥的长度　港珠澳大桥位于广东省伶仃洋区域内，桥隧全长55千米，其中主桥29.6千米，香港口岸至珠澳口岸41.6千米；桥面为双向六车道高速公路，设计速度100千米／小时。将港珠澳大桥的全长转换成古代的丈、尺单位（1丈=10尺，3尺=1米）表示。输出结果如下：

```
港珠澳大桥全长378600.000000尺
港珠澳大桥全长37860.000000丈
```

综合练习4：支付宝年账单　使用算术运算符编写一个显示支付宝年账单的代码，将过去一年中12个月的花销情况相加。输出结果如下：

```
-------------------------------------------------------------------------------------------
|   1-6 月分别在支付宝上花费 3588 元、3000 元、3888 元、2100 元、4565 元、2509 元   |
|   7-12 月分别在支付宝上花费 4551 元、4839 元、3609 元、4142 元、4895 元、4981 元  |
-------------------------------------------------------------------------------------------

          ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★
          ★   这一年在支付宝上总花费 46667   ★
          ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★
```

综合练习5：模拟饲料产生过程　蚂蚁庄园是支付宝推出的网上公益活动。网友使用支付宝付款来领取鸡饲料，使用饲料喂鸡之后可以获得鸡

蛋，通过鸡蛋可以进行爱心捐赠。模拟蚂蚁庄园一日产生的鸡饲料数量（完成一次支付产生150g饲料）。输出结果如下：

```
请输入在线支付次数、到店支付次数：
1 2
▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼ ▼
    今天蚂蚁庄园一共产生 540g 鸡饲料
▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲
```

综合练习6：统计歌手排名　《歌手2019》中，首发歌手有刘欢、杨坤、齐豫、吴青峰、Kristian kostov、逃跑计划乐队、张芯，他们的排名会根据500位观众的投票产生。编写程序，输出各位歌手的排名情况，输出结果如下：

```
歌手 2019 第一期排名
  第一名：刘欢
  第二名：吴青峰
  第三名：齐豫
  第四名：Kristian kostov
  第五名：张芯
  第六名：杨坤
  第七名：逃跑计划乐队
```

综合练习7：计算表达式的值　使用复合赋值运算符计算a+=a*=a/=a-6。其中，a的初始值是12。输出结果如下：

```
◇-◇-◇-◇-◇-◇-◇-◇-◇-◇
   the result is 8
◇-◇-◇-◇-◇-◇-◇-◇-◇-◇
```

综合练习8：单细胞细菌繁殖实验　生物实验室做单细胞细菌繁殖实验，每代细菌数量都会成倍数增长，一代菌落中只有一个细菌，二代菌落中分裂成两个细菌，三代菌落中分裂成4个细菌，以此类推，计算第五代菌落中的细菌数量。输出结果如下：

```
☆→→→→→→→→→→→→→→→→→→→→☆
★       第五代菌落细菌数量 = 32      ★
☆→→→→→→→→→→→→→→→→→→→→☆
```

第5章
常用的数据输入 / 输出函数

C语言中，语句是用来向计算机系统发出操作指令的。当需要程序按照用户输入信息执行指令时，先要使用输入语句给程序发送指示。当程序解决了问题之后，还要使用输出语句将计算的结果显示出来。

本章致力于使读者了解有关语句的概念，掌握如何对程序的输入／输出进行操作。

本章的知识架构及重难点如下：



5.1　语句



一条语句经过编译后，会产生若干条机器指令。实际程序中通常包含若干条语句，用于完成一定的操作任务。

注意

在编写程序时，声明部分不能算作语句。例如，“int iNumber;”就不是一条语句，因为不产生机器的操作，只是对变量提前进行了定义。

通过前面的介绍可知，程序包括声明部分和执行部分，其中执行部分由语句组成。

5.2 字符数据的输入 / 输出

本节将介绍标准I/O函数库中最简单也最容易理解的字符输入／输出函数getchar和putchar。

5.2.1 字符输出函数



输出字符数据使用的是putchar函数，作用是向显示设备输出一个字符。其语法格式如下：

```
int putchar(int ch);
```

使用该函数时，要添加头文件stdio.h。其中，参数ch为要输出的字符，可以是字符型变量或整型变量，也可以是常量。例如，输出字符A的代码如下：

```
putchar('A');
```

使用putchar函数也可以输出转义字符。例如，使用转义字符输出字符A：

```
putchar('\101');
```

【例5.1】打印喵喵字符画（实例位置：资源包\TM\sl\05\01）

在程序中使用putchar函数，输出喵喵字符画“（=^_^=）”，并在输出完毕之后换行。

```
#include<stdio.h>
int main()
{
    putchar('(');                    /*输出多个字符常量*/
    putchar('=');
    putchar('^');
    putchar('_');
    putchar('^');
    putchar('=');
    putchar(')');
    putchar('\n');                   /*输出转义字符，实现换行*/
    return 0;
}
```

（1）要使用putchar函数，首先要包含头文件stdio.h。

（2）putchar函数一次只能输出一个字符。要输出字符画，就需要多次调用putchar函数。

（3）字符画输出完毕后，使用putchar函数输出转义字符“\n”，进行换行操作。

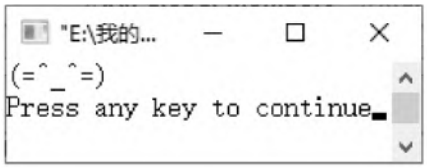


图5.1　使用putchar函数打印字符画

运行程序，显示效果如图5.1所示。

5.2.2　字符输入函数



字符数据输入使用的是getchar函数，其作用是从终端（输入设备）输入一个字符。getchar与putchar函数的区别在于getchar函数没有参数。

使用getchar函数时，需要先添加头文件stdio.h。getchar函数的语法格式如下：

```
int getchar();
```

函数的值就是从输入设备得到的字符。例如，从键盘输入一个字符并赋给变量cChar，代码如下：

```
cChar=getchar();
```

注意

getchar函数只能接收一个字符，该字符可以赋给一个字符变量或整型变量，也可以不赋给任何变量，只作为表达式的一部分，如“putchar（getchar（））;”。这里，getchar函数作为putchar函数的参数，通过getchar函数从输入设备得到一个字符，然后通过putchar函数将字符输出。

【例5.2】输出字符对应的ASCII码值（实例位置：资源包\TM\sl\05\02）

在本实例中，使用getchar函数获取键盘输入的字符，然后利用字符型和整型之间的转换，输出字符对应的ASCII码值。

```c
#include<stdio.h>
int main()
{
    char i;                                  /*定义变量 i*/
    printf("请输入字符:\n");
    i=getchar();                             /*将用户输入的字符赋给变量 i*/
    printf("对应的 ASCII 码值为%d\n",i);      /*输出对应的 ASCII 码值*/
    return 0;                                /*程序结束*/
}
```

要使用getchar函数，首先要包括头文件stdio.h。主函数中定义一个字符型变量i，通过getchar函数得到输入的字符，赋值给i，然后利用字符型和整型间的转换，用printf函数输出对应的ASCII码值。运行程序，显示效果如图5.2所示。

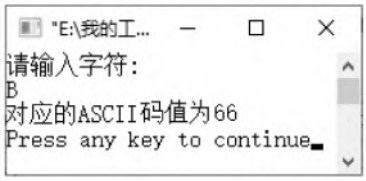


图5.2　使用getchar函数输入字符

【例5.3】getchar函数获取回车符（**实例位置：资源包\TM\sl\05\03**）

先来看一下使用getchar函数实现字符数据输入，代码如下：

```c
#include<stdio.h>
int main()
{
    char cChar1;                             /*定义变量*/
    cChar1 = getchar();                      /*将用户输入的字符赋给变量*/
    putchar(cChar1);                         /*输出字符*/
    putchar('\n');                           /*输出转义字符换行*/
    getchar();                               /*注意此行，getchar 输入字符*/
    putchar(getchar());                      /*得到输入字符，直接输出*/
    putchar('\n');                           /*换行*/
    return 0;                                /*程序结束*/
}
```

运行结果如下：

```
a
a
b
b
```

当把代码的第8行“getchar（）”这句代码删除，就可以获取回车符。代码如下：

```
#include<stdio.h>
int main()
{
    char cChar1;                          /*定义变量*/
    cChar1=getchar();                     /*将用户输入的字符赋给变量*/
    putchar(cChar1);                      /*输出字符*/
    putchar('\n');                        /*输出转义字符换行*/
                                          /*注意此行，getchar 函数被删掉了*/
    putchar(getchar());                   /*得到输入字符，直接输出*/
    putchar('\n');                        /*换行*/
    return 0;                             /*程序结束*/
}
```

运行程序，显示效果如图5.3所示。程序没有获取第二次的字符输入，而是进行了两次换行。

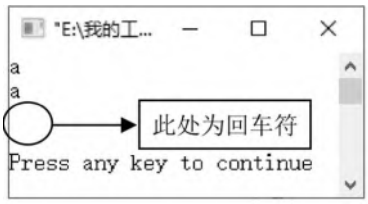


图5.3　使用getchar函数获取回车符

注意

getchar函数也可以不返回值，而是直接放在printf函数中使用。例如：

```
printf("%c\n",getchar());
```

编程训练（答案位置：资源包\TM\sl\05\编程训练\）

训练1：输出笑脸　利用putchar函数输出笑脸^_^，输出效果如下：

```
^_^
```

训练2：输出大眼萌　在程序中使用putchar函数输出大眼萌表情“@_@”，并且在字符表情输出完毕之后进行换行。输出结果如下：

```
@_@
```

5.3　字符串输入 / 输出

putchar和getchar函数都只能对单个字符进行操作，如果要操作字符串，则会很麻烦。C语言提供了两个专门对字符串进行操作的函数，分别为gets和puts函数。

5.3.1　字符串输出函数



字符串输出使用的是puts函数，作用是输出一个字符串到屏幕上。其语法格式如下：

```
int puts(char *str);
```

使用puts函数时，先要在程序中添加stdio.h头文件。其中，形式参数str是一个字符指针变量，用来接收要输出的字符串。例如，使用puts函数输出一个字符串：

```
puts("I LOVE CHINA!");                    /*输出一个字符串常量*/
```

上述语句首先会输出一个字符串，之后会自动进行换行操作。这与printf函数有所不同，在前几章的实例中，使用printf函数进行换行时，

要在其中添加转义字符“\n”。

puts函数会在字符串中判断“\0”结束符，遇到结束符时，后面的字符不再输出，并且自动换行。例如：

```
puts("I LOVE\0 CHINA!");                    /*输出一个字符串常量*/
```

在上面的语句中，加上“\0”字符后，puts函数输出的字符串就变成了“I LOVE”。

说明

前面介绍到，编译器会在字符串常量末尾自动添加“\0”结束符，因此puts函数会在输出字符串常量后自动进行换行操作。

【例5.4】打印各种各样的“ILOVECHINA”（实例位置：资源包\TM\sl\05\04）

在本实例中，使用puts函数对字符串常量“ILOVECHINA”和字符串变量进行操作，在这些操作中观察puts函数的使用方式。

```
#include<stdio.h>
int main()
{
    char* Char="ILOVECHINA";                /*定义字符串指针变量并赋初值*/

    puts("ILOVECHINA!");                    /*输出字符串常量*/
    puts("I\0LOVE\0CHINA!");                /*输出字符串常量，其中加入两个结束符“\0”*/
    puts(Char);                             /*输出字符串变量的值*/
    Char="ILOVE\0CHINA!";                   /*改变字符串变量的值*/
    puts(Char);                             /*输出改变后的字符串变量的值*/
    return 0;                               /*程序结束*/
}
```

（1）字符串常量赋值给字符串指针变量。有关字符串指针的内容将在第10章中介绍，此时可以将其看作整型变量。为其赋值后，就可以使用该变量。

（2）第一次使用puts函数输出的字符串常量中，由于在该字符串中没有结束符“\0”，所以会完整输出整个字符串，直到最后编译器为其添加结束符“\0”为止。

（3）第二次使用puts函数输出的字符串常量中，人为添加了两个“\0”，因此只能输出第一个结束符之前的字符，然后进行换行操作。

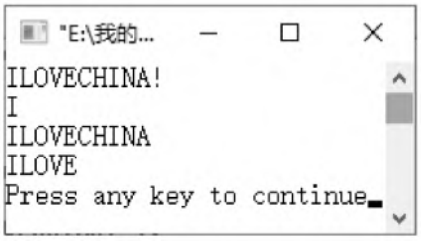


图5.4　使用puts函数输出字符串

（4）第三次使用puts函数输出的是字符串指针变量，函数根据变量的值进行输出。因为在变量的值中没有结束符，所以会完整输出整个字符串，直至最后编译器为其添加结束字符，然后进行换行操作。

（5）改变变量的值，在使用puts函数输出变量时，由于变量的值中包含一个结束符“\0”，因此将输出第一个结束符之前的所有字符，然后进行换行操作。

运行程序，显示效果如图5.4所示。

5.3.2　字符串输入函数



字符串输入使用的是gets函数，作用是将读取的字符串（即用户输入的字符串）保存在str变量中，直到出现新的一行为止。其新行的换行符将会转换为空终止符“\0”。

使用gets函数前，要添加头文件stdio.h。gets函数的语法格式如下：

```
char *gets(char *str);
```

其中，str是一个字符指针变量，用来存储用户输入的字符串。例如，定义字符数组变量cString，然后使用gets函数获取输入字符，代码如下：

```
gets(cString);
```

上面的代码中，cString变量获取了字符串，并将最后的换行符转换成了终止字符。

【例5.5】模拟用户注册系统（实例位置：资源包\TM\sl\05\05）

模拟用户注册系统。首先提醒用户输入账号、密码、姓名和身份证号，注册完毕后，显示用户输入的注册信息。代码如下：

```
#include <stdio.h>
int main()
{
    char account[20],password[20],name[20],IDcard[20];      /*定义 4 个字符数组变量*/

    puts("请输入账号：");                                   /*提示信息*/
    gets(account);                                          /*获取账号字符串*/
    puts("请输入密码：  ");                                 /*提示信息*/
    gets(password);                                         /*获取密码字符串*/
    puts("请输入姓名：  ");                                 /*提示信息*/
    gets(name);                                             /*获取姓名字符串*/
    puts("请输入身份证号：  ");                             /*提示信息*/
    gets(IDcard);                                           /*获取身份证号字符串*/

    puts("注册完成，信息如下：");
    puts("您输入的账号为：");                               /*提示信息*/
    puts(account);                                          /*输出账号字符串*/
    puts("您输入的密码为：");                               /*提示信息*/
    puts(password);                                         /*输出密码字符串*/
    puts("您输入的姓名为：");                               /*提示信息*/
    puts(name);                                             /*输出姓名字符串*/
    puts("您输入的身份证号为：");                           /*提示信息*/
    puts(IDcard);                                           /*输出身份证号字符串*/
    return 0;                                               /*程序结束*/
}
```

（1）因为要接收用户输入的账号、密码、姓名、身份证号等信息，所以定义了4个可以接收字符串的变量。account、password、name、

IDcard为字符数组变量的标识符。关于字符数组的内容将第8章中介绍，此处知道此变量可以接收字符串即可。

（2）调用gets函数，函数参数为前面定义的account、password、name、IDcard 4个变量。调用gets函数后，程序会等待用户输入字符，用户输入完一个字符串并按Enter键确定时，gets函数将获取字符串结束标志“\0”，并将对应的字符串保存到上述4个变量中。

（3）使用puts函数将获取的字符串依次输出。

运行程序，显示效果如图5.5所示。

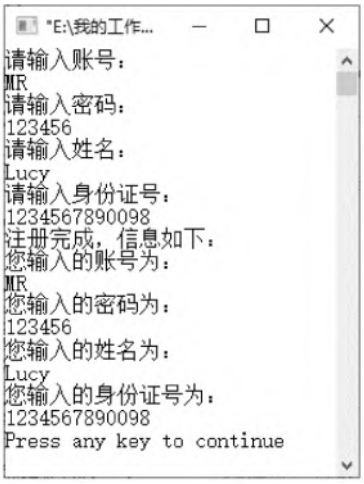


图5.5　使用gets函数获取注册信息

编程训练（答案位置：资源包\TM\sl\05\编程训练\）

训练3：输出乔布斯语录　苹果公司的创始人乔布斯在2005年给斯坦福大学做毕业演讲中提到一句话：“Stay hungry，Stay foolish.”。编写程序，输出该条语录的英文版和中文版。输出效果如下：

```
ΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦ
    Stay hungry      Stay foolish.
     求知若饥            虚心若愚
ΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦΦ
```

训练4：输出梁山好汉　编写程序，输出《水浒传》中的10个梁山好汉。输出效果如下：

```
~~~~~~~~~~~~~~~~~~~~~~~~
① 呼保义  宋江
② 玉麒麟  卢俊义
③ 智多星  吴用
④ 入云龙  公孙胜
⑤ 大刀    关胜
⑥ 豹子头  林冲
⑦ 霹雳火  秦明
⑧ 双鞭    呼延灼
⑨ 小李广  花荣
⑩ 小旋风  柴进
~~~~~~~~~~~~~~~~~~~~~~~~
```

5.4 格式输出函数



第1章中我们就接触到了scanf函数和printf函数。其中，printf函数是格式输出函数。

printf函数的作用是向终端（输出设备）输出若干任意类型的数据，其语法格式如下：

```
printf(格式控制,输出列表)
```

1．格式控制

格式控制是用双引号括起来的字符串，也称为转换控制字符串。其中包括格式字符和普通字符。

- ☑ 格式字符：用来进行格式说明的字符，作用是将输出的数据转换为指定的格式。格式字符通常以“%”开头。
- ☑ 普通字符：需要原样输出的字符，包括双引号内的逗号、空格和换行符。

2．输出列表

输出列表列出的是要进行输出的一些数据，可以是变量或表达式。

例如，要输出一个整型变量，代码如下：

```
int iInt=10;
printf("this is %d",iInt);
```

执行上面的语句，输出的字符是“this is 10”。格式控制双引号中的字符是“this is %d”，其中的“this is”字符串是普通字符，而“%d”是格式字符，表示输出的是后面的iInt数据。

printf是函数，“格式控制”和“输出列表”都是函数参数，因此printf函数的一般形式可表示为：

```
printf(参数1,参数2,…,参数n)
```

函数中的每个参数都必须按照给定的格式和顺序依次输出。例如，输出一个字符型变量和整型变量的代码如下：

```
printf("the Int is %d,the Char is %c",iInt,cChar);
```

表5.1列出了有关printf函数的格式字符。

表5.1　printf函数的格式字符

格式字符	功能说明
%d，%i	用来输出有符号的十进制整数
%u	用来输出无符号的十进制整数
%o	用来输出无符号的八进制整数
%x，%X	用来输出无符号的十六进制整数。%x 表示以小写形式输出；%X 表示以大写形式输出
%c	用来输出单个字符
%s	用来输出字符串
%f	用来输出实数，以小数形式输出
%e，%E	用来输出实数，以指数形式输出
%g，%G	以%f 或%e 中宽度较短的格式输出，不输出无意义的 0

注意

printf函数中，除X、E、G外，其他格式字符必须使用小写字母，如%d不能写成%D。

【例5.6】牛吃草问题（实例位置：资源包\TM\sl\05\06）

一头牛可以吃2千克草，现有45千克草，可供几头牛吃饱。首先定义相应的变量，然后利用表达式计算结果，最后用printf函数对结果进行输出。具体代码如下：

```
#include <stdio.h>                          /*包含头文件*/
int main()                                  /*主函数 main*/
{
    int graNum,graSum,num;                  /*graNum 为一头牛吃的草量，graSum 为总草量，num 为结果
    graNum=2,graSum=45;                         /*为变量赋值*/
    num=graSum/graNum;                          /*利用表达式计算结果*/
    printf("%d cows eat enought\n",num);        /*将结果输出*/
    return 0;                                   /*程序结束*/
}
```

程序中定义了3个整型变量，并为变量赋值。第6行使用算术表达式计算结果，并将结果赋给变量num。在printf函数中，使用格式符号“%d”进行输出。

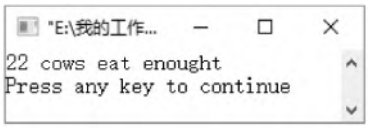


图5.6 使用printf函数求解牛吃草

运行程序，显示效果如图5.6所示。

另外，在格式说明中，在“%”符号和上述格式字符间可以插入如表5.2所示的几种附加符号。

表5.2 printf函数的附加格式说明字符

附加字符	功能说明
l	用于输出长整型整数，可加在d、o、x、u的字母前面
m	用于指定输出数据的字段宽度，数字或字符在域内向右对齐
n	对实数，表示输出n位小数；对字符串，表示截取的字符个数
-	用于指定输出的数字或字符在域内向左对齐

如果想输出“%”符号，可在格式控制处使用“%%”进行输出。

【例5.7】打印变幻莫测的“MingRiKeJi”（实例位置：资源包\TM\sl\05\07）

在本实例中，使用printf函数的附加格式说明字符，对输出的数据进行更为精准的格式设计。

```
#include<stdio.h>
int main()
{
    long iLong=100000;                              /*定义长整型变量，为其赋值*/
    printf("the Long is %ld\n",iLong);              /*使用%ld 输出长整型变量*/

    printf("the string is: %sKeJi\n","MingRi");     /*%s，输出字符串"MingRi"（按实际长度输出）*/
    printf("the string is: %10sKeJi\n"," MingRi");  /*%10s，输出宽度为 10，右对齐，左补空*/
    printf("the string is: %-10sKeJi\n"," MingRi"); /*%-10s，输出宽度为 10，左对齐，右补空*/
    printf("the string is: %10.2sKeJi\n"," MingRi"); /*%10.2 s，输出宽度为 10，截取 2 个字符右对齐输出*
    printf("the string is: %-10.2sKeJi\n"," MingRi"); /*%-10.2 s，输出宽度为 10，截取 2 个字符左对齐输出
    return 0;
}
```

（1）在程序代码中，定义的长整型变量在使用printf函数输出时，应该用“%ld”格式字符。

（2）“%s”用来输出一个字符串的格式字符，在结果中可以看到输出了字符串“MingRi”。

（3）“%10s”格式为“%ms”，表示输出字符串占m列。如果字符串长度大于m，则突破m的限制，将字符串全部输出；若字符串长度小于m，则用空格进行左补齐。可以看到在字符串“MingRi”前存在4个空格。

（4）“%-10s”格式为“%-ms”，表示如果字符串长度小于m，则在m列范围内，字符串向左对齐，右补空格。

（5）“%10.2s”格式为“%m.ns”，表示输出占m列，但只取字符串左端n个字符。这n个字符输出在m列的右侧，左补空格。

（6）“%-10.2s”格式为“%-m.ns”，其中m、n含义同上，n个字符输出在m列的左侧，右补空格。如果n>m，则m自动取n值，即保证

n个字符正常输出。

运行程序，显示效果如图5.7所示。



图5.7 在printf中使用附加格式说明字符

编程训练（答案位置：资源包\TM\sl\05\编程训练\）

训练5：一日之计在于晨 定义两个变量“char* str="日之计在于晨"”和“int num=1”，在控制台中输出“一日之计在于晨”。

训练6：鸡兔同笼问题 鸡、兔关在同一只笼子里，已知有35个头、94只脚，问笼子里几只鸡和几只兔。计算公式为：鸡的数量＝（4*鸡兔总头数-鸡兔总脚数）/2；兔的数量＝（鸡兔总脚数-2*鸡兔总头数）/2。运行结果如下。

```
笼中有23只鸡，有12只兔子!
```

5.5 格式输入函数



与格式输出函数printf相对应的是格式输入函数scanf。该函数的功能是按照指定的格式接收用户在键盘上输入的数据，最后将数据存储在指定的变量中。

scanf函数的一般格式如下：

```
scanf(格式控制,地址列表)
```

其中，“格式控制”参数与printf函数相同，“%d”表示十进制整型，“%c”表示单个字符；“地址列表”参数用于给出接收数据变量的地址。

注意

scanf函数参数的“地址列表”处要使用变量地址，而不是变量的标识符，否则会编译报错。

例如，得到一个整型数据的代码如下：

```
scanf("%d",&iInt);                    /*得到一个整型数据*/
```

在上面的代码中，“&”是取地址符号，表示取iInt变量的地址。用户不用关心变量的地址具体是多少，只要在变量的标识符前加“&”，就可以表示存取变量的地址。

误区警示

scanf函数中，当输入的数据是基本类型，如整型、字符型、实型时，需要使用取地址符号。当输入的变量本身就是一个地址时，如数组名、字符串指针等，不需要再使用“&”符号。

关于数组和指针的知识将在第8章和第10章中介绍，这里了解就行。

表5.3中列出了scanf函数中常用的格式字符。

表5.3　scanf函数的格式字符

格式字符	功能说明
%d，%i	用来输入有符号的十进制整数
%u	用来输入无符号的十进制整数
%o	用来输入无符号的八进制整数
%x，%X	用来输入无符号的十六进制整数（大小写作用相同）
%c	用来输入单个字符
%s	用来输入字符串
%f	用来输入实数，以小数形式输入
%e，%E	用来输入实数，以指数形式输入
%g，%G	用来输入实数，与%f和%e作用相同

说明

格式字符“%s”用来输入字符串。将字符串送到一个字符数组中，在输入时以非空白字符开始，以第一个空白字符结束。字符串以串结束标志“\0”作为最后一个字符。

【例5.8】根据身高和体重计算BMI值（实例位置：资源包\TM\sl\05\08）

输入身高和体重，计算身体质量指数（BMI），计算公式为“BMI=体重／身高^2”。

在本实例中，利用scanf函数得到用户输入的两个数据，即身高和体重，因为scanf函数只能用于输入操作，所以若在屏幕上显示信息，还需要使用输出函数。

```
#include <stdio.h>
int main()
{
    float height,weight;                          /*定义 height 为身高，weight 为体重*/
    double BMI;                                   /*定义身体质量指数 BMI*/
    printf("请输入身高、体重：");                  /*提示信息*/
    scanf("%f %f",&height,&weight);               /*输入身高、体重*/
    BMI=weight/(height*height);                   /*计算 BMI 值*/
    printf("您的身体质量指数（BMI）:%f\n",BMI);    /*输出 BMI 值*/
    return 0;                                     /*程序结束*/
}
```

（1）为了能接收用户输入的整型数据，定义了两个实型变量height和weight。

（2）scanf函数只能接收输入的数据，不能显示信息，所以先使用printf函数输出一段信息提示。

（3）调用scanf格式输入函数，在函数参数中使用双引号将格式字符括起来，“%f”表示输入的是十进制的小数。在参数中的地址列表位置使用“&”符号表示变量的地址。

（4）变量height和weight得到用户输入的数据后，根据公式计算BMI值，再调用printf函数将BMI值输出。注意，printf函数使用的是变量的标识符，而不是变量地址。scanf函数使用的是变量地址，而不是变量的标识符。

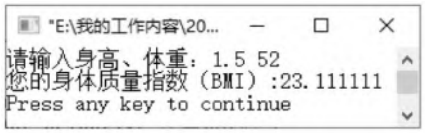


图5.8　使用scanf函数获取用户数据

运行程序，显示效果如图5.8所示。

说明

程序是怎样将输入的内容分别保存到两个指定变量中的呢？scanf函数使用空白字符分隔输入的数据，这些空白字符包括空格、换行和制表符（tab）。本例中使用了换行作为空白字符。

注意

使用scanf函数输入数据时，实际输入的变量格式必须与scanf函数参数中指定的变量格式、顺序一一对应。例如：

```
scanf("%d %c %f ",&a,&b,&c);
```

这里要求输入3个变量，格式分别是整型、字符型和浮点型，数据间采用空格进行分隔。用户通过控制台实际输入时，也必须按照这样的格式及顺序进行。如果scanf函数里使用的格式列表是这样的“%d#%c#%f”，那么在输入数据时，就需要使用“#”进行分隔。

相应地，scanf函数中也有附加格式，用于更为具体的格式说明，如表5.4所示。

表5.4　scanf函数的附加格式说明字符

附加字符	功能说明
l	用于输入长整型数据（如%ld、%lo、%lx、%lu）以及double型数据（如%lf、%le）
h	用于输入短整型数据（如%hd、%ho、%hx）
n	用于指定输入数据的字段宽度，多余的数据将被忽略
*	表示指定的输入项在读入后不赋给相应的变量

【例5.9】输入不同类型的数据（实例位置：资源包\TM\sl\05\09）

在本实例中，依次使用scanf函数的附加格式进行格式输入，对比输入前后的结果，观察附加格式的效果。

```
#include<stdio.h>
int main()
{
    long iLong;                                          /*长整型变量*/
    short iShort;                                        /*短整型变量*/
    int iNumber1=1;                                      /*整型变量，为其赋值 1*/
    int iNumber2=2;                                      /*整型变量，为其赋值 2*/
    char cChar[10];                                      /*定义字符数组变量*/

    printf("Enter the long integer\n");                  /*输出信息提示*/
    scanf("%ld",&iLong);                                 /*%ld，输入长整型数据*/
    printf("Enter the short integer\n");                 /*输出信息提示*/
    scanf("%hd",&iShort);                                /*%hd，输入短整型数据*/
    printf("Enter the number:\n");                       /*输出信息提示*/
    scanf("%d*%d",&iNumber1,&iNumber2);                  /*%d*%d，输入整型数据，忽略第 2 个输入的数
    printf("Enter the string but only show three character\n");  /*输出信息提示*/
    scanf("%3s",cChar);                                  /*%3s，输入字符串，只接收前 3 个字符

    printf("the long interger is: %ld\n",iLong);         /*显示长整型值*/
    printf("the short interger is: %hd\n",iShort);       /*显示短整型值*/
    printf("the Number1 is: %d\n",iNumber1);             /*显示整型 iNumber1 的值*/
    printf("the Number2 is: %d\n",iNumber2);             /*显示整型 iNumber2 的值*/
    printf("the three character are: %s\n",cChar);       /*显示字符串*/
    return 0;
}
```

（1）为了scanf函数能接收数据，需要在程序代码中先定义所使用的变量。为了演示不同格式字符的情况，定义的变量类型有长整型、短整型和字符数组。

（2）使用printf函数显示提示信息，提示待输入的数据为长整型，调用scanf函数使变量iLong得到用户输入的数据。在scanf函数的格式控制部分，使用格式字符%ld表示长整型。

（3）再使用printf函数显示提示信息，提示输入的数据为短整型。调用scanf函数时，使用格式字符%hd表示短整型。

（4）使用格式字符“*”的作用是表示指定的输入项在读入后不赋给相应的变量。具体分析如下：第一个“%d”是输入iNumber1变量，第二个“%d”是输入iNumber2变量，但是在第二个“%d”前有一个“*”附加格式说明字符，这样第二个输入的值被忽略，也就是说，iNumber2变量不保存输入的值。

（5）“%s”是用来表示字符串的格式字符，将一个整数n放入“%s”中间，可以指定数据的宽度。在程序中，scanf函数中指定的数据宽度为3，因此在输入字符串时只接收前3个字符。

（6）利用printf函数将获取的输入数据输出。

运行程序，显示效果如图5.9所示。

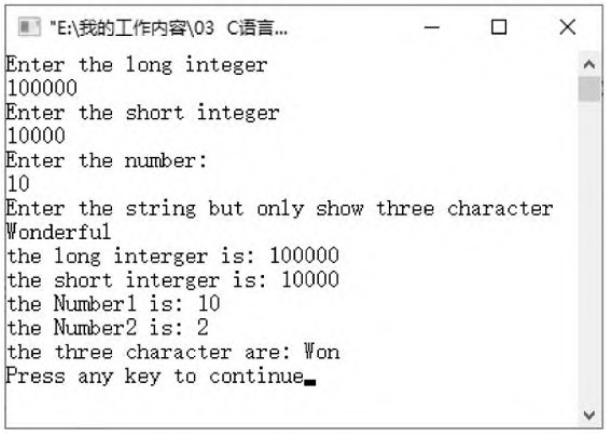


图5.9　在scanf函数中使用附加格式说明

编程训练（答案位置：资源包\TM\sl\05\编程训练\）

训练7：输出《拍手歌》　使用scanf函数输入数值，用printf函数输出《拍手歌》。例如，输入数值1，输出“你拍1，我拍1”。运行结果如下：

```
请输入你想拍几：
1
你拍1,我拍1
```

训练8：计算圆的周长和球的体积　使用scanf函数输入一个半径，计算对应圆的周长和对应球的体积。运行结果如下：

```
请输入半径：
2.5
周长是：15.70
体积是：49.06
```

5.6 顺序程序设计应用

本节将介绍几个顺序程序设计的实例，帮助读者巩固前面所讲的内容。

【例5.10】交换数值（实例位置：资源包\TM\sl\05\10）

在本实例中，利用scanf函数输入x，y两个数值。交换后调用printf函数输出。具体代码如下：

```
#include <stdio.h>                          /*包含头文件*/
int main()                                  /*主函数 main*/
{
    int x,y;                                /*定义变量*/
    printf("please enter two numbers:\n");  /*提示信息*/
    scanf("x=%d,y=%d",&x,&y);               /*输入 x,y 数据*/
    x=y-x;                                  /*交换 x,y 的值*/
    y=y-x;
    x=y+x;
    printf("x=%d,y=%d\n",x,y);              /*输出交换后的数据*/
    return 0;                               /*程序结束*/
}
```

（1）使用printf函数显示一串提示信息，提示输入两个数据，调用scanf函数使变量x，y得到用户输入的数据。

（2）代码第7~9行实现了x，y两个数的交换。最后利用printf函数将调换后的两个数输出。

运行程序，显示效果如图5.10所示。

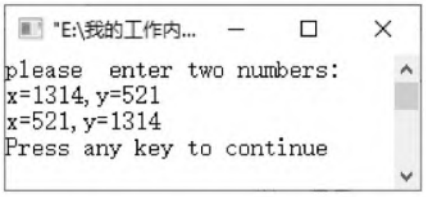


图5.10 交换数值

【例5.11】英文大小写字符互相转换（实例位置：资源包\TM\sl\05\11）

要实现英文大、小写字符的相互转换，就需要对C语言的ASCII码有所了解。将大写字符的ASCII码加上32，即可转换成小写字符的ASCII码。

```
#include<stdio.h>
int main()
{
    char cBig;                                  /*定义字符变量，表示大写字符*/
    char cSmall;                                /*定义字符变量，表示小写字符*/

    puts("Please enter capital character:");    /*输出提示信息*/
    cBig=getchar();                             /*得到用户输入的大写字符*/
    puts("Minuscule character is:");            /*输出提示信息*/
    cSmall=cBig+32;                             /*将大写字符转换成小写字符*/
    printf("%c\n",cSmall);                      /*输出小写字符*/
    return 0;                                   /*程序结束*/
}
```

（1）为了将大写字符转换为小写字符，要为其定义变量并进行保存。cBig表示要输入的大写字符变量，而cSmall表示要转换成的小写字符变量。

（2）输出信息，提示用户输入字符。因为得到一个输入字符即可，所以在此处使用getchar函数就可以满足程序的要求。

（3）大写字符与小写字符的ASCII码值相差32。例如，字符A的ASCII值为65，a的ASCII值为97。因此要将一个大写字符转换成小写字符，将大写字符的ASCII值加上32即可。

（4）字符变量cSmall得到转换的小写字符后，利用printf格式输出函数将字符输出，其中使用的格式字符为“%c”。

运行程序，显示效果如图5.11所示。

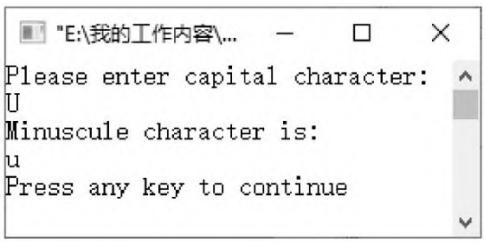


图5.11　将大写字符转换成小写字符

编程训练（答案位置：资源包\TM\sl\05\编程训练\）

训练9：显示菱形图案　使用printf函数的附加格式字符输出菱形图案，运行结果如下：

```
  *
* * *
  *
```

训练10：记录个人基本信息　使用scanf函数输入你的性别、年龄、体重（kg）和身高（m），运行结果如下：

```
请输入您的性别:W
请输入您的年龄:28
请输入您的身高:1.68
请输入您的体重:49.4
---您的基本信息----
性别：W
年龄：28
身高：1.68
体重：49.40
```

5.7　实践与练习

（答案位置：资源包\TM\sl\05\实践与练习\）

综合练习1：模拟缴纳电费　编写程序，先模拟输出如图5.12所示的电费账单，然后要求用户输入缴费金额（大于574元），输出当前账户电费余额，输出结果如图5.13所示。



图5.12　电费账单



图5.13　模拟账单效果

综合练习2：秘密电文　为了防止敌方获取我方机密，需要以数字代替我方文件中的字母，即用字母的ASCII码值替换字母。编写程序，输入字母后，输出该字母对应的ASCII码值。输出结果如下：

```
输入文件中字母：Z
☆ ☆ ☆ ☆ ☆ ☆ ☆
☆     替换为：90    ☆
☆ ☆ ☆ ☆ ☆ ☆ ☆
```

综合练习3：挑战10秒　某商场推出“挑战10秒”打折促销活动，编写程序，实现效果如图5.14所示。

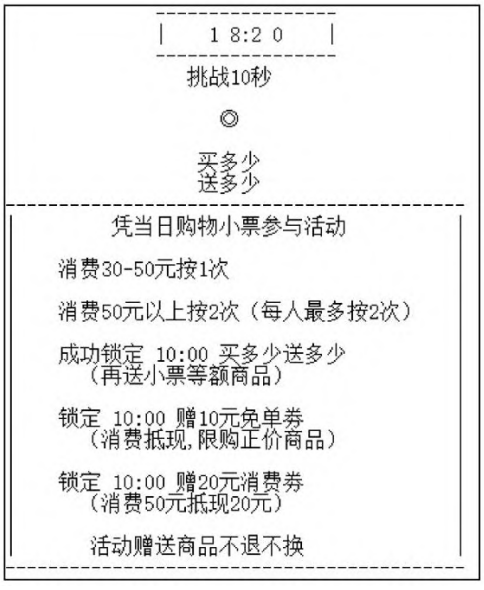


图5.14　挑战10秒促销活动

综合练习4：小写变大写　使用getchar函数输入一个小写字母，输出对应的大写字母。输出结果如下：

```
请输入一个小写字母：
q
&&&&&&&&&&&&&&
&  转换以后的字母：Q    &
&&&&&&&&&&&&&&
```

综合练习5：模拟用户登录　利用gets函数输入用户的账号和密码，用puts函数输出对应的用户账号和密码，输出结果如下：

```
请输入账号：
mingrikeji
您输入的账号为：
mingrikeji
请输入密码：
1234***456
您输入的密码为：
1234***456
≈≈≈≈≈≈≈≈≈≈≈≈≈≈
|    您的账号登录成功    |
≈≈≈≈≈≈≈≈≈≈≈≈≈≈
```

综合练习6：模拟手机充值　编写一个C语言程序，模拟手机充值信息提示。输出结果如下：

```
话费余额为：8 元
请输入充值金额:100
当前可用余额：108 元
```

综合练习7：给电影打分　《肖申克的救赎》是一部经典影片。编写程序，征集评分。打分时只能输入数字1~9，输出根据用户打分形成的星级（*）评价。输出结果如下：

```
请你为《肖申克的救赎》这部电影打分（只能输入数字1~9）：
8
你为《肖申克的救赎》电影的评价为：此处有8个*
```

综合练习8：能量兑换红包　图5.15为淘宝能量兑换成功的界面，参照该图，定义浮点型变量表示红包金额，提示“请输入您的能量兑换红包的金额”，输入完成后使用print函数输出变量。实现效果如图5.16所示。



图5.15　能量兑换红包

```
请输入您的能量兑换红包的金额：
23.05
  我的能量兑换了
    ¥23.05
感谢曾经努力的自己
```

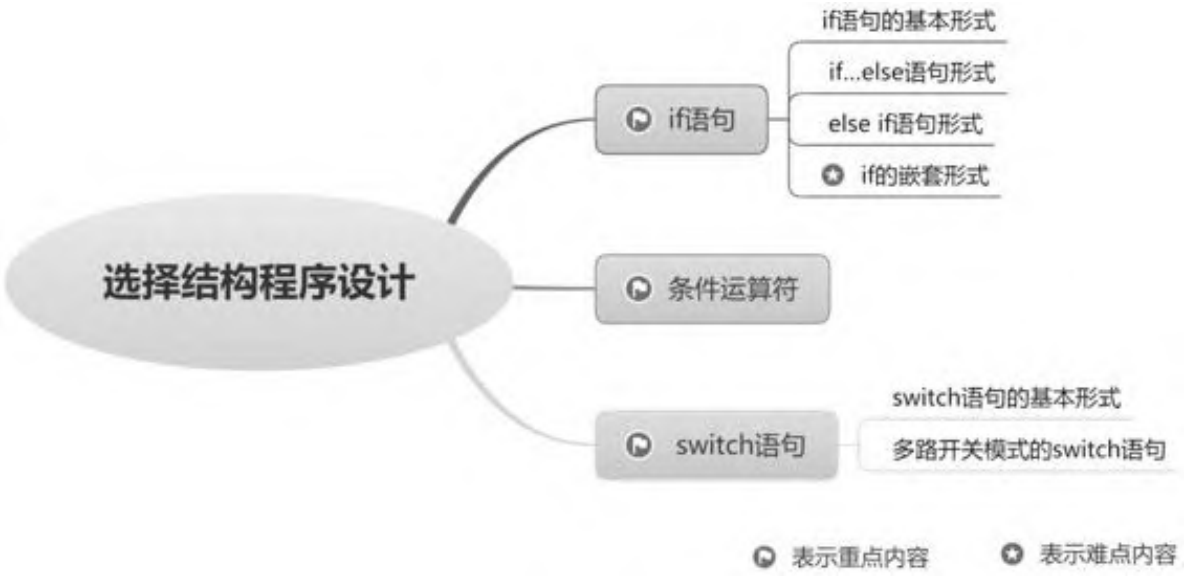
图5.16　实现效果

第6章
选择结构程序设计

顺序结构程序设计最简单，但通常无法解决生活中的选择性问题。选择结构程序设计需要用到一些条件判断语句，可实现的程序功能更加复杂，程序的逻辑性与灵活性也更加强大。

本章致力于使读者掌握使用if语句进行条件判断的方法，并掌握switch语句的使用方式。

本章的知识架构及重难点如下：



6.1 if语句



日常生活中，为了使交通畅通有序，一般会在路口设立交通信号灯。信号灯为绿色时，车辆可以行驶通过；信号灯为红色时，车辆就要停止行驶。可见，信号灯给出了信号，人们通过不同的信号进行判断，然后根据判断的结果进行相应的操作。

在C语言程序中，要想完成这样的判断操作，利用的就是if语句。if语句的功能就像路口的信号灯一样，通过判断不同的条件，决定是否进行操作。

程序开发中，会遇到许多类似于红绿灯这样的“决策”，可将其简化成对某个“条件”的检验，进而转换为判断某个表达式的值是真还是假，最后根据真假情况选择性地执行后续的操作。除了没有任何返回值的函数和返回无法判断真假的结构函数，几乎所有表达式的返回值都可以判断真假。

6.2 if语句的基本形式

if语句中，首先判断表达式的值，然后根据该值的真假情况决定后续程序流程。表达式的值不等于0，就表示为真；否则，就是假值。if语句有if、if…else和else if 3种形式，下面介绍每种形式的具体使用方式。

6.2.1 if语句形式



if语句的一般形式如下：

```
if(表达式)   语句
```

其中，括号中的表达式就是要进行判断的条件，语句则是要执行的对应操作。如果if判断括号中的表达式为真（非0），就执行后面的语句；如果为假（0），则不执行后面的语句。例如下面的代码：

```
if(iNum) printf("The truevalue");
```

上述代码中，如果变量iNum的值为真，则执行后面的printf输入语句；如果变量iNum的值为假，则不执行后面的printf语句。

用if语句解决红绿灯问题时，可以约定信号灯为1表示绿灯，则判断代码为：

```
if(iSignal==1) printf("the Signal Light is%d:",iSignal);
```

上述代码中，首先判断表达式“iSignal==1”是否成立，如果条件成立，那么判断结果是真值，执行后面的输出语句；如果条件不成立，那么判断结果为假值，不执行后面的输出语句。

前面两处示例代码中，if后的执行部分都只有一条语句。如果有两条语句，又该怎么办呢？这时可以使用大括号，使之成为语句块。例如：

```
if(iSignal==1)
{
    printf("the Signal Light is%d:\n",iSignal);
    printf("Cars can run");
}
```

将执行的语句都放在大括号中，这样当if语句判断条件为真时，语句块内的内容将会全部被执行。使用这种方式的好处是可以规范、清晰地表达出if语句所包含的范围。建议大家在使用if语句时，都使用大括号将执行语句包括在内。

误区警示

if语句的括号后一定不要加英文分号（;），否则将编译报错。例如：

```
if(i > 100);
{
    sum=i+5000;
    printf("%d\n",sum);
}
```

如果已经报错，可根据错误提示，找到差错并修改代码。

if语句的执行流程图如图6.1所示。

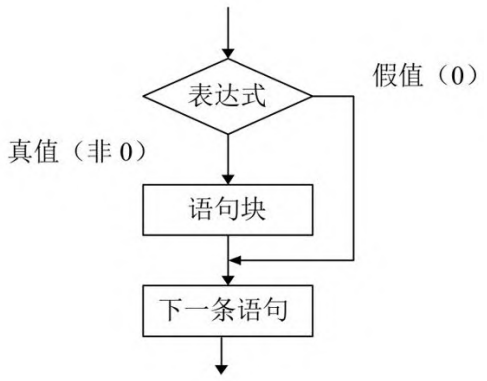


图6.1　if语句的执行流程图

误区警示

选择结构处理问题一定要把条件描述清楚，例如下面的语句是错误的。

```
if(i/6< >0) {...}
```

【例6.1】使用if语句模拟打卡签到（实例位置：资源包\TM\sl\06\01）

模拟此场景：一位职工早上上班打卡，她的工位号是13，密码是111，输入正确门禁卡会出现“谢谢，已签到”的字样。

在本实例中，为了模拟打卡签到，要使用if语句判断工位号和密码。如果工位号是13，同时密码是111，则通过printf函数提示“谢谢，已签到”。

```
#include<stdio.h>
int main()
{
    int loc,ser;                                    /*定义变量 loc 表示工位号，ser 表示密码*/
    printf("请输入工位号、密码：");
    scanf("%d,%d",&loc,&ser);                       /*输入工位号和密码*/
    if(loc==13&&ser==111)                           /*判断输入是否相同*/
        printf("谢谢，已签到\n");
    return 0;
}
```

（1）为了模拟打卡签到，要根据工位号和密码情况进行判断，这样就需要两个变量分别表示工位号和密码。在程序代码中，定义变量loc表示工位号，ser表示密码。

（2）输出提示信息，提醒用户输入loc、ser变量的值。此时用键盘输入“13，111”，表示工位号是13，密码是111。

（3）使用if语句判断，如果为真，则表示签到成功；如果为假，则表示签到不成功。因为输入的变量loc的值为13，ser的值是111，表达式loc==13&&ser==111的条件成立，因此判断的结果为真值，从而执行if语句后面大括号中的语句。

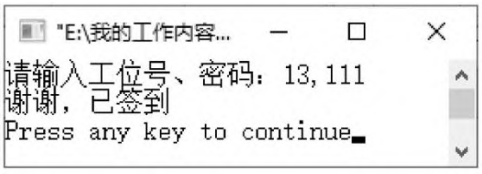


图6.2　使用if语句模拟打卡签到

运行程序，显示效果如图6.2所示。

if语句可以多次使用，使得程序能根据不同的分支条件给出相应的操作。例如在例6.1中，虽然使用if语句对输入的loc、ser变量进行了判断，但是只给出了判断正确，也就是loc等于13、ser等于111时执行的操作，

并没有给出判断错误时对应的操作。为了保证在用户输入错误的工位号、密码时，程序也能做出反应，需要再使用一次if语句。现在对例6.1进行完善。

【例6.2】完善模拟打卡签到（实例位置：资源包\TM\sl\06\02）

例6.1仅对打卡成功情况做出了相应操作，为进一步完善打卡签到失败时的操作，在程序中再添加一次if语句，对打卡签到失败的情况进行判断，并且在条件成立时给出相应操作。

```
#include<stdio.h>
int main()
{
    int loc,ser;                                    /*定义变量 loc 为工位号，ser 为密码*/
    printf("请输入工位号、密码：");
    scanf("%d,%d",&loc,&ser);                       /*输入工位号和密码*/
    if(loc==13&&ser==111)                           /*判断输入是否相同*/
        printf("谢谢，已签到\n");
    if(loc!=13&&ser!=111)                           /*判断输入是否不相同*/
        printf("对不起，工位号或密码有错误\n");
    return 0;
}
```

（1）在例6.1的基础上进行修改，完善程序的功能。在代码中添加一个if判断语句，用来表示当loc!=13&&ser!=111时要进行的相应操作。

（2）从程序的开始处来分析整个程序的运行过程。使用scanf函数输入数据，这次用户输入的是“12、123”，表示工位号是12，密码是123。

（3）程序继续执行，第一个if语句判断loc==13&&ser==111是否为真，如果判断的结果为真，则说明签到成功。因为loc的值是12，ser的值是123，所以判断的结果为假，不会执行第一个printf语句中的内容。

（4）接下来是新添加的if语句，在其中判断loc!=13&&ser!=111是否为真，如果判断成立为真，则表示签到不成功。因为输入的值为12、123，所以loc!=13&&ser!=111条件成立，执行第二个printf语句内容。

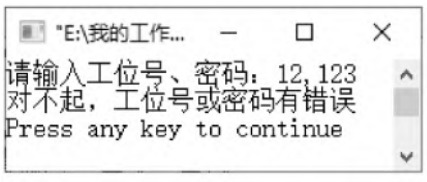


图6.3　完善if语句的使用

运行程序，显示效果如图6.3所示。

初学编程的人在程序中使用if语句时，常常会将下面的两个判断弄混：

```
if(value) {...}                                    /*判断变量值是否为真*/
if(value==0) {...}                                 /*判断表达式的值是否为真*/
```

这两行代码中都有value变量，value值虽然相同，但是判断的结果却不同。第1行代码判断的是value的值是否为真，第2行代码判断的是value==0这个表达式是否成立。假定其中value的值为0，那么在第一个if语句中，value值为0即说明判断的结果为假，所以不会执行if后的语句。在第二个if语句中，判断的是value是否等于0，因为设定value的值为0，所以表达式成立，那么判断的结果就为真，执行if后的语句。

6.2.2 if...else语句形式



if语句只给出了条件为真时要执行某个操作，并未给出条件为假时具体要干什么。如果想在条件为假时执行另外一段代码，需要利用if...else语句来完成，其一般形式如下：

```
if(表达式)
{
    语句块 1;
{
else
{
    语句块 2;
}
```

if...else语句的执行流程图如图6.4所示。

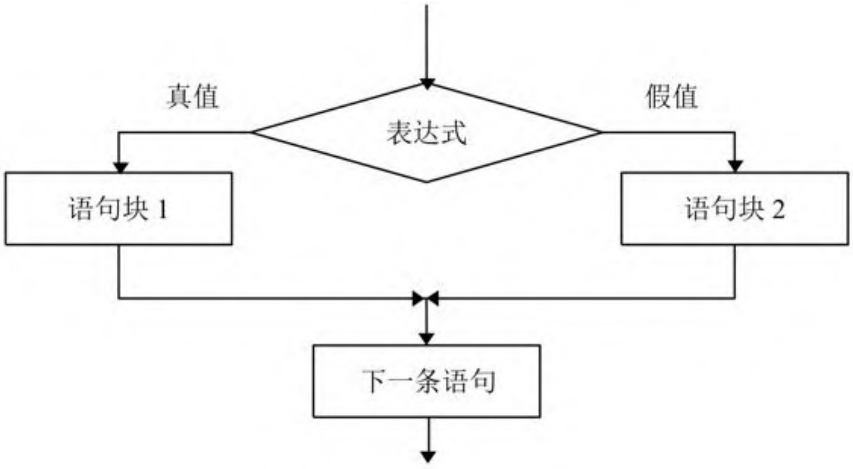


图6.4　if…else语句的执行流程图

在if后的括号中判断表达式的结果，如果判断的结果为真值，则执行if后的语句块内容；如果判断的结果为假值，则执行else后的语句块内容。也就是说，当if语句检验的条件为假时，就执行else后面的语句或者语句块。例如：

```
if(value)
{
    printf("the value is true");
}
else
{
    printf("the value is false");
}
```

注意

else语句必须跟在一个if语句的后面，不能独立使用。

【例6.3】条件判断真真假假（实例位置：资源包\TM\sl\06\03）

在本实例中，使用if…else语句判断用户输入的数值。输入的数字为0，表示条件为假；输入的数字为非0，表示条件为真。

```
#include<stdio.h>
int main()
{
    int iNumber;                                          /*定义变量*/
    printf("Enter a number\n");                           /*显示提示信息*/
    scanf("%d",&iNumber);                                 /*输入数字*/
    if(iNumber)                                           /*判断变量的值*/
    {
        printf("the value is true and the number is: %d\n",iNumber);    /*判断为真时执行输出*/
    }
    else
    {
        printf("the value is flase and the number is: %d\n",iNumber);   /*判断为假时执行输出*/
    }
    return 0;
}
```

（1）程序中定义变量iNumber用来保存用户输入的数据，然后通过if…else语句判断变量的值。

（2）用户输入数据的值为0，if语句判断iNumber变量，此时也就是判断输入的数值。因为0表示的是假，所以不会执行if后面紧跟着的语句块，而会执行else后面语句块中的操作，显示一条信息并将数值进行输出。

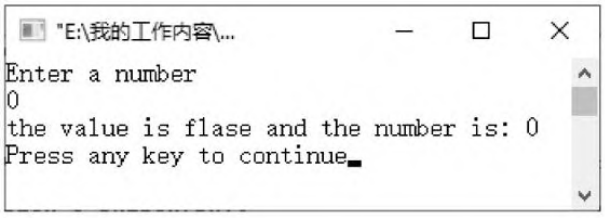


图6.5　使用if…else语句进行选择判断

（3）从程序的运行结果也可以看出，当if语句检验的条件为假时，就执行else后面的语句或者语句块。

运行程序，显示效果如图6.5所示。

if…else语句也可以用来判断表达式，根据表达式的结果选择不同的操作。

【例6.4】计算员工工资（实例位置：资源包\TM\sl\06\04）

某公司的薪酬设计如下：工龄在5年以上，工资＝基本工资＋全勤奖+2倍的全勤奖；否则，工资＝基本工资＋全勤奖。本实例利用if...else语句判断并计算工资，具体代码如下：

```
#include<stdio.h>
int main()
{
    int year,sal=5000,bon=300 ;                    /*定义变量表示年数、基本工资和全勤奖*/
    printf("请输入您的工龄：");
    scanf("%d",&year);                             /*输入工作年数*/
    if(year>5)                                     /*如果工作年数在 5 年以上*/
        printf("您的工资为：%d\n",sal+bon+2*bon);
    else                                           /*否则，工作年数在 5 年以下*/
        printf("您的工资为：%d\n",sal+bon);
    return 0;
}
```

（1）在程序运行过程中，利用printf函数先显示一条信息，提示用户输入一个数据，例如输入3。这个数据的数值由变量year保存。

（2）if语句判断表达式year>5的真假。如果判断的结果为真，说明工龄大于5，则执行if后的语句，输出“基本工资＋全勤奖+2倍的全勤奖”的值；如果判断的结果为假，说明工龄小于等于5，则执行else后的语句，输出“基本工资＋全勤奖”的值。因为year的值为3，所以year>5的关系表达式结果为假，这样执行的就是else后的语句，输出“基本工资＋全勤奖”的值。

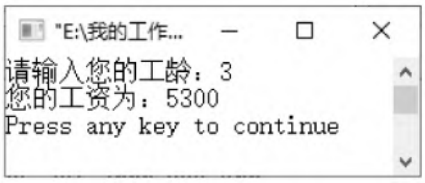


图6.6　计算员工工资

运行程序，显示效果如图6.6所示。

误区警示

在使用if语句比较浮点数时，不要使用浮点值，这样就会导致实际结果的偏差，因为浮点值属于近似值。

【例6.5】根据时间输出问候语（实例位置：资源包\TM\sl\06\05）

输入一个时间，当时间小于12时，输出“Good day”；当时间等于12时，输出“Good noon”；否则输出“Good evening”。本实例使用

if…else语句实现，具体代码如下：

```
#include<stdio.h>
int main()
{
    int time;                                   /*定义变量*/
    printf("请输入此时的时间：");
    scanf("%d",&time);                          /*输入时间*/
    if(time<12)                                 /*输入时间小于 12 时*/
        printf("Good day\n");                   /*判断结果为真时输出*/
    if(time==12)                                /*输入时间等于 12 时*/
        printf("Good noon\n");                  /*判断结果为真时输出*/
    else                                        /*输入时间大于 12 时*/
        printf("Good evening\n");               /*判断结果为真时输出*/
    return 0;
}
```

（1）程序运行时，先输出信息，提示用户输入一个时间。

（2）输入数字14，将其保存到变量time中。接下来使用if语句进行判断。

（3）第一个if语句判断time是否小于12，很明显判断结果为假，因此不会执行第一个if后语句块中的内容。

（4）第二个if语句判断time是否等于12，结果为假，因此不会执行第二个if后语句块中的内容。

（5）因为第二个if语句为假值，不执行第二个if语句的话，就会执行else后的语句块输出信息“Good evening”。

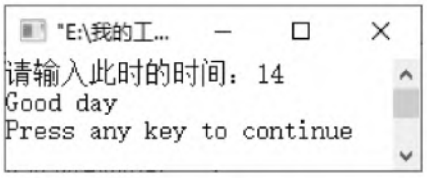


图6.7　根据时间输出问候语

运行程序，显示效果如图6.7所示。

注意

上面这个程序实际上是存在一些问题的，假如用户输入的数值为5，第一个if语句判断为真值，则会执行后面紧跟着的语句块。紧接

着，第二个if语句判断出time值不等于12，为假值，所以会执行else后的语句块，如图6.8所示。注意，else后的语句执行是我们不希望发生的，6.2.3节中会介绍解决这个问题的方法。

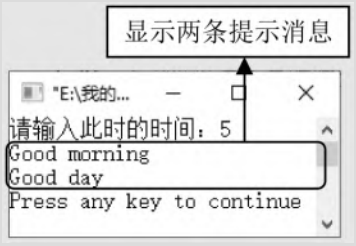


图6.8　使用if…else语句根据时间输出问候语可能出现的错误

6.2.3　else if语句形式



利用if和else关键字的组合，可以实现else if语句。该语句可对一系列互斥的条件进行检验，从而解决较复杂的判断和决策问题。else if语句的一般形式如下：

```
if(表达式 1)
{
    语句块 1
}
else if(表达式 2)
{
    语句块 2
}
else if(表达式 3)
{
    语句块 3
}
...
else if(表达式 m)
{
    语句块 m
}
else
{
    语句块 n
}
```

else if语句的执行流程图如图6.9所示。首先对if语句中的表达式1进行判断，如果结果为真值，则执行后面跟着的语句块1，然后跳过else if语句和else语句，如果结果为假，那么判断else if语句中的表达式2，如果表达式2为真值，那么执行语句块2而不会执行后面else if的判断或者else语句。当所有的判断都不成立，也就是都为假值时，执行else后的语句块n。

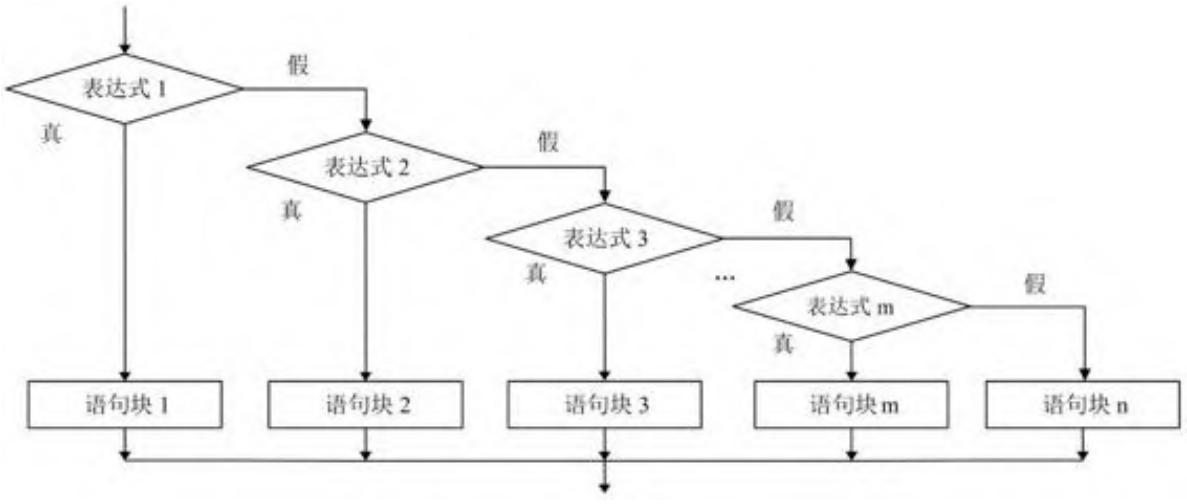


图6.9 else if语句的执行流程图

误区警示

使用多个else if语句时，要格外注意下方的复合语句（即语句块）部分，大括号一定要保证成对匹配。

【例6.6】计算分段函数的值（实例位置：资源包\TM\sl\06\06）

利用选择结构设计一个程序，使其能计算如下分段函数：

$$y=\begin{cases} x & (x<1) \\ 2x-1 & (1\leqslant x<10) \\ 3x-11 & (x\geqslant 10) \end{cases}$$

当输入x值时，计算显示y值。具体实现代码如下：

```
#include<stdio.h>
int main()
{
    int x,y;                          /*定义变量 x,y*/
    printf("请输入 x 的值：");
    scanf("%d",&x);                   /*输入 x 的数*/
    if(x<1)                           /*判断输入的 x 是否小于 1*/
    {
        y=x;                          /*如果为真，将 x 的值赋给 y*/
        printf("y=%d\n",y);           /*输出 y 的值*/
    }
    else if(x>=1&&x<10)               /*判断输入的 x 是否大于等于 1 且小于 10*/
    {
        y=2*x-1;                      /*如果为真，将 2*x-1 的值赋给 y*/
        printf("y=%d\n",y);           /*输出 y 的值*/
    }
    else                              /*x 为其他数值时*/
    {
        y=3*x-11;                     /*如果为真，将 3*x-11 的值赋给 y*/
        printf("y=%d\n",y);           /*输出 y 的值*/
    }
    return 0;
}
```

（1）程序中使用printf函数显示一条信息，提示用户输入x。

（2）这里假设输入的数字为5，变量x将输入的数值保存，用来执行后续判断。

（3）if语句首先判断x是否小于1，然后else if语句判断x是否大于等于1且小于10，如果都不满足，则会执行else处的语句。因为x的值为5，所以x>=1&&x<10关系表达式为真，执行相应else if处的语句块，输出提示信息。

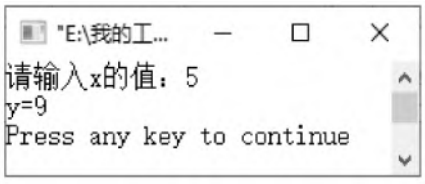


图6.10　使用else if语句求解函数

运行程序，显示效果如图6.10所示。

【例6.7】学生成绩等级划分（实例位置：资源包\TM\sl\06\07）

很多小学一、二年级的学生成绩并不显示分数，而是显示为优秀、良好、合格、不合格4个等级。各等级的划分标准如下。

❶ 优秀：大于等于90分。

❷ 良好：大于等于80分，小于90分。

❸ 合格：大于等于60分，小于80分。

❹ 不合格：小于60分。

根据控制台输入的成绩，输出与该成绩对应的等级。使用else if语句实现，具体代码如下：

```
#include<stdio.h>
int main()
{
    int score;                                  /*定义变量*/
    printf("请输入成绩：");
    scanf("%d",&score);                         /*输入成绩*/
    if(score>=90)                               /*成绩是否大于等于 90 分*/
        printf("优秀\n");                       /*如果为真，输出“优秀”*/
    else if(score>=80&&score<90)                /*成绩是否大于等于 80 分且小于 90 分*/
        printf("良好\n");                       /*如果为真，输出“良好”*/
    else if(score>=60&&score<80)                /*成绩是否大于等于 60 分且小于 80 分*/
        printf("合格\n");                       /*如果为真，输出“合格”*/
    else                                        /*成绩输入其他数值*/
        printf("不合格\n");                     /*如果为真，输出“不合格”*/
    return 0;
}
```

（1）程序中使用printf函数显示一条信息，提示用户进行输入成绩。这里假设输入的成绩是72，变量score将输入的数值保存，用来执行后续判断。

（2）if语句首先判断score是否大于等于90，然后使用else if语句判断score大于等于80且小于90的情况，再使用else if语句判断score大于等于60且小于80的情况，如果都不满足则会执行else处的语句。因为score的值为72，所以score>=60&&score<80关系表达式为真，执行第二个else if处的语句块，输出提示信息。

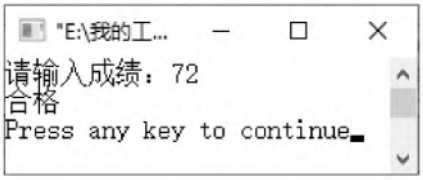


图6.11　使用else if语句判断成绩等级

运行程序，显示效果如图6.11所示。

编程训练（答案位置：资源包\TM\sl\06\编程训练\）

训练1：可乐还是咖啡　利用if语句判断用户输入的数值，输出对应的饮品。输入数字1代表喝CocaCola，输入数字2代表喝coffee。运行结果如下：

```
数字1代表喝CocaCola，数字2代表喝coffee，请选择：
```

```
2
您要喝的是coffee
```

训练2：模拟交通信号灯情况　“红灯停，绿灯行，黄灯等一等”是我们从小就熟知的安全通行口诀。当交通信号灯是红色时，停止通行；为绿灯时，可以通行；为黄灯时，需要等待。用代码模拟红绿灯的当前状态及通行情况，输出结果如下：

```
数字1表示红灯亮，数字2表示黄灯亮，数字3表示绿灯亮
请输入目前交通灯的状态：
3
目前交通灯是绿灯，可以通行
```

训练3：模拟抽奖游戏　某公司年会抽奖，抽奖细则如下。

❶ 抽中1，代表中一等奖，奖品是42寸彩电。

❷ 抽中2，代表中二等奖，奖品是光波炉。

❸ 抽中3，代表中三等奖，奖品是加湿器。

❹ 抽中4，代表中安慰奖，奖品是16GB的U盘。

编写代码，根据控制台输入的中奖等级，输出对应的奖品。输出结果如下：

```
请输入中奖等级：
2
奖品是光波炉
```

训练4：买彩票中大奖　编写代码模拟某彩民的梦想：如果买彩票中奖，就换辆豪车；如果没中奖，那就该干啥干啥。当然，中奖金额不同，要换的车子也不同。例如，具体的中奖设想如下。

❶ 超过500万元，换兰博基尼。

❷ 在500万~200万元，换卡宴。

❸ 在200万~50万元，换奔驰。

❹ 在0~50万元，换奥迪。

❺ 没中奖，该吃饭吃饭，该睡觉睡觉。

用if…else if…else语句输入中奖金额，输出要实现的梦想，运行结果如下：

```
请输入中奖金额：56
我中了56万，买奔驰
```

6.3 if的嵌套形式



if语句中还可以包含一个或多个if语句，此种情况称为if语句的嵌套。一般形式如下：

```
if(表达式 1)
{
    if(表达式 2)
    {
        语句块 1
    }
    else
    {
        语句块 2
    }
}
else
{
    if(表达式 3)
    {
        语句块 3
    }
    else
    {
        语句块 4
    }
}
```

使用if语句的嵌套形式，可将判断的条件进行细化，从而实现更复杂的判断和操作。

这就好比人们在生活中随时在进行的多重判断和选择。每天早上醒来，我们通常会想一下今天是周几，如果是周末，就是休息日；如果不是周末，就要上班。同时，休息日可以是星期六，也可以是星期日，星期六我们可以和朋友去逛街，星期日我们可以陪家人在家。工作日如果是星期一，就要开工作例会，其他时间则正常上班。

如何用if语句来实现上述判断呢？这里就要用到if嵌套语句。外层if语句的表达式1用于判断今天是否是周末，如果判断结果为真，则用一个嵌套if…else语句，通过表达式2判断今天是否是星期日，然后再判断今天是不是星期六。如果为星期六，就陪朋友逛街；如果为星期日，就陪家人在家。接下来返回外层，用else语句给出不是周末时的情况。这里同样要嵌套一个if…else语句，通过表达式3判断是否是周一，如果是就开例会，否则就正常上班。代码如下：

```
if(iDay>Friday)                    /*判断为周末的情况*/
{
    if(iDay==Saturday)             /*判断为星期六时的操作*/
    { }
    else                           /*为星期日时的操作*/
    { }
}
else                               /*不为周末的情况*/
{
    if(iDay==Monday)               /*判断为星期一时的操作*/
    { }
    else                           /*其他工作日时的操作*/
    { }
}
```

注意

if语句、if…else语句、else if语句可以根据需要互相嵌套。在复杂的嵌套中，一定要注意if与else的配对情况。else总是与其前面最近未配对的if进行配对。

前面曾经介绍过，if语句中如果要执行的操作只有一个语句，可以不写大括号。这里是否也可以呢？修改一下上面的代码，让其先判断是否为工作日，在工作日中只判断星期一的情况，然后再判断周末时的情况。代码如下：

```
if(iDay<=Friday)                    /*判断为工作日的情况*/
    if(iDay==Monday)                /*判断为星期一时的操作*/
    { }
else
    if(iDay==Saturday)              /*判断为星期六时的操作*/
    { }
    else
    { }
```

写这段代码时，我们的设想是先判断是否为工作日，如果是工作日，则判断是否为星期一；如果不是工作日，则判断是否是星期六，否则就是星期日。但因为else总是与其前面最近且未配对的if进行配对，所以粗体处的else不会与第一个if语句（if（iDay<Friday））配对，而是会与第二个if语句（if（iDay==Monday）配对，从而形成一个整体内嵌语句块，偏离了设计的初衷。此时的递进层次如下：

```
if(iDay<Friday)                     /*判断为工作日的情况*/
    if(iDay==Monday)                /*判断为星期一时的操作*/
    { }
    else
        if(iDay==Saturday)          /*判断为星期六时的操作*/
        { }
        else
        { }
```

内嵌 if 语句块

如果为最外层if语句后的语句块加上大括号，就可避免出现这种情况了。因此，建议大家在条件判断中即使只有一条语句，也要使用大括号进行层次划分。

【例6.8】购买粽子（实例位置：资源包\TM\sl\06\08）

粽子有甜的，有咸的。甜粽子有5元和10元的，咸粽子有4元和12元的。编写程序，根据输入的口味和钱数，判断并打印出能吃到哪种粽子。

其中，输入1代表选择甜粽子，其他数字代表选择咸粽子。用户输入“1”和“12”，输出“您可以吃到10元的甜粽子”。

使用if嵌套语句选择粽子种类，实现具体代码如下：

```
#include<stdio.h>
int main()
{
    int type,price;                                  /*定义变量 type 表示粽子口味，money 表示钱数*/
    printf("数字 1 表示甜粽子，否则就是咸粽子\n");
    printf("请输入粽子口味和可支付金额：");
    scanf("%d,%d",&type,&money);                     /*输入口味和手中的钱数*/
    if(type==1)                                      /*如果输入数字等于 1，表示选择甜粽子*/
    {
        if(money>=5&&money<10)                       /*判断输入的钱数，如果大于等于 5 且小于 10*/
            printf("您可以吃到 5 元的甜粽子\n");     /*输出信息*/
        else if(money>=10)                           /*判断输入的钱数，如果大于等于 10*/
            printf("您可以吃到 10 元的甜粽子\n");    /*输出信息*/
        else                                         /*输入的钱数不够*/
            printf("您不可以吃到甜粽子\n");          /*输出信息*/
    }
    else                                             /*如果输入其他数字，表示选择咸粽子*/
    {
        if(money>=4&&money<12)                       /*判断输入的钱数，如果大于等于 4 且小于 12*/
            printf("您可以吃到 4 元的咸粽子\n");     /*输出信息*/
        else if(money>=12)                           /*判断输入的钱数，如果大于等于 12*/
            printf("您可以吃到 12 元的咸粽子\n");    /*输出信息*/
        else                                         /*输入的钱数不够*/
            printf("您不可以吃到咸粽子\n");          /*输出信息*/
    }
    return 0;
}
```

（1）定义变量type用来保存后面用户输入的粽子口味，定义变量money用来保存用户输入的钱数。

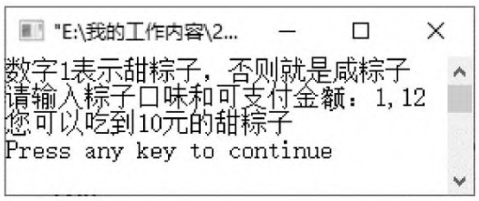


图6.12　使用if嵌套语句选择粽子

（2）在运行时，这里输入“1”，代表选择甜粽子。if语句首先判断表达式type==1，如果成立，则表示选择的是甜粽子，否则执行else表示咸粽子的部分。如果判断为真，则再利用if语句判断price>=5&&price<10的值，如果条件为真，则输出“您可以吃到5元的甜粽子”。else if语句判断price>=10的值，如果为真，则输出“您可以吃到

10元的甜粽子”，else语句表示的是钱数不够，则输出“您不可以吃到甜粽子”。

（3）因为type保存的数值为1，并且price大于10，所以输出信息是“您可以吃到10元的甜粽子”。

运行程序，显示效果如图6.12所示。

编程训练（答案位置：资源包\TM\sl\06\编程训练\）

训练5：销售等级划分　一个商品可能畅销，也可能滞销。根据输入的商品7天销售数据，输出该商品的销售等级。销售级别划分如下。

❶ 销售量大于1000，级别A。

❷ 销售量小于1000且大于等于500，级别B。

❸ 销售量小于500且大于等于300，级别C。

❹ 销售量小于300，级别D。

使用if嵌套语句，根据输入的7天销售量，输出对应的销售等级。输出结果如下：

```
请输入商品7天销量：1526
本商品的销售等级为A!!!
```

训练6：人生不同阶段　利用if嵌套语句输出人生的不同阶段。

❶ 年龄小于13岁，且小于2岁，输出他是婴儿。

❷ 年龄小于13岁，且年龄在2~4岁，输出他正在蹒跚学步。

❸ 年龄小于13岁，且年龄在4~13岁，输出他是儿童。

❹ 年龄大于13岁，且年龄在13~20岁，输出他是青少年。

❺ 年龄大于13岁，且年龄在20~65岁，输出他是成年人

❻ 年龄大于13岁，且年龄大于65岁，输出他是老年人。

定义一个变量age，根据age的值，判断处于人生的哪个阶段。运行结果如下：

```
请输入年龄：
28
他是成年人
```

6.4 条件运算符



首先来用if语句实现对两个数中最大值的求解，代码如下：

```
if(a>b)
    { max=a; }
else
    { max=b; }
```

上面的代码可以用条件运算符“? :”来简化：

```
max=(a>b)?a:b;
```

条件运算符可对一个表达式中值的真假情况进行检验，然后根据检验结果返回另外两个表达式中的一个。条件运算符的一般形式如下：

```
表达式1?表达式2:表达式3;
```

在运算中，首先对第一个表达式的值进行检验。如果值为真，则返回第二个表达式的结果值；如果值为假，则返回第三个表达式的结果值。例如，代码“max=（a>b）?a:b;”中，首先判断表达式a>b是否成立，成立则说明结果为真，否则为假。当为真时，将a的值赋给max变量；如果为假，则将b的值赋给max变量。

【例6.9】计算出租车费用（实例位置：资源包\TM\sl\06\09）

本实例要求计算出租车的收费情况。出租车行驶时，路径不足3千米，收取起步价6元；路程如果超过3千米，则会在起步价金额上加上超出部分的费用（按每千米2元收费）。使用条件运算符进行判断选择，具体代码如下：

```
#include<stdio.h>                                   /*包含头文件*/
int main()                                          /*主函数 main*/
{
    int jour,fee;                                   /*定义变量 jour 表示行驶路程，fee 表示车费*/
    printf("请输入出租车走的公里数:\n");              /*提示输入出租车行驶的路程*/
    scanf("%d",&jour);                              /*输入公里数*/
    fee=(jour<3)?6:6+(jour-3)*2;                    /*利用条件运算符计算费用*/
    printf("所花的费用是 %d 元\n",fee);               /*输出所花的费用*/
    return 0;                                       /*程序结束*/
}
```

（1）定义变量jour表示公里数，fee表示所花费用。

（2）根据提示信息，用户输入行驶公里数数据。假设用户输入12，表示公里数超过3。接下来使用条件运算符判断表达式jour<3是否成立，成立时为真，将“?”号后的值6赋给fee变量；否则将“6+（jour-3）*2”的值赋给fee变量。因为jour<3表达式不成立，所以fee的值为24。

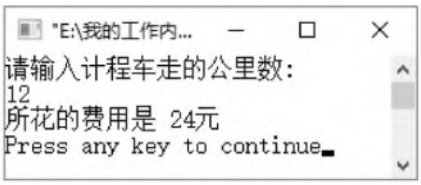


图6.13 计算出租车费用

运行程序，显示效果如图6.13所示。

编程训练（答案位置：资源包\TM\sl\06\编程训练\）

训练7：校园网收费标准 某校园网流量的收费标准是一元一天，如果购买时间超过30天，就按每天（包括30天）0.75元收费，否则就按原价收费。学生输入自己想购买的天数，输出所要花费的费用。输出结果如下：

```
请输入天数：
```

```
65
所花费用是：48.75
```

训练8：模拟京东自营配送收费　京东自营是京东商城的一大特色，通过在全国建有多个仓库，实现就近发货和快速到货。模拟京东自营的配送收费，满99元即包邮，不满99元加收6元快递费。利用条件运算符编写程序，运行结果如下：

```
您共计消费：
34
您实际需要付40.0元
```

6.5　switch语句

在实际问题中，常需要用到多分支的选择。当然，使用嵌套的if语句也可以实现多分支选择，但如果分支较多，就会使得嵌套的if语句层数过多，程序冗余，可读性也不好。C语言中可以使用switch语句直接处理多分支选择的情况，将程序代码的可读性提高。

6.5.1　switch语句的基本形式



switch语句是多分支选择语句。一般形式如下：

```
switch(表达式)
{
    case 情况 1:
        语句块 1;
    case 情况 2:
        语句块 2;
    …
    case 情况 n:
        语句块 n;
    default:
        默认语句块;
}
```

switch语句的执行流程图如图6.14所示。

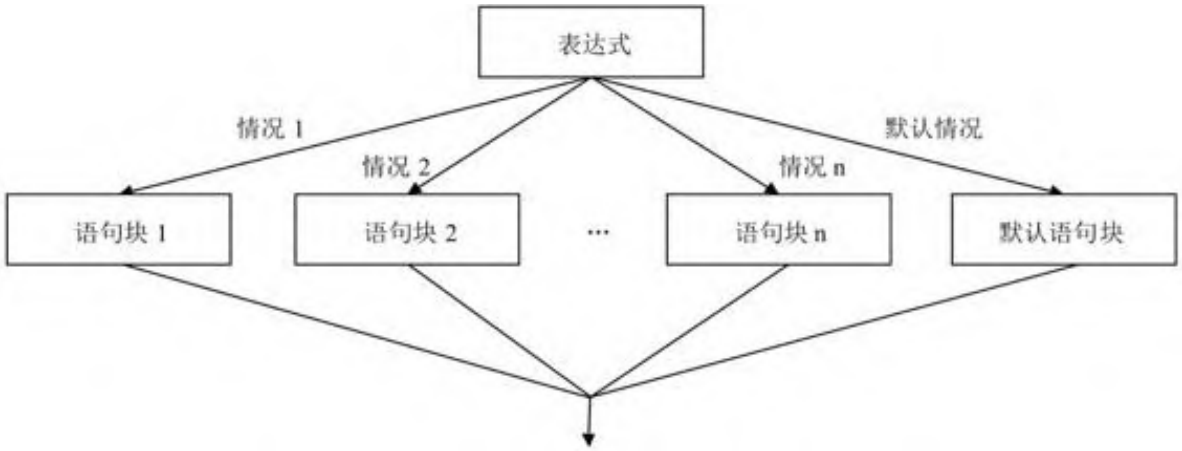


图6.14　switch多分支选择语句流程图

switch后面的表达式就是要进行判断的条件，然后使用case关键字表示检验条件符合的各种情况，其后的各语句块是相应的操作。其中还有一个default关键字，作用是如果没有符合条件的情况，那么执行default后的默认情况语句。

说明

switch语句检验的条件必须是一个整型表达式，这意味着其中也可以包含运算符和函数调用。而case语句检验的值必须是整型常量，即常量表达式或者常量运算。

下面通过示例代码分析一下switch语句的使用方法。

```
switch(selection)
{
    case 1:
        printf("Processing Receivables\n");
        break;
    case 2:
        printf("Processing Payables\n");
        break;
    case 3:
        printf("Quitting\n");
        break;
    default:
        printf("Error\n");
        break;
}
```

首先switch判断selection变量的值，然后利用case语句检验selection值的不同情况。假设selection的值为2，那么执行case为2时的情况，执行后跳出switch语句。如果selection的值不是case中所列出的情况，那么执行default中的语句。在每一个case或default语句后都有一个break关键字。break语句用于跳出switch结构，不再执行下面的代码。

注意

switch语句中，如果所有case后面的值都不能匹配switch语句的条件，就执行default语句后面的代码。其中，任意两个case语句都不能使用相同的常量值；每个switch结构只能有一个default语句，而且default可以省略。

【例6.10】计算商品折扣金额（实例位置：资源包\TM\sl\06\10）

某大型商超为答谢新老顾客，当累计消费金额达到一定数额时，顾客可享受不同的折扣。

❶ 累计未满500元，当前消费不享受任何折扣优惠。

❷ 累计满500元，当前消费可享9折优惠。

❸ 累计满1000元，当前消费可享8折优惠。

❹ 累计满2000元，当前消费可享7折优惠。

❺ 累计满3000元，当前消费可享6折优惠。

❻ 累计3000元以上，当前消费可享5折优惠。

根据顾客购物小票上的消费金额，在控制台上输出该顾客将会享受的折扣与打扣后需要支付的金额。在本实例中，要求使用switch语句来判断打折后的金额。具体代码如下：

```
#include<stdio.h>
int main()
{
    float money=1026,m;                    /*定义变量 money 为购物金额，m 为打折之后的费用*/
    float rebate=0.0f;                     /*定义折扣变量*/
        int grade=(int)money/500;          /*计算折扣等级*/
        switch(grade)                      /*根据等级计算打折后的金额*/
        {
        case 0:                            /*未满 500 元*/
            rebate=0.0f;                   /*折扣为 0*/
            m=money;                       /*原价购买*/
            break;                         /*跳出*/
        case 1:                                                  /*刚好满 500 元*/
            rebate=0.9f;                                         /*折扣为 9 折*/
            m=money*rebate;                                      /*计算折扣之后的费用*/
            break;                                               /*跳出*/
        case 2:                                                  /*满 1000 元*/
            rebate=0.8f;                                         /*折扣为 8 折*/
            m=money*rebate;                                      /*计算折扣之后的费用*/
            break;                                               /*跳出*/
        case 4:                                                  /*满 2000 元*/
            rebate=0.7f;                                         /*折扣为 7 折*/
            m=money*rebate;                                      /*计算折扣之后的费用*/
            break;                                               /*跳出*/
        case 6:                                                  /*满 3000 元*/
            rebate=0.6f;                                         /*折扣为 6 折*/
            m=money*rebate;                                      /*计算折扣之后的费用*/
            break;                                               /*跳出*/
        default:
            rebate=0.5f;                                         /*折扣为 5 折*/
            m=money*rebate;                                      /*计算折扣之后的费用*/
            break;
        }
        printf("原价是：%f\n 顾客将享受的折扣:%f\n",money,rebate);    /*输出原价和对应折扣*/
        printf("打扣后需支付的金额:%f\n",m);                         /*输出折扣后所花费用*/
        return 0;
}
```

（1）定义变量grade用来保存折扣等级。计算方法为“折扣等级＝购物金额/500元”。

（2）使用switch语句判断整型变量grade，其中使用case关键字检验可能出现的级别情况，并且在每个case语句的最后使用break语句跳出。如果没有符合的情况，则会执行default默认语句。

注意

在case语句表示的条件后有一个冒号“:”，在编写程序时不要忘记。

运行程序，显示效果如图6.15所示。

switch语句中，每个case情况内都使用了break语句。如果不使用break语句，会出现什么情况呢？break的作用是跳出switch语句，因此如果本例中没有break语句说明，程序后面的内容将会都执行。为了验证猜测是否正确，将上面程序中的break注释掉，运行程序，显示结果如图6.16所示。

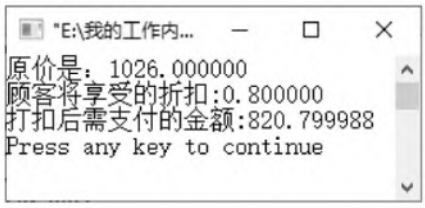


图6.15　使用switch语句判断折扣等级

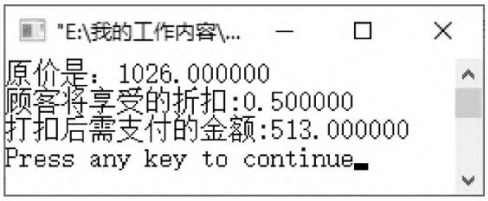


图6.16　不添加break的情况

从运行结果可以看出，当去掉break语句后，将计算default下的语句，很明显是错误的。因此，break语句在case语句中是不能缺少的。

【例6.11】日程安排（实例位置：资源包\TM\sl\06\11）

如果是周末就是休息日，如果不是周末就要上班。同时，休息日可以是星期六，也可以是星期日，星期六我们可以和朋友去逛街，星期日我们可以陪家人在家。用switch语句来模拟此场景。

```
#include<stdio.h>
int main()
{
    int iDay=0;                                          /*定义变量 iDay 表示输入的日期*/
    printf("enter a day of week to get course:\n");      /*提示信息*/
    scanf("%d",&iDay);                                   /*输入日期*/
    switch(iDay)
    {
    case 1:                                              /*iDay 的值为 1 时*/
        printf("Have a meeting in the company\n");
        break;
    case 6:                                              /*iDay 的值为 6 时*/
        printf("Go shopping with friends\n");
        break;
    case 7:                                              /*iDay 的值为 7 时*/
        printf("At home with families\n");
        break;
    default:                                             /*iDay 的值为其他情况时*/
        printf("Working with partner\n");
        break;
    }
    return 0;
}
```

在程序中，使用switch…case进行检验时，不要忘记case检验的条件只能是常量或者常量表达式，而不能对变量进行检验。

运行程序，显示效果如图6.17所示。

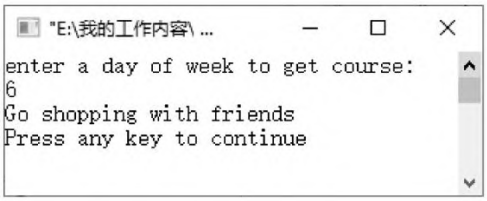


图6.17　日程安排程序

误区警示

初学switch语句时，很容易认为break是switch语句的一部分，是switch的必要条件。但是并不是这样，break可以不与switch匹配。

6.5.2　多路开关模式的switch语句



将某个case语句后的语句块及break去掉之后，可以设计出多路开关模式的switch语句。例如：

```
switch(表达式)
{
    case 1:
        语句 1
        break;
    case 2:
    case 3:
        语句 2
        break;
    …
    default:
        默认语句
        break;
}
```

可以看到，如果在case 2后不使用break语句，那么符合case 2检验与符合case 3检验时的输出效果是一样的。也就是说，多路开关模式可使得多种检验条件执行同一个操作。

【例6.12】判断一年四季情况（实例位置：资源包\TM\sl\06\12）

在本实例中，要求使用switch语句判断输入的月份属于哪个季节。已知3、4、5月是春季，6、7、8月为夏季，9、10、11月为秋季，12、1、2月为冬季。

```
#include<stdio.h>
int main()
{
    int month;                                  /*定义变量 month 表示月份*/
    printf("please enter a month:\n");
    scanf("%d",&month);                         /*输入月份*/
    switch(month)                               /*根据月份判断季节*/
    {
        /*多路开关模式*/
        case 3:
        case 4:
        case 5:
            printf("%d is spring\n",month);     /*3,4,5 月是春季*/
            break;
        /*多路开关模式*/
        case 6:
        case 7:
        case 8:
            printf("%d is summer\n",month);     /*6,7,8 月是夏季*/
            break;
        /*多路开关模式*/
        case 9:
        case 10:
        case 11:
            printf("%d is autumn\n",month);     /*9,10,11 月是秋季*/
            break;
        /*多路开关模式*/
        case 12:
        case 1:
        case 2:
            printf("%d is winter\n",month);     /*12,1,2 月是冬季*/
            break;
        default:
            printf("error!!!\n");               /*无此月份*/
    }
    return 0;                                   /*程序结束*/
}
```

本例程序中使用多路开关模式，使得检测month的值为3、4、5这3种情况时，都会输出相同的结果。同时，利用default语句对不符合月份的数字输入，提示错误。

运行程序，显示效果如图6.18所示。

编程训练（答案位置：资源包\TM\sl\06\编程训练\）

训练9：猜猜哪个灯能亮　已知一个灯泡并联3个开关，分别为开关1、开关2、开关3；另有一串彩灯，串联开关4；开关5和开关6并联一个

白炽灯和节能灯。电路如图6.19所示。问：随机按下开关，哪个灯能亮。输出结果如下：

```
输入开关编号：
4
彩灯亮
```

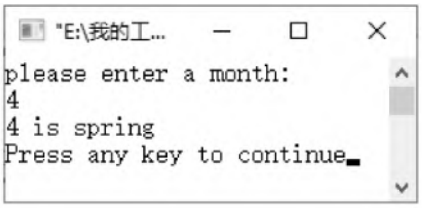


图6.18　使用多路开关模式判断四季情况

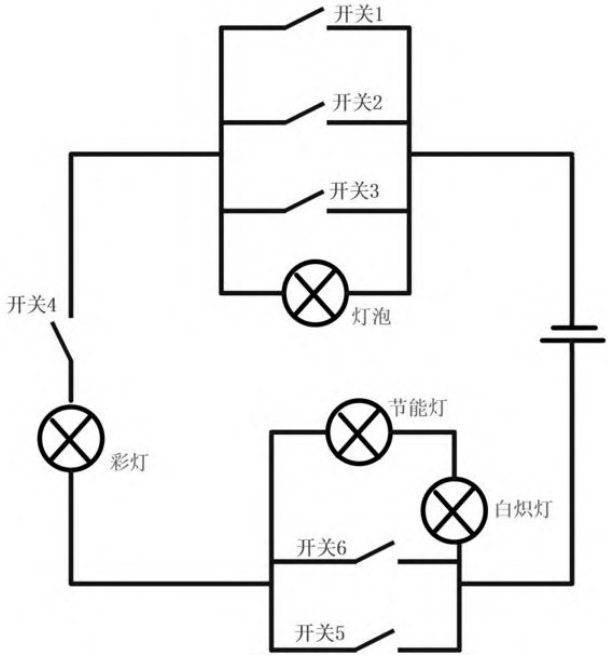


图6.19　猜猜哪个灯能亮

训练10：高考填报志愿　高考是人生中最重要的考试。不同的分数可选择不同的大学。根据模拟分数线，输出可以报考哪些大学。

❶ 671分：北京大学。

❷ 661分：清华大学。

❸ 668分：复旦大学。

❹ 664分：上海交通大学。

❺ 默认情况：其他大学。

利用switch语句编写程序，最终运行结果如下：

```
请输入您的高考成绩：664
选择上海交通大学
```

6.6 if...else语句和switch语句的区别



if…else语句和switch语句都可根据不同的情况检验条件，并做出相应的判断。那么if…else语句和switch语句有什么区别呢？下面从两者的语法和效率方面进行比较。

1．语法的比较

使用时，if需要配合else关键字，switch需要配合case关键字；if语句是先对条件进行判断，而switch语句是后进行判断。

2．效率的比较

if…else结构对少量的检验，判断速度比较快。但是随着检验深度的增长，会逐渐变慢，并且也不容易进行后续的条件添加和扩充。

switch结构中，除default默认情况外，对其他每一项case的检验速度都是相同的。default默认情况比其他情况都快。

当需要判定的情况较少时，使用if…else结构比使用switch结构检验速度快。也就是说，如果分支在3个或者4个以下，用if…else结构比较好，否则应选择switch结构。

【例6.13】模拟拨打10086服务热线（实例位置：资源包\TM\sl\06\13）

给10086移动客服中心打电话，如果不是移动电话的用户，则提示“暂时无法提供服务”；如果是移动的用户，则提示“查询话费请拨1，人工服务请拨0”。用户输入1之后，显示话费余额。

使用if…else语句和switch语句模拟此场景，代码如下：

```
#include<stdio.h>
int main()
{
    int telephone,num;                                  /*定义两个变量，分别表示电话号码和选择的服务*/
    printf("请输入您要拨打的电话：");
    scanf("%d",&telephone);                             /*输入电话号码*/
    if(telephone==10086)                                /*判断是否为移动用户*/
    {
        /*是移动用户，则进行以下操作*/
        printf("你好，欢迎致电中国移动，客服将为您提供以下服务：查询话费请拨 1，人工服务请拨 0\n");
        scanf("%d",&num);                               /*输入一个数字*/
        switch(num)                                     /*判断输入的数字*/
        {
        case 1:                                         /*数字 1 表示查询话费*/
            printf("您的话费余额为 9.89 元\n");
            break;
        case 0:                                         /*数字 0 表示人工服务*/
            printf("您好，欢迎使用移动客服业务，为了保证通话质量，您的通话将会被录音，谢谢合作。\n
            break;
        default:                                        /*输入其他数字*/
            printf("对不起，系统没有此业务。\n");       /*提示信息，无此业务*/
        }
    }
    else                                                /*不是移动用户*/
    {
        printf("暂时无法提供服务");                     /*提示信息，暂时无法提供服务*/
    }
        return 0;
}
```

首先，判断用户拨打的是否是10086，只有两种可能，检验的条件少，所以使用if…else语句更为方便。接下来判断用户选择的服务，就要

对2~3种甚至更多种情况进行检验，由于检验条件比较多，所以使用switch语句比较合适。其中，case语句用来判断用户输入的服务选项，default语句用于在输入数字不符合条件时提示“暂时无法提供服务”。

运行程序，显示效果如图6.20所示。

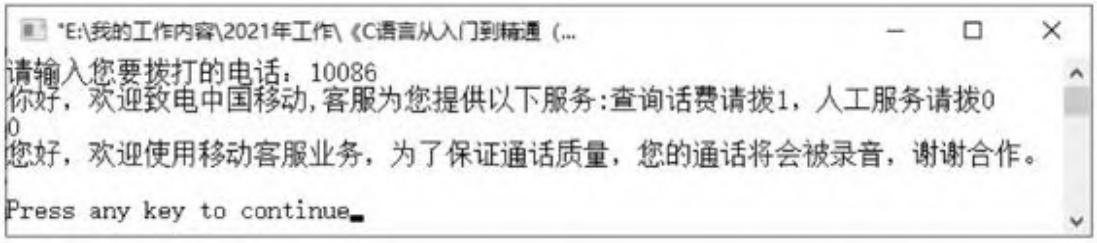


图6.20　if…else语句和switch语句的综合应用

编程训练（答案位置：资源包\TM\sl\06\编程训练\）

训练11：猜价格游戏　猜商品的价格，并提示猜的结果偏大还是偏小。本实例中，利用if...else语句实现，然后利用printf函数显示相应的信息。输出结果如下：

```
请输入猜的价格：
25
您猜小了
```

训练12：考试成绩等级　要求按照考试成绩的等级输出分数段，其中要使用switch语句来判断分数的情况。分级详细信息如下：

A：90~100分。

B：80~89分。

C：70~79分。

D：60~69分。

F：小于60分。

利用switch语句编写，最终运行结果如下：

```
请输入要查看的等级：
B
分数段是:80~89
```

6.7 实践与练习

（答案位置：资源包\TM\sl\06\实践与练习\）

综合练习1：微信小程序，我该玩哪个 微信小程序由于不需要下载App，深受大家青睐。利用if语句判断输入的数值，输出对应的微信小程序游戏。输出结果如下：

```
1代表跳一跳，2代表好友画我，3代表头脑王者，请选择：
3
您现在选择的是3
所以您要玩"头脑王者"游戏
```

综合练习2：模拟自动售货机 在自动售货机上购买商品时，通常是通过按键选择商品。例如，按键1代表选择巧克力，按键2代表选择小面包，按键3代表选择可口可乐。利用switch语句模拟实现此场景，运行结果如下：

```
**********************
*     可选择的按键：     *
*      1.巧克力          *
*      2.小面包          *
*      3.可口可乐        *
**********************
从 1~3 中选择按键：
3
您选择了可口可乐
```

综合练习3：输出玫瑰花语 女生都喜欢玫瑰花，因为每种玫瑰花都代表不同的含义。例如，红玫瑰代表“我爱你、热恋，希望与你永恒相爱”；白玫瑰代表“纯洁、谦卑、尊敬，我们的爱情是纯洁的”；粉玫瑰代表“初恋，喜欢你那灿烂的笑容，年轻漂亮”；蓝玫瑰代表“憨厚、善

良”。选择不同的玫瑰，输出对应的花语，运行结果如图6.21和图6.22所示。



图6.21　输出白玫瑰花语

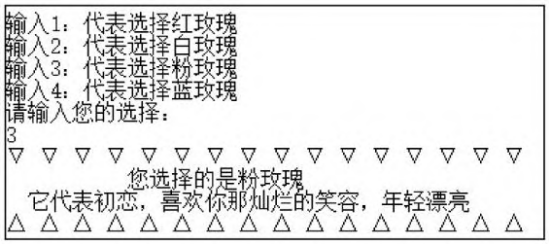


图6.22　输出粉玫瑰花语

综合练习4：寻找最大值　设计一个程序，通过键盘输入任意3个整数，输出其中最大的数。运行结果如下：

```
请输入3个数字：
87
9
65
最大值是 a：87
```

综合练习5：自助服务　某商店支持自助服务，支付方式有3种，数字1代表网络支付，数字2代表银行卡支付，数字3代表现金支付，顾客可根据输入的数字选择一种支付方式。输出结果如下：

```
本店支持以下自助支付方式：
1、网络支付
2、银行卡支付
3、现金支付
请输入数字选择支付方式
^_^ ^_^ ^_^ ^_^ ^_^ ^_^ ^_^
```

```
1
请用微信或支付宝扫描对应二维码以完成支付
```

综合练习6：输出美团外卖待支付金额　利用条件运算符输出美团外卖订单。订单金额满15元可免费配送，否则就要加上5元的配送费。运行程序，结果如图6.23和图6.24所示。

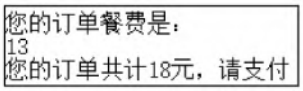


图6.23　订单金额小于15元

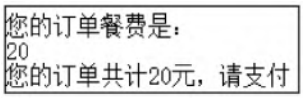


图6.24　订单金额大于15元

综合练习7：空气质量指数　空气质量指数AQI（Air Quality Index）是定量描述空气质量状况的指数，分为6级，对应空气质量的6个类别，其数值越大，说明空气污染状况越严重。表6.1为空气质量等级、指数范围及相应的空气质量状况对应表。

表6.1　空气质量等级、指数范围及相应的空气质量状况对应表

空气质量等级	空气质量指数	空气质量状况	建　议
1	0～50	优秀	可多参加户外活动，呼吸新鲜空气
2	51～100	良好	除少数对某些污染物特别容易过敏的人群外，其他人群可以正常进行室外活动
3	101～150	轻度污染	敏感人群需减少体力消耗较大的户外活动
4	151～200	中度污染	敏感人群应尽量减少外出，一般人群适当减少户外活动
5	201～300	重度污染	敏感人群应停止户外活动，一般人群尽量减少户外活动
6	>300	严重污染	除由特殊需要的人群外，尽量不要留在室外

编写程序，输入所在城市的空气质量指数，输出空气质量状况和出行建议，效果如图6.25和图6.26所示。

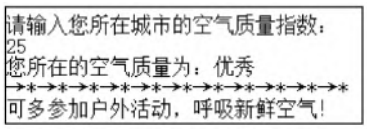


图6.25　空气质量指数为25

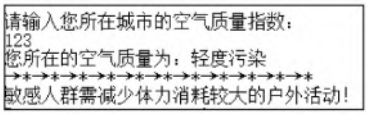


图6.26　空气质量指数为123

综合练习8：判断是否为酒后驾车　国家质量监督检验检疫局发布的《车辆驾驶人员血液、呼气酒精含量阈值与检验》中规定：车辆驾驶人员100ml血液中的酒精含量小于20mg不构成饮酒驾驶行为；大于等于20mg且小于80mg为饮酒驾车；大于等于80mg为醉酒驾车。编写程序，输入血液中的酒精含量，判断是否为酒驾。运行结果如下：

```
为了您和他人的安全，严禁酒后驾车
请输入每 100 毫升血液的酒精含量：
90

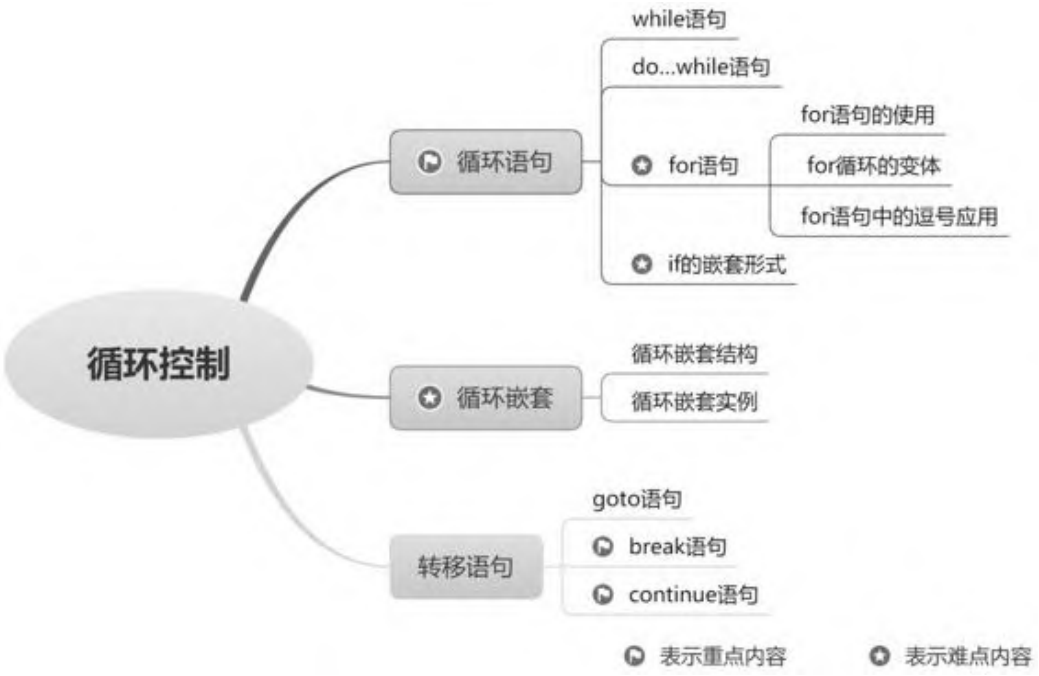
已经达到醉酒驾驶标准，千万不要开车！
```

第7章
循环控制

生活中总会有许多简单而重复的工作，为完成这些重复性工作，需要花费很多时间。使用循环语句来处理程序开发中简单、重复性的工作是最好不过的了。

本章致力于使读者了解while、do…while和for 3种循环结构的特点，以及转移语句的相关内容。

本章的知识架构及重难点如下：



7.1 循环语句



程序在运行时可以通过判断、检验条件做出选择，帮我们解决许多问题。此处，程序还必须能够重复，也就是能反复执行一段指令，直到满足某个条件为止。例如，要计算一个公司的月消费额，就要将所有的消费金额依次相加。如果要计算全年的消费总额，还需要将1~12月的消费金额依次相加。

这种重复的过程就称为循环。C语言中有3种循环语句，即while、do…while和for循环语句。循环结构是结构化程序设计的基本结构之一，因此熟练掌握循环结构是程序设计的基本要求。

7.2 while语句



使用while语句可以解决当某个条件满足时反复执行某个循环体的问题。其一般形式如下：

```
while(表达式)
{
    语句块(循环体)
}
```

while语句的执行流程图如图7.1所示。

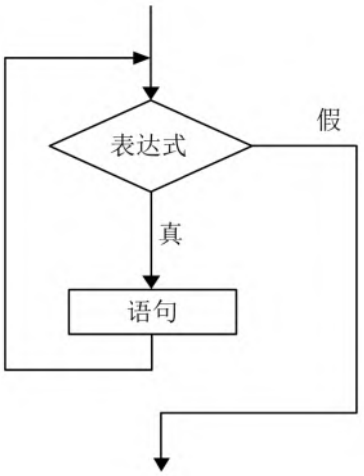


图7.1　while语句的执行流程图

while语句首先检验一个条件，也就是括号中的表达式。当条件为真时，就执行紧跟其后的循环体语句或者语句块。每执行一遍循环，程序都将回到while语句处，重新检验条件是否满足。如果一开始条件就不满足，则跳过循环体中的语句，直接执行后面的程序代码。如果第一次检验时条件满足，那么在第一次或其后的循环过程中，必须得有使条件为假的操作，否则循环将无法终止。

说明

无法终止的循环常被称为死循环或者无限循环。

例如下面的代码：

```
while(iSum<100)
{
      iSum+=1;
}
printf("iSum 为：%d",iSum);
```

在这段代码中，while语句首先判断iSum变量是否小于常量100，如果小于100，为真，那么执行紧跟其后的循环语句块；如果不小于100，为假，那么跳过循环语句块中的内容直接执行printf语句。在循环语句块

中，可以看到对其中的变量进行加1的运算，这里的加1运算就是循环结构中使条件为假的操作，也就是使得iSum不小于100，否则程序会一直循环下去。

误区警示

初学者很容易在while语句后添加英文分号（;），从而导致编译报错。一定要注意，if、if...else、while语句的判断表达式后是没有分号的。

【例7.1】计算n~100的累加和（while版）**（实例位置：资源包\TM\sl\07\01）**

用户输入一个值，从这个值开始，依次与之后的连续若干个自然数相加，当该自然数大于100时结束，并输出这若干个自然数的和。这里用到while循环，代码如下：

```
#include<stdio.h>
int main()
{
    int n,sum=0;                          /*定义变量 n 表示输入的数据，sum 表示累加和*/
    printf("请输入一个自然数:");
    scanf("%d",&n);                       /*输入自然数 n*/
    while(n<=100)                         /*使用 while 循环，当 n 小于等于 100 时*/
    {
        sum=sum+n;                        /*进行累加，并将累加和保存在 sum 中*/
        n++;                              /*n 自增运算，表示后续一个自然数*/
    }
    printf("结果为：%d\n",sum);           /*输出 n～100 的累加和*/
    return 0;
}
```

（1）在程序代码中，因为要计算n~100的累加结果，所以要定义两个变量，n表示用户输入的数据，sum表示累加的和。

（2）使用while语句判断n是否小于等于100，如果条件为真，则执行其后语句块中的内容；如果条件为假，则跳过语句块执行后面的内容。

当用户输入90时，表示n的初始值为90，判断的条件为真，因此执行累加操作。

（3）在while语句块中，总和sum等于先前计算的总和结果加上现在n表示的数字，完成累加操作，n++通过自身加1操作表示下一个自然数。语句块执行结束后，while会再次判断新的n值。也就是说，“n++;”语句可以使得循环停止。

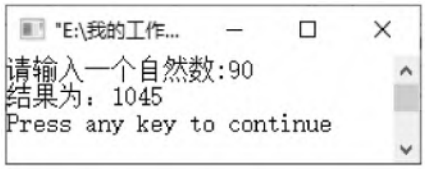


图7.2　计算n~100的累加和

（4）当n大于100时，循环操作结束，将结果sum输出。

运行程序，显示效果如图7.2所示。

误区警示

使用while语句时，很容易多执行一次或少执行一次循环，这类错误被称为“差一错误”。例如，把“number <= 20”写作“number < 20”，就会少执行一次循环，因此一定要给出正确的循环条件。

【例7.2】细菌的繁殖数量（while循环实现）**（实例位置：资源包\TM\sl\07\02）**

生物实验室做单细胞细菌繁殖实验。一代菌落中只有一个细菌，二代菌落中分裂成两个细菌，三代菌落中分裂成4个细菌，以此类推，每代细菌数量都会成倍数增长。计算第十二代菌落中的细菌数量。使用while循环语句实现，具体代码如下：

```c
#include<stdio.h>
int main()
{
    int x=1;                                    /*定义变量 x，表示菌落的代数*/
    int num=1;                                  /*定义变量 num，表示细菌数量*/
    while(x<=12)                                /*检验条件，代数小于等于 12*/
    {
        num*=2;                                 /*细菌成倍数增长*/
        x++;                                    /*代数增加*/
    }
    printf("第十二代菌落中的细菌数量是%d\n",num);   /*将结果输出*/
    return 0;
}
```

（1）在程序代码中，因为要计算1~12代的细菌成倍繁殖结果，所以要定义两个变量，x表示代数，num表示计算的细菌数量。

（2）使用while语句判断x是否小于等于12，如果条件为真，则执行其后语句块中的内容；如果条件为假，则跳过语句块执行后面的内容。初始x值为1，判断的条件为真，因此执行语句块。

（3）在语句块中，num等于先前计算的结果乘以2，完成细菌数量成倍操作，x++表示代数自身加1操作。语句块执行结束后，while会再次判断新的x值。也就是说，“x++;”语句可以使循环停止。

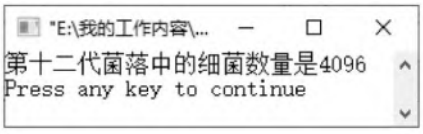


图7.3　计算细菌的繁殖数量

（4）当x大于12时，循环操作结束，将结果num输出。

运行程序，显示效果如图7.3所示。

编程训练（答案位置：资源包\TM\sl\07\编程训练\）

训练1：农夫卖瓜　编写程序，模拟农夫卖西瓜。农夫一共有1020个西瓜，第一天卖掉一半多2个，第二天卖掉剩下的一半多2个，如此循环下去，需要卖几天才能卖完。输出结果如下：（提示：剩余的数量＝原来数量/2-2，且每次循环，天数都要自增加1，直到西瓜的数量为0，跳出循环，输出天数）

```
这些西瓜，一共卖8天
```

训练2：打印100次名字　利用while循环，输出100个自己的名字。

7.3　do...while语句



在有些情况下，不论条件是否满足，循环过程必须执行至少一次，这时可以采用do…while语句。do…while语句的特点就是先执行循环体语句块中的内容，然后再判断循环条件是否成立。其一般形式如下：

```
do
{
    语句块(循环体)
} while(表达式);
```

do…while语句的执行流程图如图7.4所示。

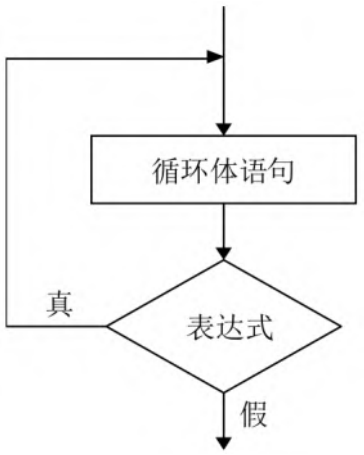


图7.4　do…while语句的执行流程图

do…while语句是这样执行的，首先执行一次循环体语句中的内容，然后判断表达式，当表达式的值为真时，返回重新执行循环体语句，然后再次判断表达式，直到表达式的值为假，此时循环结束。

说明

while语句和do…while语句的区别在于：while语句在每次循环之前检验条件，do…while语句在每次循环之后检验条件。这也可以从两种循环结构的代码上看出来，while结构中的while语句出现在循环体的前面，do…while结构中的while语句出现在循环体的后面。

例如下面的代码：

```
do
{
    iNumber++;
} while(iNumber<100);
```

在上面的代码中，首先执行iNumber++的操作，也就是说，不管iNumber是否小于100，都会执行一次循环体中的内容。然后判断while后括号中的内容，如果iNumber小于100，则再次执行循环语句块中的内容；条件为假时，执行后续的程序操作。

注意

在使用do…while语句时，条件表达式要放在while关键字后面的括号中，最后必须加上一个分号（;），这里和while语句不同，初学者一定要格外注意。

【例7.3】计算n~100的累加和（do…while版）**（实例位置：资源包\TM\sl\07\03）**

在例7.1中，计算n~100所有数字的累加方法使用的是while语句，在本实例中使用do…while语句实现相同的功能。在程序运行过程中，虽然两者的结果是相同的，但是要了解其中操作的区别。

```
#include<stdio.h>
int main()
{
    int n,sum=0;                          /*定义变量 n 表示输入的数据，sum 表示累加和*/
    printf("请输入一个自然数:");
    scanf("%d",&n);                       /*输入数据 n*/
    do                                    /*使用 do...while 循环*/
    {
        sum=sum+n;                        /*进行累加*/
        n++;                              /*增加数字*/
    } while(n<=100);                      /*while 循环条件*/
    printf("结果为：%d\n",sum);           /*将结果输出*/
    return 0;
}
```

（1）在程序中，同样定义n表示用户输入的数字，而sum表示计算的总和。

（2）do关键字之后是循环语句，在语句块中进行累加操作，并对n变量进行自加操作。语句块下方是while语句检验条件，如果检验为真，则继续执行上面的语句块操作；为假时，程序执行下面的代码。

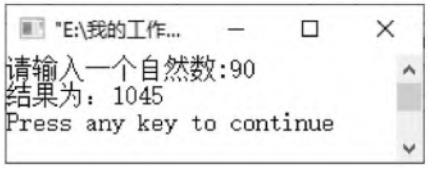


图7.5　计算n~100的累加和

（3）在循环操作完成之后，将结果输出。

运行程序，显示效果如图7.5所示。

编程训练（答案位置：资源包\TM\sl\07\编程训练\）

训练3：验证高斯的猜想　用do...while语句验证高斯的猜想。10岁的高斯就很快算出了1~100之和，他找到了算术级数的对称性，然后利用算术结合律，把数目一对一对地凑在一起，巧妙地算出了1~100之和。用do...while计算一下1~100之和是否是高斯所计算的结果5050。输出结果如下：

```
1~100的和是：5050
```

训练4：猴子摘桃　猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上将第一天剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第10天早上想再吃时，发现只剩下一个桃子了。编写程序，求解猴子第一天共摘了多少个桃子。利用do...while计算一下，输出结果如下：

```
一共有1534个桃子
```

7.4　for语句

C语言中，使用for语句也可以控制一个循环，并且在每次循环时修改循环变量。在循环语句中，for语句的应用最为灵活，可以用于循环次数确定的情况，还可以用于循环次数不确定但给出了循环结束条件的情况。下面将对for语句的循环结构进行详细的介绍。

7.4.1　for循环语句



for语句的一般形式如下：

```
for(表达式 1;表达式 2;表达式 3;)
{
    语句块(循环体)
}
```

每条for语句包含3个用分号隔开的表达式。这3个表达式用一对圆括号括起来，其后紧跟着循环语句或语句块。当执行到for语句时，程序首先计算第1个表达式的值，接着计算第2个表达式的值。如果第2个表达式的值为真，程序就执行循环体的内容，并计算第3个表达式；然后检验第

2个表达式，执行循环；如此反复，直到第2个表达式的值为假，退出循环。

for语句的执行流程如图7.6所示。其执行过程如下。

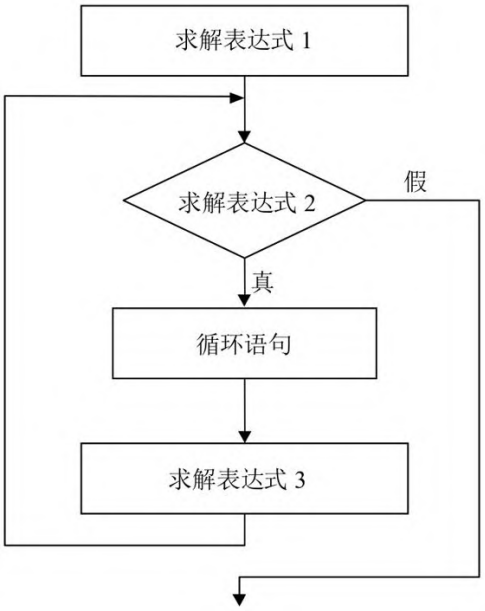


图7.6　for语句的执行流程图

（1）求解表达式1。

（2）求解表达式2，若其值为真，则执行for语句中的循环语句块，然后执行步骤（3）；若为假，则结束循环，转到步骤（5）。

（3）求解表达式3。

（4）回到步骤（2），继续执行。

（5）循环结束，执行for语句下面的一个语句。

for语句的形式也可以简单表示为：

```
for(循环变量赋初值;循环条件;循环变量改变)
{
    语句块
}
```

例如，实现一个循环输出操作：

```
for(i=1;i<100;i++)
{
    printf("the i is:%d",i);
}
```

在上面的代码中，表达式“i=1”用于对循环变量i进行赋值操作；表达式“i<100”用于判断循环条件是否为真，因为i的初值为1，小于100，所以执行语句块中的内容；表达式“i++”用于在每次循环结束后，对循环变量进行自增操作。一次循环完成后，会再次判断表达式“i<100”的状态，为真时，继续执行语句块；为假时，循环结束，执行后面的程序代码。

误区警示

使用for语句时，常常犯的错误是将for语句括号内的表达式用逗号隔开。例如：

```
for(i=1,i<100,i++) {...}
```

这句代码就是错误的。切记，3个表达式之间应用分号隔开。

【例7.4】计算小球反弹高度（**实例位置：资源包\TM\sl\07\04**）

一个球从80米高度自由落下，每次落地后反弹的高度为原高度的一半，计算小球第6次反弹的高度。在本实例中，要求使用for循环语句计算小球反弹高度。具体代码如下：

```
#include<stdio.h>
int main()
{
    int high=80;                          /*定义变量 high，表示高度*/
    int i;                                /*定义变量 i，作为循环变量*/
    for(i=0;i<6;i++)                      /*使用 for 语句循环*/
    {
        high/=2;                          /*每次反弹高度*/
        printf("当前高度是 %d\n",high);    /*输出当前反弹高度*/
    }
    printf("第 6 次反弹高度 %d\n",high);   /*输出第 6 次反弹高度*/
    return 0;                             /*程序结束*/
}
```

（1）在程序代码中，定义变量high。在for语句中先对i赋值，然后判断i<6的条件是否为真，再根据判断的结果选择是否执行循环语句。

（2）在循环语句中计算每次反弹高度，即用当前高度除以2，再用printf输出当前反弹高度。

（3）在for语句外，用printf函数输出第6次反弹高度。

运行程序，显示效果如图7.7所示。

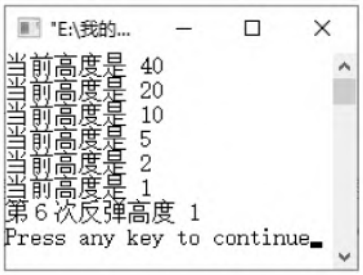


图7.7　使用for语句显示小球反弹高度

for循环的执行过程也可以使用while循环来表示：

```
表达式 1;
while(表达式 2)
{
    语句块
    表达式 3;
}
```

上面就是使用while循环表示for循环的一般形式，其中的3个表达式对应着for语句括号中的3个表达式。下面通过一个实例来比较一下这两种

操作。

【例7.5】计算n~100的累加和（for版）**（实例位置：资源包\TM\sl\07\05）**

在本实例中，先使用for语句实现一个n~100循环累加的操作，再使用while语句实现相同的功能。注意观察实例中for语句中的表达式与while语句中的表达式所对应的位置。

```
#include<stdio.h>
int main()
{
    int n,sum=0;                          /*定义变量 n 表示输入的数据，sum 表示累加和*/
    int i,n1,sum1=0;                      /*定义变量 n1 表示输入的数据，sum1 表示累加和*/
    printf("请输入一个自然数:");
    scanf("%d",&n1);                      /*输入数据 n1*/
    for(i=n1;i<=100;i++)                  /*使用 for 循环计算累加和*/
    {
        sum1=sum1+i;                      /*进行累加*/
    }
    printf("结果为：%d\n",sum1);          /*将结果输出*/

    printf("请输入一个自然数:");
    scanf("%d",&n);                       /*输入数据 n*/
    while(n<=100)                         /*使用 while 循环计算累加和*/
    {
        sum=sum+n;                        /*进行累加*/
        n++;                              /*循环变量自增*/
    }
    printf("结果为：%d\n",sum);           /*将结果输出*/
    return 0;
}
```

（1）定义变量n、n1表示用户输入的数字，变量sum表示计算的累加和，变量i用作for循环语句的循环变量。

（2）使用for语句执行循环操作，括号中表达式“i=n”为循环变量进行赋值。表达式“i<=100”判断条件，条件为真，执行for循环体中的内容；条件为假，不进行循环操作。

（3）在for循环语句块中进行累加运算，然后执行for括号中的第3个表达式“i++”，对循环变量进行自增操作。循环操作后，将保存有计算结果的变量sum进行输出。

（4）在使用while语句之前要恢复变量的值。再次输入变量n1，就相当于for语句中第一个表达式的作用，为变量设置初值。然后在while括号中的表达式“n<=100”与for语句中第二个表达式相对应。最后n++自加操作与for语句括号中的最后一个表达式相对应。

运行程序，显示效果如图7.8所示。

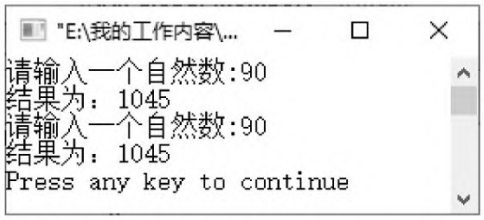


图7.8　使用while语句模仿for语句的一般形式

7.4.2　for循环的变体



通过上面的学习可知，for语句的一般形式中有3个表达式。这3个表达式都是必不可少的，还是可以根据情况进行省略呢？接下来对不同情况进行讲解。

1．for语句中省略表达式1

表达式1的作用是对循环变量设置初值，如果省略了表达式1，就会影响对表达式2（即循环条件）的判断。为保证程序正常运行，应在for语句之前先给循环变量赋值。例如：

```
for(;iNumber<10;iNumber++)
```

注意

省略表达式1时，其后的分号不能省略，要格外注意。

【例7.6】求n!（实例位置：资源包\TM\sl\07\06）

在本实例中，根据用户输入的数据计算其阶乘，并且将for语句中第一个表达式省略。具体代码如下：

```
#include<stdio.h>
int main()
{
    int n,i=1,result=1;                          /*定义变量，为变量 i, result 赋初始值*/
    printf("请输入 n 的值:\n");                  /*提示用户输入数据*/
    scanf("%d",&n);                              /*输入数据*/
    for(;i<=n;i++)                               /*for 循环，省略表达式 1*/
    {
        result*=i;                               /*计算阶乘*/
    }
    printf("%d!=%d\n",n,result);                 /*输出结果*/
    return 0;                                    /*程序结束*/
}
```

在代码中可以看到，在定义i变量时直接为其赋了初值，这样在使用for语句循环时就不用再为i赋初值，从而可以省略表达式1。

运行程序，显示效果如图7.9所示。

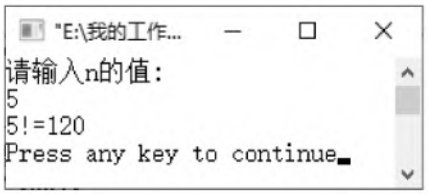


图7.9　省略for语句中的表达式1

2．for语句中省略表达式2

表达式2给出的是循环条件，如果表达式2省略，则无法判断循环条件，也即默认表达式2始终为真，因此循环将无终止地进行下去。例如：

```
for(iCount=1; ;iCount++)                         /*缺少循环结束条件，陷入死循环*/
{
    sum=sum+iCount;
}
```

括号中，表达式“iCount=1”为赋值表达式，第二个表达式是空缺的，这样就相当于使用了如下while语句：

```
iCount=1;
while(1)
{
    sum=sum+iCount;
    iCount++;
}
```

可见，for语句中表达式2是不能省略的。

3．for语句中省略表达式3

表达式3用于改变循环变量，可以省略，但此时程序设计人员应在循环体内增加类似功能的语句，以保证循环能正常结束，否则程序也会无终止地循环下去。例如：

```
for(iCount=1;iCount<50;)                    /*for 循环，省略表达式 3*/
{
    sum=sum+iCount;
    iCount++;                               /*此处语句可代替表达式 3 的功能*/
}
```

7.4.3　for循环中的逗号应用



在for语句中，表达式1和表达式3处除了可以使用简单的表达式，还可以使用逗号表达式，即包含一个以上的简单表达式，中间用逗号间隔。例如，在表达式1处为变量iCount和iSum设置初始值：

```
for(iSum=0,iCount=1;iCount<100;iCount++)
{
    iSum=iSum+iCount;
}
```

或者在表达式 3 处执行循环变量自加操作两次：

```
for(iCount=1;iCount<100;iCount++,iCount++)
{
    iSum=iSum+iCount;
}
```

也就是说，表达式1和表达式3都可以是逗号表达式。在逗号表达式内按照自左至右顺序求解，整个逗号表达式的值为其最右边的表达式的值。例如：

```
for(iCount=1;iCount<100;iCount++,iCount++)
```

就相当于：

```
for(iCount=1;iCount<100;iCount=iCount+2)
```

【例7.7】计算1~100所有偶数的累加结果（**实例位置：资源包\TM\sl\07\07**）

在本实例中，为变量赋初值的操作都放在for语句中，并且对循环变量进行两次自加操作，这样所求出的结果就是所有偶数的和。

```
#include<stdio.h>
int main()
{
    int iCount,iSum;                                    /*定义变量*/
    /*在 for 循环中，为变量赋值，对循环变量进行两次自增运算*/
    for(iSum=0,iCount=0;iCount<=100;iCount++,iCount++)
    {
        iSum=iSum+iCount;                               /*进行累加计算*/
    }
    printf("the result is:%d\n",iSum);                  /*输出结果*/
    return 0;
}
```

在程序代码中，在for语句中对变量iSum、iCount进行初始化赋值。每次循环语句执行完后进行两次iCount++操作，最后将结果输出。

运行程序，显示效果如图7.10所示。

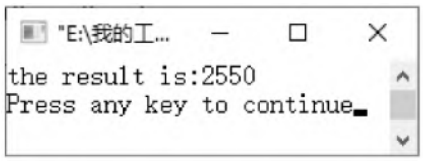


图7.10　计算1~100所有偶数的累加结果

误区警示

浮点值的误差是不可避免的。如果在for循环的条件表达式中使用浮点值，那么将导致数值错误。因此，建议开发者不要使用浮点类型定义for语句的循环变量。

编程训练（答案位置：资源包\TM\sl\07\编程训练\）

训练5：数学计算　用for语句求解1-3+5-7+9-…-99+10的结果，输出结果如下：

```
sum=51
```

训练6：模拟用公积金买房子　使用公积金买房，贷款的利息点比商业贷款低很多，能够为买房者省下很大一笔钱。模拟用公积金分期付款买房子，计算每个月需要还的月供金额，输出结果如下：

```
请输入你想要买房子的总价格：670000
房子的总价是：670000.0
首付370000元之后还剩300000.0元
将所剩的300000.0元进行分30年分期付款：
从我买房子开始，接下来的30年每个月需要还3500.12元钱
```

7.5　3种循环语句的比较



前面介绍了3种可以执行循环操作的语句。一般情况下，这3种循环之间可以相互替换。下面是对这3种循环语句在不同情况下的比较。

- ☑ while和do…while循环只在while后面指定循环条件，在循环体中应包含使循环趋于结束的语句（如i++或者i=i+1等）。for循环可以在表达式3中包含使循环趋于结束的操作。因此，for语句的功能更强，凡用while循环能完成的功能，用for循环都能实现。
- ☑ 用while和do…while循环时，循环变量初始化的操作应在while和do…while语句之前完成；而for语句可以在表达式1中实现循环变量的初始化。
- ☑ while循环、do…while循环和for循环都可以用break语句跳出循环，用continue语句结束本次循环（break和continue语句将在7.7节中进行介绍）。

7.6 循环嵌套

一个循环体内还可以包含一个或多个其他循环结构，这称之为循环的嵌套。如果嵌套的循环有多层，这就是多层循环。不管在什么语言中，关于循环嵌套的概念都是一样的。

7.6.1 循环嵌套的结构



while循环、do…while循环和for循环之间可以互相嵌套。例如，下面几种嵌套方式都是正确的。

1．while循环中嵌套while循环

```
while(表达式)
{
    语句块
    while(表达式)
    {
        语句块
    }
}
```

2．do...while循环中嵌套do...while循环

```
do
{
    语句块
    do
    {
        语句块
    }
    while(表达式);
}
while(表达式);
```

3．for循环中嵌套for循环

```
for(表达式;表达式;表达式)
{
    语句块
    for(表达式;表达式;表达式)
    {
        语句块
    }
}
```

4．do...while循环中嵌套while循环

```
do
{
    语句块
    while(表达式);
    {
        语句块
    }
}
while(表达式);
```

5．do...while循环中嵌套for循环

```
do
{
    语句块
    for(表达式;表达式;表达式)
    {
        语句块
    }
}
while(表达式);
```

类似的嵌套形式还有很多种，在此不一一列举。读者只要将每种循环结构的方式把握好，就可以正确写出一个循环嵌套。

7.6.2 循环嵌套实例



本节通过两个实例讲解，了解一下循环嵌套的使用方法。

【例7.8】百钱买百鸡（实例位置：资源包\TM\sl\07\08）

中国古典《算经》中有一道著名的百钱买鸡问题：鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一；一百钱买鸡，问翁母雏各几只。利用循环嵌套输出结果，具体实现代码如下：

```
#include<stdio.h>
int main()
{
    int x,y,z;                                          /*定义变量 x 为鸡翁，y 为鸡母，z 为鸡雏
    for(x=0;x<20;x++)                                   /*for 循环，最多可以买 20 个鸡翁*/
    {   for(y=0;y<33;y++)                               /*循环嵌套，最多可以买 33 个鸡母*/
        {
            z=100-x-y;                                  /*鸡雏的数量，三者满足 100 只鸡*/
            if(5*x+3*y+z/3.0==100)                      /*判断这 3 种鸡要花 100*/
                printf("鸡翁=%d,鸡母=%d,鸡雏=%d\n",x,y,z);  /*如果为真，输出结果*/
        }
    }
    return 0;
}
```

在代码中可以看到，首先定义3个变量x，y，z，分别代表鸡翁、鸡母和鸡雏的数量。然后是两层for循环，外层循环计算出最多买多少只鸡翁，内层循环计算出最多买多少只鸡母，再用“100-鸡翁-鸡母”计算出可以买多少只鸡雏。最后用if语句判断这3种鸡的价格是否正好需要花100钱，如果为真，则用printf函数输出结果。运行效果如图7.11所示。

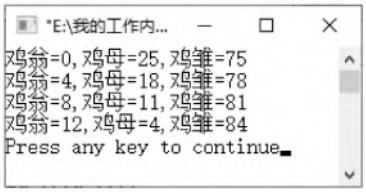


图7.11　使用嵌套语句输出百钱买百鸡结果

【例7.9】打印乘法口诀表（实例位置：资源包\TM\sl\07\09）

本实例要求打印出乘法口诀表，每一项都是行和列的相乘算式。根据这个特点，使用循环嵌套将其显示。

```
#include<stdio.h>
int main()
{
    int iRow, iColumn;                                          /*定义变量 iRow 为行，iColum 为列*/
    for(iRow = 1; iRow <= 9; iRow++)                            /*for 循环，先逐行循环*/
    {
        for(iColumn = 1; iColumn <= iRow; iColumn++)            /*再逐行循环，且列数小于等于行数*/
        {
            printf("%d*%d=%d ", iRow,iColumn,iRow *iColumn);    /*输出乘法口诀*/
        }
        printf("\n");                                           /*进行换行*/
    }
    return 0;
}
```

在本实例中用到两次for循环，第一个for循环可看成乘法口诀表的行数，同时也是每行进行乘法运算的第一个因子；第二个for循环范围的确定建立在第一个for循环的基础上，即第二个for循环的最大取值是第一个for循环中变量的值。

运行程序，显示效果如图7.12所示。

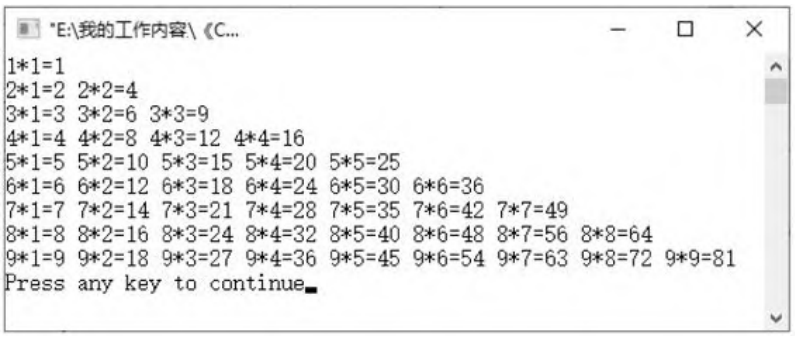


图7.12　打印乘法口诀表

编程训练（答案位置：资源包\TM\sl\07\编程训练\）

训练7：打印杨辉三角　用for语句的嵌套循环结构编写程序，打印杨辉三角（输出前5行）。输出结果如下：

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

训练8：输出金字塔形状　利用嵌套循环输出金字塔形状。输出一个三角形要考虑3点：首先要控制输出三角形的行数，其次控制三角形的空白位置，最后将三角形输出显示。输出结果如下：

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

7.7　转移语句

转移语句包括goto语句、break语句和continue语句。通过这3种语句，程序的执行流程会发生一定的转移。下面对这3种语句的使用方式进行详细介绍。

7.7.1　goto语句



goto语句为无条件转移语句，可以使程序立即跳转到函数内部的任意一条可执行语句处。goto关键字后面带一个标识符，该标识符是同一个函数内某条语句的标号。标号可以出现在任何可执行语句的前面，并且以一个冒号“:”作为后缀。goto语句的一般形式如下：

```
goto 标识符;
```

goto后的标识符就是要跳转的目标，当然这个标识符要在程序的其他位置给出，并且其标识符要位于函数内部。函数的内容将会在后面章节进行介绍，在此对其简单了解即可。例如：

```
goto Show;
printf("the message before ShowMessage");
Show:
    printf("ShowMessage");
```

在上述代码中，goto后的Show为跳转的标识符，第3行的“Show:”代码表示goto语句要跳转的位置，因此第2行的printf函数将不会被执行，而会执行第4行的printf函数。

注意

跳转的方向可以向前，也可以向后；可以跳出一个循环，也可以跳入一个循环。

【例7.10】“下一步”还是“退出”（实例位置：资源包\TM\sl\07\10）

本实例要求在执行循环操作的过程中，当用户输入退出指令后，程序跳转到循环外部。

```
#include<stdio.h>
int main()
{
    int iStep;                                  /*定义变量，表示外层 for 循环的步骤号*/
    int iSelect;                                /*定义变量，保存用户输入的选项*/
    for(iStep=1;iStep<10;iStep++)               /*外层 for 循环，记录循环的次数（步骤号）*/
    {
        printf("步骤号:%d\n",iStep);            /*输出当前循环的步骤号*/
        do                                      /*内层 do...while 循环，保证用户输入 0 或 99*/
        {
            printf("请输入一个选择序号：\n");   /*输出提示信息*/
            printf("(0 是退出,99 是下一步)\n");
            scanf("%d",&iSelect);               /*用户输入选择*/
            if(iSelect==0)                      /*判断输入的是否为 0*/
            {
                goto exit;                      /*如果是 0，执行 goto 跳转语句*/
            }
        }
        while(iSelect!=99);                     /*判断用户输入，只要不是 99 就重新输入序号*/
    }
exit:                                           /*跳转语句执行位置*/
    printf("退出程序!\n");
    return 0;
}
```

（1）程序运行时，外层for循环控制程序步骤，输出的循环步骤为1。信息提示输入数字，其中0表示退出，99表示下一个步骤。

（2）在for循环中使用do…while语句判断用户输入，当条件为假时，循环结束并执行for循环的下一步。假如用户输入数字3，既不退出也不执行下一步骤，而是继续提醒输入序号。当输入数字为99时，跳转到下一步，显示提示信息“步骤号:2”。

（3）如果用户输入的是0，那么通过if语句判断为真，执行其中的goto语句进行跳转，其中exit为跳转的标识符。循环的外部使用“exit:”表示goto跳转的位置，输出“退出程序！”。

运行程序，显示效果如图7.13所示。



图7.13　使用goto语句从循环内部跳出

7.7.2　break语句



有时会遇到这样的情况，不管表达式检验的结果如何，都需要强行终止循环，这时可以使用break语句。break语句用于终止并跳出当前循环，然后继续执行后面的代码。break语句的一般形式如下：

```
break;
```

例如，在while循环语句中使用break语句：

```
while(1)
{
    printf("Break");
    break;
}
```

在上述代码中，while语句是一个条件永远为真的循环，但由于在其中使用了break语句，使得程序流程跳出循环。

break语句不能用于除循环语句和switch语句之外的任何其他语句中。另外，在多层循环嵌套的情况下，使用break语句只能跳出当前循

环，这点一定要注意。

注意

> 循环体中的break语句和switch…case分支结构中的break语句的作用是不同的。

【例7.11】寻找break（**实例位置：资源包\TM\sl\07\11**）

使用for语句执行循环输出10次的操作，在循环体中判断输出的次数。当循环变量为5次时，使用break语句跳出循环，终止循环输出操作。

```
#include<stdio.h>
int main()
{
    int iCount;                                  /*定义循环控制变量*/
    for(iCount=0;iCount<10;iCount++)             /*执行 10 次循环*/
    {
        if(iCount==5)                            /*判断条件，如果 iCount 等于 5，则跳出*/
        {
            printf("Break here\n");
            break;                               /*跳出循环*/
        }
        printf("the counter is:%d\n",iCount);    /*输出循环的次数*/
    }
    return 0;
}
```

变量iCount在for语句中被赋值为0，因为iCount<10，所以循环执行10次。在循环语句中使用if语句判断当前iCount的值。当iCount值为5时，if判断为真，使用break语句跳出循环。

运行程序，显示效果如图7.14所示。



图7.14　使用break语句跳出循环

7.7.3　continue语句



在某些情况下，程序需要返回到循环头部继续执行，而不是跳出循环，此时可以使用continue语句。continue语句的一般形式如下：

```
continue;
```

其作用就是结束本次循环，即跳过循环体中尚未执行的部分，直接执行下一次的循环操作。

注意

continue语句和break语句的区别是：continue语句只结束本次循环，而不是终止整个循环的执行；break语句则是结束整个循环过程，不再判断执行循环的条件是否成立。

【例7.12】寻找Continue（**实例位置：资源包\TM\sl\07\12**）

本实例与例7.11基本相似，区别在于将使用break语句的位置改写成了continue。因为continue语句只结束本次循环，所以剩下的循环还是会继续执行。

```
#include<stdio.h>
int main()
{
    int iCount;                                  /*定义循环控制变量*/
    for(iCount=0;iCount<10;iCount++)             /*执行 10 次循环*/
    {
        if(iCount==5)                            /*判断条件，如果 iCount 等于 5，则跳出*/
        {
            printf("Continue here\n");
            continue;                            /*跳出本次循环*/
        }
        printf("the counter is:%d\n",iCount);    /*输出循环的次数*/
    }
    return 0;
}
```

运行程序，显示效果如图7.15所示。可以看到在iCount等于5时，调用continue语句使得本次的循环结束。但是循环本身并没有结束，因此程序会继续执行。

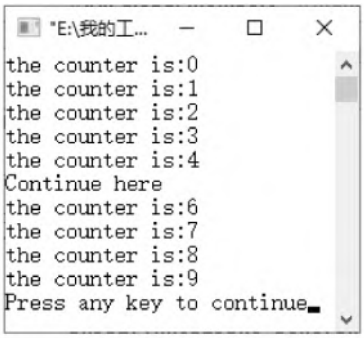


图7.15　使用continue语句结束本次循环

编程训练（答案位置：资源包\TM\sl\07\编程训练\）

训练9：健身房邂逅　假设一个正在跑步的男生邂逅了一个女生，这个男生跑到10分钟的时候就停止跑步，去询问女生要微信号，请用break语句来模拟此场景。输出结果如下：

```
已经跑了1分钟
已经跑了2分钟
已经跑了3分钟
已经跑了4分钟
已经跑了5分钟
已经跑了6分钟
```

```
已经跑了7分钟
已经跑了8分钟
已经跑了9分钟
看见美女了，我要与她约会1
```

训练10：自动售货机收费系统　自动售货机内有3种饮料，价格分别为3元、5元和7元。该售货机仅支持1元硬币支付。请编写该自动售货机的收费系统，输出结果如下：

```
正在售卖的饮料及其价格：1：3元　　2：5元　　3：7元
请输入要购买的饮料编号：
2
本自动售卖机仅支持1元硬币支付，请投币：
1
少于5元！请继续投币……
2
请您投入1元硬币！
1
少于5元！请继续投币……
1
少于5元！请继续投币……
1
少于5元！请继续投币……
1
已支付全款！请拿走您的饮料。
```

7.8　实践与练习

（答案位置：资源包\TM\sl\07\实践与练习\）

综合练习1：手机分期付款　输入想购买的手机价格，减掉首付300元，剩下的钱分6个月分期付款。已知6个月分期付款时，每个月的利息是0.6%，计算每个月需要还多少钱。运行程序，输出结果如下：

```
请输入你想买的手机价格：2699
手机的总价格是：2699.0元.
首付300之后还剩2399.0元
将所剩2399.0元进行分6期付款：
从我买手机开始，接下来的6个月每月需要还414.4元钱
```

综合练习2：测测你的星座　星座有12个，每个月对应一个星座（这里做了简化，假设每个星座都对应一个自然月）。编写程序，输入（循环输入）出生月份，就可以知道是什么星座的，还可以知道幸运数字。运行结果如下：

```
请输入你的出生月份：
8
你的星座是狮子座
幸运数是1
请输入你的出生月份：
7
你的星座是巨蟹座
幸运数是2
请输入你的出生月份：
```

综合练习3：模拟“跳一跳”小游戏的加分块　“跳一跳”是微信推出的小游戏。在游戏中，玩家停留在不同的加分块上，可以额外加分。例如在中心，能增加2分；在音乐盒子上，可以增加30分；在微信支付块上，可以增加10分。编写程序，模拟“跳一跳”小游戏的加分块，运行结果如下：

```
--------------跳一跳-------------
欢迎回来，请开始游戏……
请输入（1:中心 2:音乐块 3:微信支付块）：
2
您选择的是音乐块。
您的分数为：30
请输入（1:中心 2:音乐块 3:微信支付块）：
1
您选择的是中心。
您的分数为：32
请输入（1:中心 2:音乐块 3:微信支付块）：
4
对不起，您输入的有误
```

综合练习4：查询银行名称　图7.16所示为各大银行的标志，编写程序，输入要查询的银行号码，输出对应的银行名称。运行结果如图7.17所示。



图7.16　各大银行logo

```
1 对应银行为：  ICBI
2 对应银行为：  ABC
3 对应银行为：  CCB
4 对应银行为：  BOC
5 对应银行为：  CMB
6 对应银行为：  CEB
7 对应银行为：  CMBC
8 对应银行为：  CIB
9 对应银行为：  BCM
---------银行全称查询系统---------
请输入要查询的数据：
5
5 对应银行为：招商银行
请输入要查询的数据：
1
1 对应银行为：  中国工商银行
请输入要查询的数据：
9
9 对应银行为：  中国交通银行
请输入要查询的数据：
0
您的输入有误，请重新输入……
```

图7.17　查询银行名称

综合练习5：猜数字游戏　**编写一个猜数字的小游戏，随机生成一个1~10（包括1和10）的数字作为基准数，玩家每次通过键盘输入一个数字，如果输入的数字和基准数相同，则成功过关，否则重新输入。运行结果如下：**

```
请输入一个数字：
200
你猜大了,请重新输入：
150
你猜大了,请重新输入：
140
你猜小了,请重新输入：
145
```

```
你猜小了,请重新输入：
147
恭喜你,猜对了！！
```

综合练习6：客车承载量提醒　一辆客车只能承载25人，如果超过25人，司机就会拒绝载客。编写程序，提醒客车还能承载多少人。运行结果如下：

```
目前车上有的乘客数量是:20
还能承载 5 人，
还能承载 4 人，
还能承载 3 人，
还能承载 2 人，
还能承载 1 人，
还能承载 0 人，
座位已满，不能再承载了。
```

综合练习7：剧院卖票　某剧院发售演出门票，演播厅观众席有4行，每行有10个座位。为了不影响观众视角，在发售门票时，屏蔽掉最左一列和最右一列的座位，其他座位均可售出。编写程序，模拟剧院卖票，显示效果如图7.18所示。

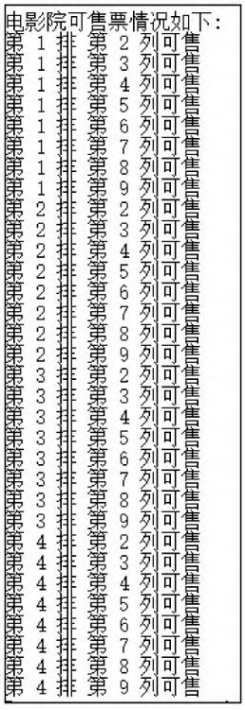


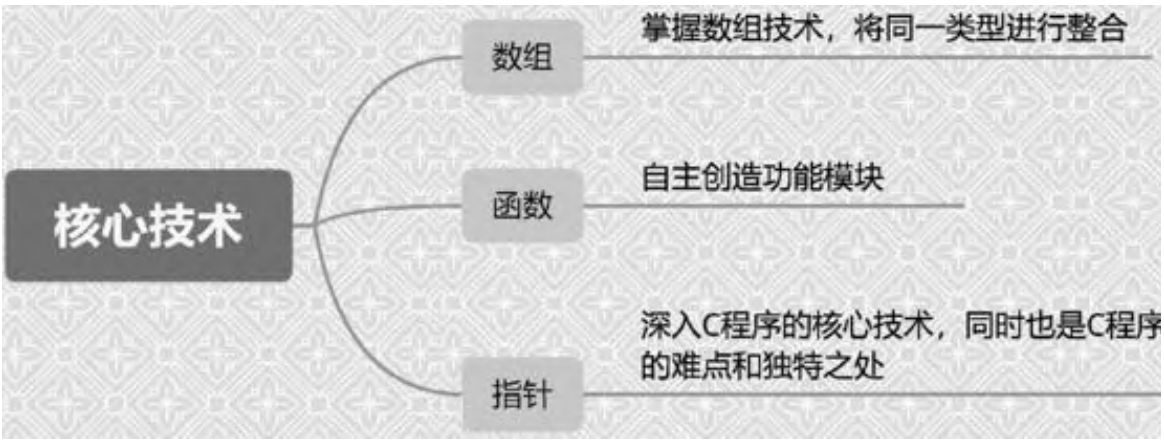
图7.18　剧院售票

综合练习8：模拟10086查询功能　10086是中国移动的客户服务热线，用户可以拨打10086查询自己的手机号码的套餐情况。编写程序，查询当前话费余额、剩余流量、剩余通话时长等信息。运行结果如下：

```
—          —10086查询功能—          —
输入1，查询当前话费余额
输入2，查询当前剩余流量
输入3，查询当前剩余通话
输入0，退出自助查询系统！请输入：
1
当前话费余额为：999元
请输入：
2
当前剩余流量为：5G
请输入：
3
当前剩余通话为：189分钟
请输入：
0
退出自助查询系统！
```

第2篇
核心技术

本篇介绍了C语言中数组、函数和指针这三大部分内容，并将前面所学的基础内容融入其中，学习更高级的程序设计知识。读者学习完本篇知识，可编写一些简单的C语言应用程序。



第8章
数组

编写程序过程中，有时会用到大量相同类型的数据。如果每个变量都需要单独定义，编程过程将会变得极其烦琐。使用数组可以很好地解决这个问题。

本章致力于使读者掌握一维数组和二维数组的应用，能利用所学知识解决一些实际问题；掌握字符数组的使用及其相关操作；并能通过数组掌握常见的排序算法。

本章的知识架构及重难点如下：



8.1 一维数组

8.1.1 一维数组的定义和引用



1．一维数组的定义

一维数组实际上是一组相同类型数据的线性集合。其定义形式如下：

```
类型说明符 数组标识符[常量表达式];
```

☑ 类型说明符：表示数组中元素的类型。

☑ 数组标识符：表示该数组型变量的名称，命名规则与变量名一致。

☑ 常量表达式：定义了数组中存放的数据元素个数，即数组长度。

例如，定义一个包含5个整型元素的数组，代码如下：

```
int iArray[5];
```

代码中的int为数组元素的类型，iArray为数组变量名，括号中的5表示数组中包含5个元素。

2．一维数组的引用

数组定义后，可以引用其中的数组元素，引用方式为“数组标识符[下标]”。

例如，引用数组iArray中的第3个变量，格式为iArray[2]。其中，iArray是数组变量名，2为数组下标。有的读者会问：为什么引用第3个数组元素使用的是下标2呢？这是因为数组下标是从0开始的，iArray[0]表示第一个元素，iArray[1]表示第2个数组元素，iArray[2]表示第3个数组元素。

注意

数组下标可以是整型常量或整型表达式。在数组iArray[5]中，只能使用iArray[0]、iArray[1]、iArray[2]、iArray[3]、iArray[4]，而不能使用iArray[5]。若使用iArray[5]，则会出现下标越界错误。

【例8.1】使用数组保存手机号（实例位置：资源包\TM\sl\08\01）

在本实例中，使用数组保存用户输入的手机号，并输出显示。

```
#include<stdio.h>
int main()
{
    int iArray[11], index;                          /*定义数组及变量为基本整型*/
    printf("请输入手机号:\n");
    for (index= 0; index< 11; index++)              /*逐个输入手机号码，保存为数组元素*/
    {
        scanf("%d",&iArray[index]);
    }
    printf("手机号是:\n");
    for (index = 0; index< 11; index++)             /*循环输出数组中的元素*/
    {
        printf("%d", iArray[index]);
    }
    printf("\n");
    return 0;
}
```

（1）定义index作为循环控制变量，定义数组iArray[11]用来保存11位手机号码。

（2）通过一个for循环，依次输入11位手机号码（用空格隔开），存储到各数组元素对应的地址中。这里，iArray[index]就是对数组元素的引用，“&”为取地址符。

（3）循环输出各数组元素，得到完整的手机号码。



图8.1 使用数组保存手机号

运行程序，显示效果如图8.1所示。

8.1.2 一维数组的初始化



一维数组的初始化，可以用以下3种方法实现。

（1）定义数组时直接对数组元素赋初值（数组元素值放在一对大括号中）。例如：

```
int i,iArray[6]={1,2,3,4,5,6};
```

定义和初始化之后，iArray[0]=1，iArray[1]=2，iArray[2]=3，iArray[3]=4，iArray[4]=5，iArray[5]=6。

（2）如果只给一部分数组元素赋值，则未赋值的元素默认为被赋值0。例如：

```
int iArray[6]={0,1,2};
```

数组iArray包含6个元素，但初始化时只给出了3个值，结果是数组前3个元素得到赋值，后3个元素被默认赋值为0。

（3）当对全部数组元素都赋初值时，可以不指定数组长度。例如：

```
int iArray[]={1,2,3,4};
```

上述代码的大括号中有4个元素，因此系统会默认为该数组变量的长度为4。

【例8.2】计算篮球平均成绩（实例位置：资源包\TM\sl\08\02）

记分员记录了某球员在10场篮球比赛中的成绩，求该球员的平均成绩。

```
#include<stdio.h>
int main()
{
    int grade[10]={12,5,21,15,32,10,25,14,30,20};      /*定义数组，给出球员的 10 场比赛成绩*/
    int total=0;                                        /*定义变量，表示总成绩*/
    int i;
    float avg;                                          /*定义变量，表示平均成绩*/
    for(i=0;i<10;i++)                                   /*循环累计总成绩*/
    {
        total+=grade[i];
    }
    avg=((float)total/10);                              /*计算平均成绩*/
    printf("篮球比赛的平均成绩是：%f\n",avg);
    return 0;
}
```

程序中首先定义一个数组grade，初始化数值为10场球赛的成绩；定义整型变量total，用来保存总成绩；定义浮点型变量avg，用来保存平均成绩。接着通过一个for循环，控制循环变量使10场球赛成绩累加，相加结果赋给total。最后计算平均成绩，赋给avg，并用printf函数输出结果。

运行程序，显示效果如图8.2所示。

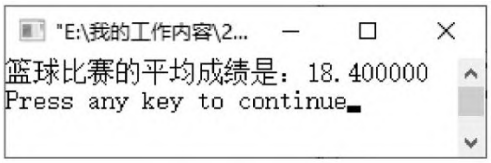


图8.2　初始化一维数组

8.1.3　一维数组的应用



一个班级中往往有很多学生，使用数组来保存这些学生的姓名、编号等，管理起来非常方便。

【例8.3】输出插队之后的编号（实例位置：资源包\TM\sl\08\03）

体育课上，老师按身材高矮给20名同学编号。刚编完号，一位男生姗姗来迟，老师比对后将他排在8号位置，并重新排列后面的同学。使用数组，输出男生插队后有变化的学生编号。代码如下：

```
#include<stdio.h>
int main()
{
    /*定义数组及变量为基本整型*/
    int iArray[20]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}, index;
    int iArray1[21]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21}, index1;
    printf("体育老师按身材高矮排队编号，老师排好，编号是:\n");  /*提示信息*/
    for (index = 0; index< 20; index++)                          /*循环输出插队前所有学生的编号*/
    {
        printf("%d ", iArray[index]);
    }
    printf("\n");
    printf("重新排列插队男生及后面的同学，他们的编号是:\n");    /*男生插入后，从 8 号开始重新编号*/
    for (index1 = 7; index1< 21; index1++)                       /*从下标 7 开始，循环输出后续编号*/
    {
        printf("%d ", iArray1[index1]);
    }
    printf("\n");
    return 0;
}
```

可以看出，要想从第8名同学开始编号，就应该从下标7开始。运行程序，效果如图8.3所示。

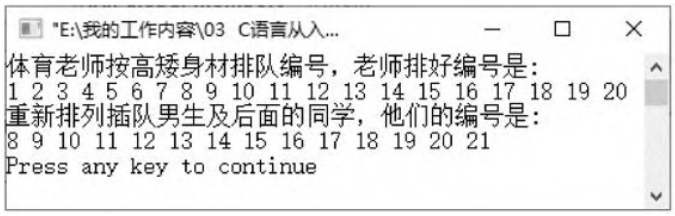


图8.3　重新输出编号

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练1：保存语、数、外成绩　编写程序，输入语文、数学、英语3门学科的成绩，使用数组保存并输出。运行效果如下：

```
请输入语文、数学、英语的成绩：
89 95 96
```

```
语文、数学、英语的成绩分别如下:
89 95 96
```

训练2：清空购物车　2018年亚运会上，中国女排3:0战胜泰国女排，夺得冠军，当时阿里巴巴承诺要帮女排姑娘们清空购物车。模拟输出某女排姑娘的购物车清单。运行效果如下:

```
淘宝购物车清单:

====生活用品类:====
纸抽,纸巾,收纳箱,水杯,垃圾袋,剪刀,挂钩,拖鞋,小闹钟

====化妆品类:====
保湿套装,气垫cc,隔离霜,防晒霜,眉粉,眼影色盘,睫毛膏

====运动类商品:====
运动服,球鞋,护腕,护膝,护掌,排球,瑜伽垫,瑜伽球

====保健类商品:====
蛋白粉,口服液,眼部按摩仪,血压计,脚底按摩器
```

8.2　二维数组

8.2.1　二维数组的定义和引用



1．二维数组的定义

二维数组可以看作是特殊的一维数组，其各个元素仍然是一个数组。其定义形式如下:

```
数据类型 数组名[常量表达式1][常量表达式2];
```

其中，“常量表达式1”定义了二维数组的行数，“常量表达式2”定义了二维数组的列数。

不管是行下标还是列下标，其索引都是从0开始的。因此，一个m行n列的二维数组array[m][n]，其行下标取值范围为0~m-1，列下标取值范围为0~m-1，最大下标元素是array[m-1][n-1]。

例如，定义一个3行4列的整型数组，代码如下：

```
int array[3][4];
```

二维数组array[3][4]共包含12个数组元素（3×4），分别为array[0][0]、array[0][1]、array[0][2]、array[0][3]、array[1][0]、array[1][1]、array[1][2]、array[1][3]、array[2][0]、array[2][1]、array[2][2]和array[2][3]。可见，C语言中数组是按行排列的，数组array[3][4]在内存中先按行顺次存放，依次为array[0]行、array[1]行和array[2]行，每行有4个元素，也依次存放。

2．二维数组的引用

二维数组元素的引用形式为“数组名[下标][下标]”。例如，array[1][2]表示对array数组的第2行第3个元素进行引用。

和一维数组一样，二维数组也要注意下标越界的问题。例如：

```
int array[2][4];
...                                    /*对数组元素进行赋值*/
array[2][4]=9;                         /*错误，下标越界！*/
```

8.2.2　二维数组的初始化



二维数组赋初值，有以下4种情况。

（1）将所有数据写在一个大括号内，按照数组元素排列顺序对元素赋值。例如：

```
int array[2][2] = {1,2,3,4};
```

如果大括号内的数据少于数组元素的个数，则系统会默认后面未被赋值的元素值为0。

（2）为所有元素赋初值时，可以省略行下标，但不能省略列下标。例如：

```
int array[][3] = {1,2,3,4,5,6};
```

系统会根据数据的个数进行分配，一共有6个数据，而数组分为3列，因此数组有2行。

（3）分行给数组元素赋值。例如：

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

在分行赋值时，可以只对部分元素赋值。例如，下面的代码中，a[0][0]的值是1；a[0][1]的值是2；a[0][2]的值是0；a[1][0]的值是4；a[1][1]的值是5；a[1][2]的值是0。

```
int a[2][3] = {{1,2},{4,5}};
```

（4）直接对数组元素赋值。例如：

```
int a[2][3];
a[0][0] = 1;
a[0][1] = 2;
```

【例8.4】魔方阵数据（实例位置：资源包\TM\sl\08\04）

一个3×3的网格，将1~9的数字放入方格中（矩阵中心元素为5），使得每行、每列以及每条对角线上的3个数相加都相同。通过键盘为二维数组元素赋值，并显示二维数组。

```
#include<stdio.h>
int main()
{
    int a[3][3];                          /*定义二维数组*/
    int i,j;                              /*定义循环控制变量*/
    for(i=0;i<3;i++)                      /*先行后列，依次输入数据，为数组元素赋值*/
    {
        for(j=0;j<3;j++)
        {
            printf("a[%d][%d]=",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("输出二维数组:\n");            /*信息提示*/
    for(i=0;i<3;i++)                      /*先行后列，循环输出所有数组元素的值*/
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",a[i][j]);       /*使用制表符控制间距，可排列更整齐*/
        }
        printf("\n");                     /*使数组元素分行显示*/
    }
    return 0;
}
```

（1）程序先依次输入相应的数组元素值，然后将这个3行3列的数组输出。

（2）给数组赋值时，使用了for循环嵌套，第一层循环行，第二层循环列，用printf函数输出元素名，用scanf函数输入数据，赋给对应的数组元素。

（3）输出数组时，也使用了for循环嵌套，第一层循环行，第二层循环列，用printf函数逐行输出数组元素。为了使输出的数据更整齐，使用制表符“\t”来控制间距。

运行程序，显示效果如图8.4所示。

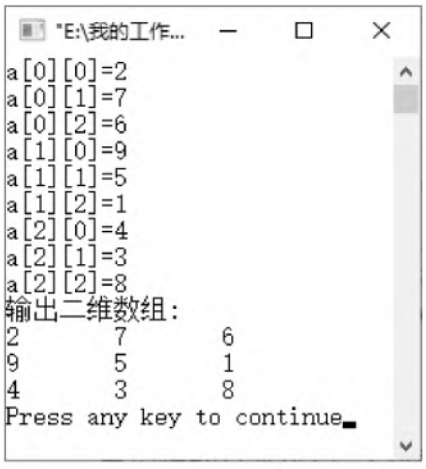


图8.4　使用二维数组保存数据

8.2.3　二维数组的应用



【例8.5】将矩阵行列对换（实例位置：资源包\TM\sl\08\05）

本实例中，把二维数组中各行的元素换成列元素，各列的元素换成行元素，生成一个新的二维数组。代码如下：

```c
#include<stdio.h>
int main()
{
    int a[2][3],b[3][2];                    /*定义两个数组*/
    int i,j;                                /*定义循环控制变量*/

    for(i=0;i<2;i++)                        /*先行后列，输入数据，为数组元素赋值*/
    {
        for(j=0;j<3;j++)
        {
            printf("a[%d][%d]=",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("输出二维数组:\n");              /*信息提示*/
    for(i=0;i<2;i++)                        /*先行后列，循环输出二维数组 a 中的元素*/
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n");                       /*使数组元素分行显示*/
    }
    for(i=0;i<2;i++)                        /*将数组 a 转置后存入数组 b 中*/
    {
        for(j=0;j<3;j++)
        {
            b[j][i] = a[i][j];              /*通过行列互换，b 为 a 的转置数组*/
        }
    }
    printf("输出转换后的二维数组:\n");
    for(i=0;i<3;i++)                        /*先行后列，循环输出转置数组 b 中的元素*/
    {
        for(j=0;j<2;j++)
        {
            printf("%d\t",b[i][j]);
        }
        printf("\n");                       /*使数组元素分行显示*/
    }
    return 0;
}
```

（1）利用双重for循环为2行3列的数组赋值，然后分行显示其元素。

（2）通过一个双重for循环，将二维数组a中的元素赋值到转置后的二维数组b中。

（3）通过循环控制，将转置后的二维数组b中的元素输出。

运行程序，显示效果如图8.5所示。



图8.5　矩阵转置运行图

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练3：打印数组对角线上的字符　有如下二维数组：

```
char ccArray[5][5] = {
        {'a','b','c','d','e'},
        {'b','a','8','d','d'},
        {'c','d','a','e','c'},
        {'d','j','f','a','b'},
        {'e','d','a','f','a'},
};
```

编写程序，打印数组对角线（左上至右下）上的字符。输出结果如下：

```
对角线上的字符是:a
对角线上的字符是:a
对角线上的字符是:a
对角线上的字符是:a
对角线上的字符是:a
```

训练4：求平均成绩　用二维数组求表8.1中各科的平均成绩。

表8.1　成绩表

	宋小美	张大宝	高心心	彭果	邓丽
数学	93	87	90	76	70
语文	90	76	60	80	81
英语	70	88	72	77	96

运行结果如下：

```
请输入成绩：
array[0][0]=93
array[0][1]=87
array[0][2]=90
array[0][3]=76
array[0][4]=70
array[1][0]=90
array[1][1]=76
array[1][2]=60
array[1][3]=80
array[1][4]=81
array[2][0]=70
array[2][1]=88
array[2][2]=72
array[2][3]=77
array[2][4]=96
数学的平均成绩是（取整数）:83
语文的平均成绩是（取整数）:77
英语的平均成绩是（取整数）:80
```

8.3 字符数组

数组中的元素类型为字符型时，称为字符数组。字符数组中的每个元素可以存放一个字符。字符数组的定义和引用方法与其他数据类型相似。

8.3.1 字符数组的定义和引用



1．字符数组的定义

字符数组的定义形式如下：

```
char 数组标识符[常量表达式]
```

例如，下述代码定义了一个字符数组cArray，该数组中包含5个字符型的变量元素。

```
char cArray[5];
```

2．字符数组的引用

字符数组的引用也采用下标的形式。例如，引用数组cArray中的元素，并依次赋值，代码如下：

```
cArray[0]='H';
cArray[1]='e';
cArray[2]='l';
cArray[3]='l';
cArray[4]='o';
```

8.3.2 字符数组的初始化



在对字符数组进行初始化操作时，有以下几种方法。

（1）逐个字符赋给数组中的元素。

这是最容易理解的初始化字符数组的方式。例如：

```
char cArray[5]={'H','e','l','l','o'};
```

定义了一个包含5个元素的字符数组，在初始化的大括号中，每个字符赋值给一个数组元素。

（2）定义字符数组的同时进行初始化，此时可以省略数组长度。

如果初值个数与预定的数组长度相同，在定义时可以省略数组长度，系统会自动根据初值个数来确定数组长度。例如，上面初始化字符数组的代码可以写成：

```
char cArray[]={'H','e','l','l','o'};
```

代码中定义的cArray[]没有给出数组大小，但系统会自动根据初值的个数确定数组长度为5。

（3）利用字符串给字符数组赋初值。

字符数组可用来存放字符串。因此，也可以用字符串的方式对字符数组进行初始化赋值。例如：

```
char cArray[]={"Hello"};
```

或者将“{}”去掉，写成：

```
char cArray[]="Hello";
```

【例8.6】打印停车场标志（实例位置：资源包\TM\sl\08\06）

本实例将利用字符数组输出“Park”。定义一个字符数组，通过初始化操作保存一串字符，然后通过循环引用将每一个数组元素输出。

```
#include<stdio.h>
int main()
{
    char cArray[5]={'P','a','r','k'};          /*定义字符数组并初始化*/
    int i;                                     /*循环控制变量*/
    for(i=0;i<5;i++)                           /*循环输出字符数组元素*/
    {
        printf("%c",cArray[i]);
    }
    printf("\n");                              /*输出换行*/
    return 0;                                  /*程序结束*/
}
```

初始化字符数组时要注意，每个元素都要使用一对单引号（' '）括起来。在for循环中，因为输出的类型是字符型，所以printf函数中使用的是“%c”格式符。通过循环变量i，cArray[i]实现了对数组中不同元素的引用。

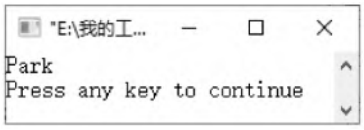


图8.6　使用字符数组输出一个字符串

运行程序，显示效果如图8.6所示。

8.3.3　字符数组的结束标志



在C语言中，使用字符数组保存字符串时，系统会自动为其添加“\0”作为结束符。也就是说，用字符串方式赋值比用字符逐个赋值要多占一个字节，多占的这个字节用于存放字符串结束标志“\0”。

例如，使用下述代码可以初始化一个字符数组：

```
char cArray[]="Hello";
```

上面的字符数组cArray在内存中的实际存放情况如图8.7所示。

H	e	l	l	o	\0

图8.7　cArray在内存中的实际存放情况

“\0”是由C编译系统自动加上的。因此上面的赋值语句等价于：

```
char cArray[]={'H','e','l','l','o','\0'};
```

字符数组并不要求最后一个字符为“\0”，甚至可以不包含“\0”。因此下面的写法也是合法的：

```
char cArray[5]={'H','e','l','l','o'};
```

是否加“\0”，应根据需要来决定。由于系统对字符串常量会自动加一个“\0”，因此，为了使处理方法一致，且便于测定字符串的实际长度，在字符数组中也常常人为地加上一个“\0”。例如：

```
char cArray[6]={'H','e','l','l','o','\0'};
```

注意

如果在一个字符数组中先后存放多个不同长度的字符串，则应使数组长度大于最长的那个字符串的长度。

8.3.4 字符数组的输入和输出



字符数组的输入和输出可以使用两种格式字符：“%c”和“%s”。

使用格式字符“%c”可实现数组中字符的逐个输入与输出。例如，逐个输出字符数组中的元素：

```
for(i=0;i<5;i++)                                    /*进行循环*/
{
    printf("%c",cArray[i]);                         /*逐个字符输出数组元素*/
}
```

使用格式字符“%s”可将整个字符串输入或输出。例如，输出一个字符串：

```
char cArray[]="GoodDay!";                    /*初始化字符数组*/
printf("%s",cArray);                         /*输出字符串*/
```

使用格式字符“%s”输出字符串时，需要注意以下几种情况。

- ☑ 输出字符中不包括结束符“\0”。
- ☑ printf函数中的输出项是字符数组名cArray，而不是数组中的元素名cArray[0]等。
- ☑ 即使数组长度大于字符串实际长度，也只会输出到“\0”为止。
- ☑ 如果一个字符数组中包含多个“\0”结束字符，则在遇到第一个“\0”时输出就结束。

【例8.7】输出一条心灵鸡汤（实例位置：资源包\TM\sl\08\07）

输出名言“Education is the door to freedom”（教育是通向自由之门）。对定义的字符数组初始化，然后采用两种方式输出字符数组中保存的数据，先逐个字符输出数组元素，再直接将字符串整体输出。

```
#include<stdio.h>
int main()
{
    int iIndex;                                  /*循环控制变量*/
    char cArray[33]="Education is the door to freedom";   /*定义字符数组，用于保存字符串*/
    char cArray2[100]="教育是通向自由之门";

    for(iIndex=0;iIndex<33;iIndex++)
    {
        printf("%c",cArray[iIndex]);             /*逐个字符输出数组元素*/
    }
    printf("\n%s\n",cArray2);                    /*将字符串整体输出*/
    return 0;
}
```

在代码中，对数组元素逐个字符输出时，使用的是循环方式。直接输出字符串时，使用的是printf函数中的格式字符“%s”。要注意，直接输出字符串时，不能使用格式字符“%c”。

运行程序，显示效果如图8.8所示。

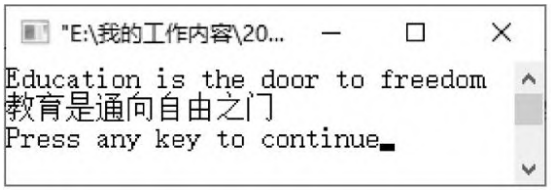


图8.8　使用两种方式输出字符串

注意

C语言中只存在字符类型，不存在字符串类型。字符类型只能存放单个字符，如果要存放字符串，就需要使用字符数组。当然，也可以使用字符型指针存放字符串。

8.3.5　字符数组的应用



【例8.8】计算字符串长度（实例位置：资源包\TM\sl\08\08）

随机输入一个字符串，计算该字符串的长度。

```
#include<stdio.h>
int main()
{
    int iIndex;                                  /*循环控制变量*/
    int length=0;                                /*定义变量 length，保存字符串长度*/
    char cArray[80];                             /*定义字符数组，保存用户输入的字符串*/
    printf("请输入字符串：\n");
    gets(cArray);                                /*用 gets 函数输入字符串*/
    for(iIndex=0;cArray[iIndex]!='\0';iIndex++)  /*循环字符数组，直到遇到“\0”结束*/
    {
        length++;                                /*每遍历一个字符，字符串长度加 1*/
    }
    printf("字符串长度是：%d\n",length);          /*输出字符串长度*/
    return 0;
}
```

使用gets函数将输入的字符串保存在cArray字符数组中。使用for循环判断当前数组元素是否为结束符“\0”，如果不是，则字符串长度加1；如果是，则退出循环。

运行程序，显示效果如图8.9所示。

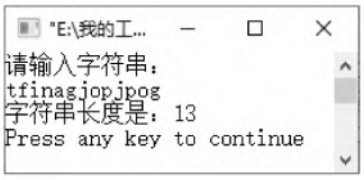


图8.9　计算字符串长度

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练5：统计单词个数　用户输入一行字符，然后统计其中有多少个单词。要求每个单词之间用空格分隔开，且最后的字符不能为空格。输出结果如下：

```
请输入字符串：
I Love China
一共有3个单词
```

训练6：确定男女主角　利用字符数组保存男女主角的名字，使用for循环遍历字符数组，通过两种形式分别输出男女主角的名字。输出结果如下：

```
这部电影的男主角分别是：
雨石,玉轩,团子
这部电影的女主角分别是：
小点,紫轩,若美
```

8.4　多维数组



多维数组的声明和二维数组相同，只是下标更多。其一般形式如下：

```
数据类型 数组名[常量表达式1][常量表达式2]…[常量表达式n];
```

例如，下面的代码中定义了一个三维数组iArray1和一个四维数组iArray2。

```
int iArray1[3][4][5];
int iArray2[4][5][7][8];
```

由于数组元素的位置可以通过偏移量计算出来，因此对于三维数组a[m][n][p]来说，元素a[i][j][k]所在的地址是从a[0][0][0]算起到i*n*p+j*p+k个单位的位置。

8.5 数组的排序算法

数组是一组有序数据的集合。这里，“有序”指的是数组元素在内存中的存放方式是有序的，其引用方式也有规律可循，而不是说数组元素在数组中是按照数值大小有序排列的。那么，有没有可能让数组元素按照数值大小有序排列呢？当然可以，下面就一起来学习下数组的各种常用排序算法。

8.5.1 选择法排序



选择法排序的原理如下：每次在待排序数组中查找最大或最小的数组元素，将其值与最前面没有进行过排序的数组元素的值互换。这里，由大到小排序应查找最大值，由小到大排序则应查找最小值。

下面以数字9、6、15、4、2为例，利用选择法使其从小到大排序，每次交换后的数字顺序如表8.2所示。

表8.2　选择法排序（从小到大）

排 序 过 程	元素【0】	元素【1】	元素【2】	元素【3】	元素【4】
起始值	9	6	15	4	2
第 1 次	2	6	15	4	9
第 2 次	2	4	15	6	9
第 3 次	2	4	6	15	9
第 4 次	2	4	6	9	15
排序结果	2	4	6	9	15

可以发现，第一次排序过程中，将第一个数字和最小的数字进行了位置互换；第二次排序过程中，将第二个数字和剩下数字中最小的数字进行了位置互换；依此类推，每次都将下一个数字和剩余数字中最小的数字进行位置互换，直到将一组数字按从小到大排序。

【例8.9】选择法从小到大排序（实例位置：资源包\TM\sl\08\09）

本实例中，声明了一个整型数组和两个整型变量。其中，整型数组用于存储用户输入的10个数值，而整型变量用于存储数值最小的数组元素和该元素的位置。通过双层循环进行选择法排序，最后将排好序的数组元素输出。

```c
#include <stdio.h>
int main()
{
    int i,j;
    int a[10];                          /*定义数组，存储用户输入的 10 个数*/
    int iTemp;                          /*定义变量，表示最小的数组元素*/
    int iPos;                           /*定义变量，表示元素位置*/
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                   /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }

    /*使用选择法对数组元素从小到大排序*/
    for(i=0;i<9;i++)                    /*设置外层循环下标为 0～8，表示前 9 个数组元素*/
    {
        iTemp = a[i];                   /*假设当前元素为最小值*/
        iPos = i;                       /*记录最小元素位置*/
        for(j=i+1;j<10;j++)             /*设置内层循环下标为 i+1～9，表示剩下的未排序数组元素部分
        {
            if(a[j]<iTemp)              /*如果后续的元素中有数比前面设定的最小值还小*/
            {
                iTemp = a[j];           /*重新设定最小值*/
                iPos = j;               /*修正最小元素位置*/
            }
        }
        a[iPos] = a[i];             /*此两行代码用于将最小的数组元素和当前排序次数对应的数组元素互换*
        a[i] = iTemp;
    }
     printf("排序结果如下：\n");
    for(i=0;i<10;i++)                   /*输出数组*/
    {
        printf("%d\t",a[i]);            /*用制表位分隔数据*/
        if(i == 4)                      /*如果是第 5 个元素*/
            printf("\n");               /*输出换行*/
    }
    printf("\n");
    return 0;                           /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）设置一个嵌套循环，第一层循环为前9个数组元素，并在每次循环时将对应当前次数的数组元素设置为最小值（如果当前是第3次循环，那么将数组中第3个元素，也就是下标为2的元素设置为当前的最小值）；在第二层循环中，循环比较该元素之后的各个数组元素，并将每次比较结果中较小的数设置为最小值，在第二层循环结束时，将最小值与开

始时设置为最小值的数组元素进行互换。当所有循环都完成以后，就将数组元素按照从小到大的顺序重新排列了。

（3）循环输出数组中的元素，并在输出5个元素以后换行，在下一行输出后面的5个元素。

运行程序，显示效果如图8.10所示。

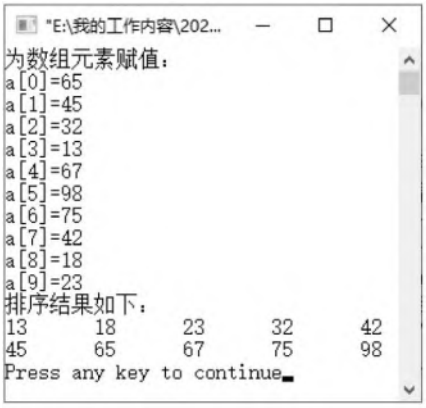


图8.10　选择法排序

8.5.2　冒泡法排序



冒泡法排序的原理如下：每次比较数组中相邻的两个数组元素的值，将较小的数排在较大的数前面，可实现数组元素从小到大排序；每次将较大的数排在较小的数前面，可实现数组元素从大到小排序。

仍以数字9、6、15、4、2为例，对这几个数字进行冒泡法排序，使其从小到大排列。每次排序后的顺序如表8.3所示。

表8.3　冒泡法排序（从小到大）

排序过程	元素【0】	元素【1】	元素【2】	元素【3】	元素【4】
起始值	9	6	15	4	2
第1次	2	9	6	15	4
第2次	2	4	9	6	15
第3次	2	4	6	9	15
第4次	2	4	6	9	15
排序结果	2	4	6	9	15

可以发现，在第一次排序过程中，将最小的数字移动到第一的位置，并将其他数字依次向后移动；在第二次排序过程中，从第二个数字开始的剩余数字中选择最小的数字，并将其移动到第二的位置，剩余数字依次向

后移动；依此类推，每次都将剩余数字中最小的数字移动到当前剩余数字的最前方，直到将一组数字按从小到大排序为止。

【例8.10】冒泡法从小到大排序（实例位置：资源包\TM\sl\08\10）

声明一个整型数组和一个整型变量，整型数组用于存储用户输入的数字，整型变量作为两个元素交换时的中间变量，通过双层循环进行冒泡法排序，最后将排好序的数组输出。

```
#include<stdio.h>
int main()
{
    int i,j;
    int a[10];
    int iTemp;
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                    /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }

    /*采用冒泡法使数组元素从小到大排序*/
    for(i=1;i<10;i++)                    /*外层循环元素的下标为 1～9，表示后 9 个数组元素*/
    {
        for(j=9;j>=i;j--)                /*内层循环元素的下标为 9～i，表示从最后一个元素开始向前循环*/
        {
            if(a[j]<a[j-1])              /*如果前一个数比后一个数大*/
            {
                /*交换两个数组元素的值，使小数前移，如同冒泡*/
                iTemp = a[j-1];
                a[j-1] = a[j];
                a[j] = iTemp;
            }
        }
    }
     printf("排序结果如下：\n");
    for(i=0;i<10;i++)                    /*输出数组*/
    {
        printf("%d\t",a[i]);             /*用制表位分隔数据*/
        if(i == 4)                       /*如果是第 5 个元素*/
            printf("\n");                /*输出换行*/
    }
      printf("\n");
    return 0;                            /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）设置一个嵌套循环，第一层循环为后9个数组元素。在第二层循环中，从最后一个数组元素开始向前循环，直到前面第一个没有进行排序的数组元素。循环比较这些数组元素，如果在比较中后一个数组元素的值小于前一个数组元素的值，则将两个数组元素的值进行互换。当所有循环都完成以后，就将数组元素按照从小到大的顺序重新排列了。

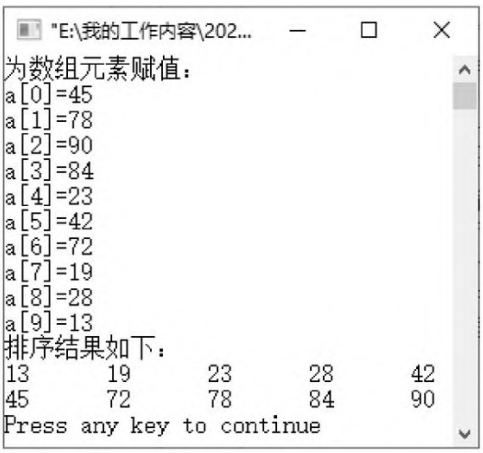


图8.11　冒泡法排序

（3）循环输出数组中的元素，并在输出5个元素以后进行换行，在下一行输出后面的5个元素。

运行程序，显示效果如图8.11所示。

8.5.3　交换法排序



交换法排序是将每一位数与其后的所有数一一比较，如果发现符合条件的数据，则交换数据。首先，用第一个数依次与其后的所有数进行比较，如果存在比其值大（小）的数，则交换这两个数，继续向后比较其他数直至最后一个数。然后再使用第二个数与其后面的数进行比较，如果存在比其值大（小）的数，则交换这两个数。继续向后比较其他数，直至最后一个数比较完成。

下面以数字9、6、15、4、2为例，进行交换法排序。每次排序后的顺序如表8.4所示。

表8.4　交换法排序（从小到大）

排 序 过 程	元素【0】	元素【1】	元素【2】	元素【3】	元素【4】
起始值	9	6	15	4	2
第 1 次	2	9	15	6	4
第 2 次	2	4	15	9	6
第 3 次	2	4	6	15	9
第 4 次	2	4	6	9	15
排序结果	2	4	6	9	15

可以发现，在第一次排序过程中将第一个数与后边的数依次进行比较。首先比较9和6，9大于6，交换两个数的位置，然后数字6成为第一个数字；用6和第3个数字15进行比较，6小于15，保持原来的位置；然后用6和4进行比较，6大于4，交换两个数字的位置；再用当前数字4与最后的数字2进行比较，4大于2，则交换两个数字的位置，从而得到表8.3中第一次的排序结果。然后使用相同的方法，从当前第二个数字9开始，继续和后面的数字进行比较，如果遇到比当前数字小的数字则交换位置，依此类推，直到将一组数字按从小到大排序为止。

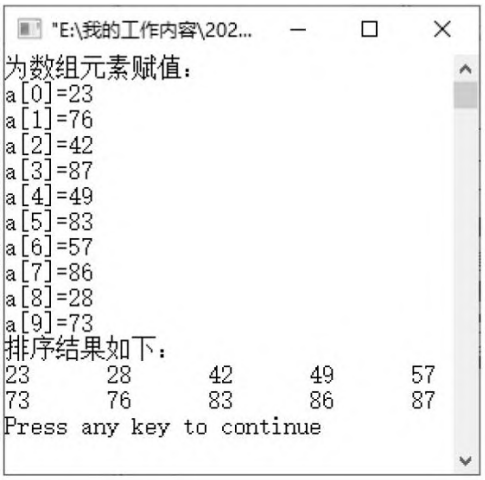
【例8.11】交换法从小到大排序（实例位置：资源包\TM\sl\08\11）

声明一个整型数组和一个整型变量，其中整型数组用于存储用户输入的数字，整型变量则作为两个元素交换时的中间变量，然后通过双层循环进行交换法排序，最后将排好序的数组输出。

```
#include<stdio.h>
int main()
{
    int i,j;
    int a[10];
    int iTemp;
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                    /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }

    /*采用交换法使数组元素从小到大排序*/
    for(i=0;i<9;i++)                     /*外层循环元素下标为 0～8，表示前 9 个数组元素*/
    {
        for(j=i+1;j<10;j++)              /*内层循环元素下标为 i+1～9，表示后面的待比较数组元素*/
        {
            if(a[j] < a[i])              /*如果后一个数比前一个数小*/
            {
                iTemp = a[i];            /*交换两个数组元素的值，使小数前移*/
                a[i] = a[j];
                a[j] = iTemp;
            }
        }
    }
    printf("排序结果如下：\n");
    for(i=0;i<10;i++)                    /*输出数组*/
    {
        printf("%d\t",a[i]);             /*用制表位分隔数据*/
        if(i == 4)                       /*如果是第 5 个元素*/
            printf("\n");                /*输出换行*/
    }
    printf("\n");
    return 0;                            /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。



（2）设置一个嵌套循环，第一层循环为前9个数组元素，然后在第二层循环中，将第一个数组元素与后面的数组元素依次进行比较，如果后面的数组元素值小于当前数组元素值，则交换两个元素值，然后使用交换后的第一个数组元素继续与后面的数组元素进行比较，直到本次循环

结束。将最小的数组元素值交换到第一个数组元素的位置，然后从第二个数组元素开始，继续与后面的数组元素进行比较。依此类推，直到循环结束，就将数组元素按照从小到大的顺序重新排列了。

图8.12　交换法排序

（3）循环输出数组中的元素，并在输出5个元素以后进行换行，在下一行输出后面的5个元素。

运行程序，显示效果如图8.12所示。

8.5.4　插入法排序



插入法排序较为复杂，其基本原理是：抽出一个数据，在前面的数据中寻找相应的位置插入，然后继续下一个数据，直到完成排序。

下面以数字9、6、15、4、2为例，使用插入法使其从小到大排序。每次排序后的顺序如表8.5所示。

表8.5　插入法排序（从小到大）

排序过程	元素【0】	元素【1】	元素【2】	元素【3】	元素【4】
起始值	9	6	15	4	2
第1次	9				
第2次	6	9			
第3次	6	9	15		
第4次	4	6	9	15	
排序结果	2	4	6	9	15

可以发现，在第一次排序过程中将第一个数取出来，并放置在第一个位置；然后取出第二个数，并将第二个数与第一个数进行比较，如果第二个数小于第一个数，则将第二个数排在第一个数之前，否则排在第一个数之后；然后取出下一个数，先与排在最后面的数字进行比较，如果当前数

字比较大，则继续排在最后，如果当前数字比较小，还要与之前的数字进行比较，如果当前数字比前面的数字小，则将其排在比它小的数字和比它大的数字之间，如果没有比当前数字小的数字，则将当前数字排在最前方；依此类推，不断取出未进行排序的数字，与排好序的数字进行比较，并插入相应的位置，直到将一组数字全部按从小到大实现排序为止。

【例8.12】插入法从小到大排序（**实例位置：资源包\TM\sl\08\12**）

声明一个整型数组和两个整型变量，其中整型数组用于存储用户输入的数字，一个整型变量作为两个元素交换时的中间变量，一个用于记录数组元素的位置，然后通过双层循环进行交换法排序，最后将排好序的数组输出。

```
#include<stdio.h>
int main()
{
    int i;
    int a[10];
    int iTemp;
    int iPos;
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                          /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }

    /*采用插入法使数组元素从小到大排序*/
    for(i=1;i<10;i++)                          /*外层循环元素下标为 1～9，表示后 9 个数组元素*/
    {
        iTemp = a[i];                          /*设置插入值*/
        iPos = i-1;
        while((iPos>=0) && (iTemp<a[iPos]))    /*内层循环，寻找插入值的位置*/
        {
            a[iPos+1] = a[iPos];               /*插入数值*/
            iPos--;
        }
        a[iPos+1] = iTemp;
    }
    printf("排序结果如下：\n");
    /*输出数组*/
    for(i=0;i<10;i++)
    {
        printf("%d\t",a[i]);                   /*用制表位分隔数据*/
        if(i == 4)                             /*如果是第 5 个元素*/
            printf("\n");                      /*输出换行*/
    }
    printf("\n");
    return 0;                                  /*程序结束*/
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）设置一个嵌套循环，第一层循环为后9个数组元素，将第二个元素赋值给中间变量，并记录前一个数组元素的下标位置。在第二层循环中，首先要判断是否符合循环的条件，允许循环的条件是记录的下标位置必须大于等于第一个数组元素的下标位置，并且中间变量的值小于之前设置下标位置的数组元素，如果满足循环条件，则将设置下标位置的数组元素赋值给当前的数组元素，然后将记录的数组元素下标位置向前移动一位，继续进行循环判断。内层循环结束以后，将中间变量中保存的数值赋

值给当前记录的下标位置之后的数组元素，继续进行外层循环，将数组中下一个数组元素赋值给中间变量，再通过内层循环进行排序，依此类推，直到循环结束，就将数组元素按照从小到大的顺序重新排列了。

（3）循环输出数组中的元素，并在输出5个元素以后进行换行，在下一行输出后面的5个元素。

运行程序，显示效果如图8.13所示。

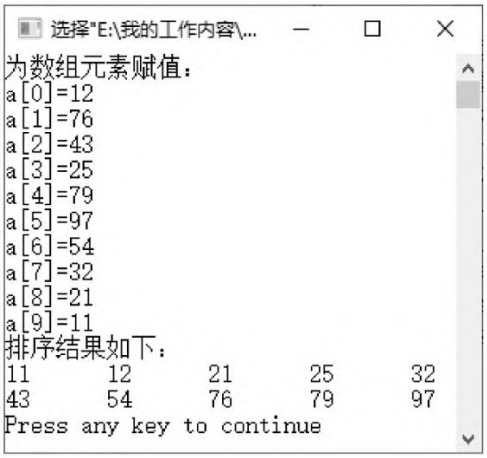


图8.13　插入法排序

8.5.5　折半法排序



折半法排序又称为快速排序，其基本原理为：选择一个中间值（在程序中使用数组中间元素的值），然后把比中间值小的元素放在左边，比中间值大的元素放在右边（具体的实现是从两边查找，找到一对后进行交换），然后再对左右两边分别递归使用折半法排序过程。

下面以数字9、6、15、4、2为例，对这几个数字使用折半法从小到大排序。每次排序后的顺序如表8.6所示。

表8.6　折半法排序（从小到大）

排 序 过 程	元素【0】	元素【1】	元素【2】	元素【3】	元素【4】
起始值	9	6	15	4	2
第 1 次	9	6	2	4	15
第 2 次	4	6	2	9	15
第 3 次	4	2	6	9	15
第 4 次	2	4	6	9	15
排序结果	2	4	6	9	15

可以发现，在第一次排序过程中，首先获取数组中间元素的值15，从左右两侧分别取出数组元素与中间值进行比较。如果左侧取出的值比中间值小，则取下一个数组元素与中间值进行比较；如果左侧取出的值比中间值大，则交换两个互相比较的数组元素值。右侧的比较正好与左侧相反，当右侧取出的值比中间值大时，取前一个数组元素的值与中间值进行比较；如果右侧取出的值比中间值小，则交换两个互相比较的数组元素值。当中间值两侧的数据都比较一遍以后，左侧以第一个元素为起点，以中间值的元素为终点，用上面的方法继续进行比较；右侧则以中间值的元素为起点，以数组最后一个元素为终点，用上述的方法继续进行比较。当比较完成以后，继续以折半的方式进行比较，直到将一组数字全部按从小到大的顺序排序为止。

注意

折半法排序中会用到函数递归调用（即函数调用自身），这部分内容将在第9章中介绍，这里了解即可。读者也可以参考后面的内容提前进行学习。

【例8.13】折半法从小到大排序（实例位置：资源包\TM\sl\08\13）

在本实例中声明了一个整型数组，用于存储用户输入的数字，再定义一个函数，用于对数组元素进行排序，最后将排序好的数组输出。

```
#include<stdio.h>
void CelerityRun(int left, int right, int array[]);          /*声明函数*/

int main()
{
    int i;
    int a[10];
    printf("为数组元素赋值：\n");
    for(i=0;i<10;i++)                                        /*输入 10 个数，为数组元素赋值*/
    {
        printf("a[%d]=",i);
        scanf("%d", &a[i]);
    }
    CelerityRun(0,9,a);                                      /*调用折半排序函数*/
    printf("排序结果如下：\n");
    for(i=0;i<10;i++)                                        /*输出数组*/
    {
        printf("%d\t",a[i]);                                 /*用制表位分隔数据*/
        if(i == 4)                                           /*如果是第 5 个元素*/
            printf("\n");                                    /*输出换行*/
    }
    printf("\n");
    return 0;                                                /*程序结束*/
}

void CelerityRun(int left, int right, int array[])           /*定义折半排序函数*/
{
    int i,j;
    int middle,iTemp;
    i = left;
    j = right;
    middle = array[(left+right)/2];                          /*求中间值*/
    do
    {
        while((array[i]<middle) && (i<right))                /*从左查找小于中间值的数*/
            i++;
        while((array[j]>middle) && (j>left))                 /*从右查找大于中间值的数*/
            j--;
        if(i<=j)                                             /*如果找到一对值*/
        {
            iTemp = array[i];                                /*交换两个值*/
            array[i] = array[j];
            array[j] = iTemp;
            i++;
            j--;
        }
    }while(i<=j);                                            /*如果两边的下标交错，就停止（完成一次）*/
    if(left<j)                                               /*递归左半边*/
        CelerityRun(left,j,array);
    if(right>i)                                              /*递归右半边*/
        CelerityRun(i,right,array);
}
```

（1）声明一个整型数组，并通过键盘输入为数组元素赋值。

（2）定义折半排序函数CelerityRun，用于对数组元素进行排序，3个参数分别表示递归调用时数组最开始元素的下标、最后元素的下标以及要排序的数组。声明两个整型变量，作为控制排序算法循环的条件，分别将两个参数赋值给变量i和j，i表示左侧下标，j表示右侧下标。首先使用do…while语句设计外层循环，条件为i≤j，表示如果两边的下标交错，就停止循环。内层的两个循环分别用来比较中间值两侧的数组元素，当左侧的数值小于中间值时，取下一个元素与中间值进行比较，否则退出第一个内层循环；当右侧的数值大于中间值时，取前一个元素与中间值进行比较，否则退出第二个内层循环。然后判断i的值是否小于等于j，如果是，则交换以i和j为下标的两个元素值，继续进行外层循环。当外层循环结束以后，以数组第一个元素到以j为下标的元素为参数递归调用该函数，同时，以i为下标的数组元素到数组最后一个参数也作为参数递归调用该函数。依此类推，直到将数组元素按照从小到大的顺序重新排列为止。

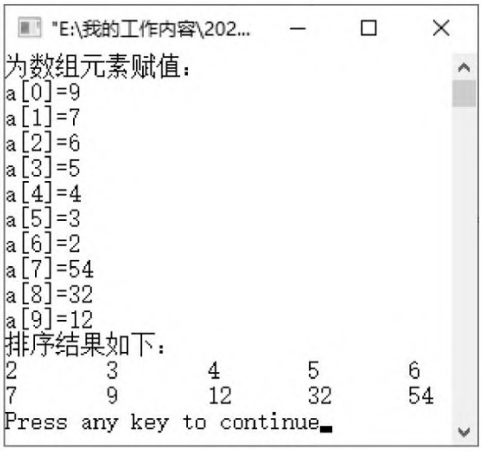


图8.14　折半法排序

（3）循环输出数组中的元素，并在输出5个元素以后进行换行，在下一行输出后面的5个元素。

运行程序，显示效果如图8.14所示。

8.5.6　排序算法的比较



前面介绍了5种排序算法，在具体应用时应该怎么选择呢？下面对5种排序算法的擅长方向进行总结。

- ☑ 选择法排序：简单、容易实现，适用于数量较小的排序。排序过程中需要进行n（n-1）/2次比较，互相交换n-1次。
- ☑ 冒泡法排序：相对稳定的排序方法，当待排序列有序时，效果比较好。最好的情况是正序，只要比较一次即可；最坏的情况是逆序，需要比较n^2次。
- ☑ 交换法排序：和冒泡法排序类似，正序时最快，逆序时最慢，排列有序数据时效果最好。
- ☑ 插入法排序：要经过n-1次插入过程，如果数据恰好位于插入序列的最后端，则不需要移动数据，可节省时间。因此，若原始数据基本有序，具有较快的运算速度。
- ☑ 折半法排序：n越大，速度越快。n很小时，比其他排序算法都慢。折半法排序是不稳定的，对应有相同关键字的记录，排序后的结果可能会颠倒次序。

总之，插入法、冒泡法、交换法排序的速度较慢，但当参加排序的序列局部或整体有序时，这种排序能达到较快的速度；在这种情况下，折半法排序反而会显得速度慢了。当n较小，对稳定性不作要求时，宜选用选择法排序；对稳定性有要求时，宜选用插入法或冒泡法排序。

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练7：电视剧收视率排名　利用交换排序法将以下电视剧收视率从高到低排序。

《Give up，hold on to me》　　收视率：1.4%

《The private dishes of the husbands》　　收视率：1.343%

《My father-in-law will do martiaiarts》 收视率：0.92%

《North Canton still believe in love》 收视率：0.862%

《Impossible task》 收视率：0.553%

《Sparrow》 收视率：0.411%

《East of dream Avenue》 收视率：0.164%

《The prodigal son of the new frontier town》 收视率：0.259%

《Distant distance》 收视率：0.394%

《Music legend》 收视率：0.562%

程序运行结果如下：

```
1.4000  1.3930  0.9200  0.8620  0.5620
0.5530  0.4110  0.3940  0.2590  0.1640
```

训练8：公司股票排名 分析10家公司的股票数据，判断哪家更值得投资。在本实例中，声明了一个整型数组和一个整型变量，其中整型数组用于存储用户输入的数字，而整型变量则作为两个元素交换时的中间变量，然后通过双层循环进行冒泡法排序，最后将排序好的数组进行输出。输出结果如下：

```
各公司股票数据如下：
```

```
a[0]=546789
a[1]=543879
a[2]=234780
a[3]=357698
a[4]=345275
a[5]=875422
a[6]=875606
a[7]=567548
a[8]=459078
a[9]=438997
234780  345275  357698  438997  459078
543879  546789  567548  875422  875606
```

8.6 数组应用



本节将运用数组知识，通过3个实例来体验一下实际开发中常会遇到的一些问题。

8.6.1 反转输出字符串

【例8.14】反转输出字符串（**实例位置：资源包\TM\sl\08\14**）

以字符串mrsoft为例，其反转的结果为tfosrm。其算法实现过程如图8.15所示。定义两个字符数组，一个表示源字符串，另一个表示反转后的目标字符串。在源字符串中从第一个字符开始遍历，将第一个字符赋给目标字符串的最后一个字符，将第二个字符赋给目标字符串的倒数第二个字符，依此类推，这样就实现了字符串的反转。

```
#include<stdio.h>
int main()
{
    int i;
    char String[7] = {"mrsoft"};                    /*定义源字符串*/
    char Reverse[7] = {0};                          /*定义反转字符串*/
    int size;
    size = sizeof(String);                          /*计算源字符串长度*/
    for(i=0;i<6;i++)                                /*循环读取字符*/
    {
        Reverse[size-i-2] = String[i];              /*源字符串倒序存入反转字符串*/
    }
    printf("输出源字符串：%s\n",String);             /*输出源字符串*/
    printf("输出目标字符串：%s\n",Reverse);          /*输出目标字符串*/
    return 0;                                       /*程序结束*/
}
```

运行程序，字符串反转输出的效果如图8.16所示。

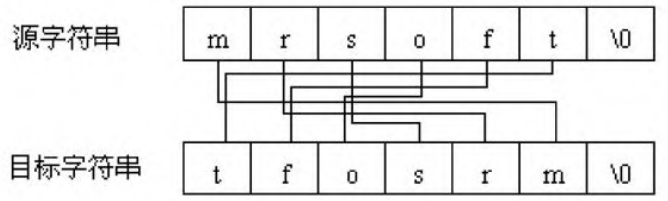


图8.15　字符串反转示意图



图8.16　反转输出字符串

8.6.2　输出系统日期和时间

【例8.15】输出系统日期和时间（实例位置：资源包\TM\sl\08\15）

设计一个程序，当用户输入0时显示帮助信息，输入1时显示系统日期，输入2时显示系统时间，输入3时退出系统。

要实现上述功能，需要解决两个问题：一是要不断地保持程序运行，等待用户输入命令，防止main函数结束；二是要获取系统日期和时间。

第一个问题可使用一个无限循环语句来实现，在循环语句中等待用户输入，如果用户输入的是3，则终止循环，结束应用程序。第二个问题需要使用时间函数time和localtime来实现。

在main函数中将各个控制命令保存在数组中，然后使用while语句设计一个无限循环。在循环中让用户输入命令，并判断用户输入的命令是否和数组中存储的命令相同。如果相同，执行相应的语句。

```
#include<stdio.h>
#include<time.h>

int main()
{
    int command[4] = {0,1,2,3};                          /*定义一个数组*/
    int num;
    struct tm *sysTime;
    printf("如需帮助可输入数字 0！ \n");                    /*输出提示信息*/
    printf("请输入命令符：\n");

    while (1)                                            /*通过 while 循环，保证始终等待用户输入*/
    {
        scanf("%d", &num);                               /*获得输入数字*/
        /*判断用于输入的字符*/
        if(command[0] == num)                            /*如果用户输入数字 0*/
        {
            /*输出帮助信息*/
            printf("输入数字 1 显示系统日期，输入数字 2 显示系统时间，输入数字 3 退出系统！ \n");
        }
        else if(command[1] == num)                       /*如果用户输入数字 1*/
        {
            time_t nowTime;
            time(&nowTime);                              /*获取系统日期*/
            sysTime= localtime(&nowTime);                /*转换为系统日期*/
            printf("系统日期：%d-%d-%d \n",1900 + sysTime->tm_year,sysTime->tm_mon + 1,sysTime->
                    tm_mday);                            /*输出信息*/
        }
        else if(command[2] == num)                       /*如果用户输入数字 2*/
        {
            time_t nowTime;
            time(&nowTime);                              /*获取系统时间*/
            sysTime = localtime(&nowTime);               /*转换为系统时间*/
            printf("系统时间：%d:%d:%d \n",sysTime->tm_hour ,sysTime->tm_min ,sysTime-> tm_sec);
                                                         /*输出信息*/
        }
        else if(command[3] == num)                       /*如果用户输入数字 3*/
        {
            return 0;                                    /*退出系统*/
        }
        printf("请输入命令符：\n");                      /*输出提示信息*/
    }
    return 0;                                            /*程序结束*/
}
```

运行程序，输出系统日期和时间的实例效果如图8.17所示。



图8.17　输出系统日期和时间

8.6.3　字符串的加密和解密

设计应用程序时，为了防止一些敏感信息泄漏，通常需要对这些信息进行加密。以用户登录密码为例，如果密码以明文形式存储在数据表中，很容易被人发现；如果密码以密文形式存储，即使别人从数据表中发现了密码，这也是加密之后的密码，根本不能够使用，因此能提高系统的安全性。

【例8.16】字符串的加密和解密（实例位置：资源包\TM\sl\08\16）

设计一个加密和解密算法，对一个指定的字符串加密后，再利用解密函数对密文解密，显示明文信息。加密的方式是将字符串中每个字符加上它在字符串中的位置和一个偏移值5。以字符串“mrsoft”为例，第一个字符m在字符串中的位置为0，那么它对应的密文是'm'+0+5，即r。

在main函数中使用while语句设计一个无限循环，并声明两个字符数组，用来保存明文和密文字符串。在首次循环中，要求用户输入字符串，将明文加密成密文，之后的操作则是根据用户输入的命令字符进行判断：输入1加密新的明文，输入2对之前加密的密文进行解密，输入3退出系统。

```
#include<stdio.h>
#include<string.h>

int main()
{
    int result = 1;
    int i;
    int count = 0;
    char Text[128] = {'\0'};                                    /*定义一个明文字符数组*/
    char cryptograph[128] = {'\0'};                             /*定义一个密文字符数组*/
    while (1)
    {
        if(result == 1)                                         /*如果用户输入 1，加密明文*/
        {
            printf("请输入要加密的明文：\n");                    /*输出提示信息*/
            scanf("%s", &Text);                                 /*获取输入的明文*/
            count = strlen(Text);
            for(i=0; i<count; i++)                              /*遍历明文*/
            {
                cryptograph[i] = Text[i] + i + 5;               /*设置加密字符*/
            }
            cryptograph[i] = '\0';                              /*设置字符串结束标记*/
            printf("加密后的密文是：%s\n",cryptograph);          /*输出密文信息*/
        }
        else if(result == 2)                                    /*如果用户输入 2，解密字符串*/
        {
            count = strlen(Text);
            for(i=0; i<count; i++)                              /*遍历密文字符串*/
            {
                Text[i] = cryptograph[i] - i - 5;               /*设置解密字符*/
            }
            Text[i] = '\0';                                     /*设置字符串结束标记*/
            printf("解密后的明文是：%s\n",Text);                 /*输出明文信息*/
        }
        else if(result == 3)                                    /*如果用户输入 3，退出系统*/
        {
            break;                                              /*跳出循环*/
        }
        else
        {
            printf("请输入命令符：\n");                          /*输出提示信息*/
        }
        printf("输入 1 加密新的明文，输入 2 对刚加密的密文进行解密，输入 3 退出系统：\n");
        printf("请输入命令符：\n");                              /*输出提示信息，提示用户输入指令*/
        scanf("%d", &result);                                   /*获取输入的命令字符*/
    }
    return 0;                                                   /*程序结束*/
}
```

运行程序，字符串的加密和解密效果如图8.18所示。

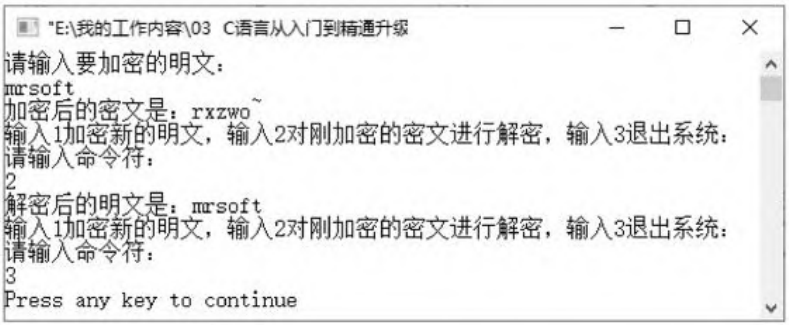


图8.18　字符串的加密和解密

编程训练（答案位置：资源包\TM\sl\08\编程训练\）

训练9：升序排列字符串　将已按升序排好的字符串a和字符串b按升序归并到字符串c中，并输出。运行结果如下：

```
请输入排序好的 a:
abcegiklnt
请输入排序好的 b:
dfmopsyz
输出排序好的 c:
abcdefgiklmnopstyz
```

训练10：竞选班长　班级竞选班长，共有3个候选人。输入参加选举的人数及每个人推举的候选人编号，输出3个候选人最终的得票数及无效选票数。输出结果如下：

```
请输入参加投票人数:
15
输入每个人推举的候选人（输入1、2、3即可）
1 2 3 2 2 3 3 5 6 1 1 2 2 3 1
结果如下:
候选人1:4
候选人2:5
候选人3:4
无效票数:2
```

8.7　实践与练习

（答案位置：资源包\TM\sl\08\实践与练习\）

综合练习1：输出游戏角色　“王者荣耀”游戏中有很多英雄，包括法师、战士、坦克、刺客、射手和辅助。用数组输出不同类别的英雄，输出结果如下：

```
“王者荣耀”游戏角色:
====坦克:====
苏烈,刘邦,钟馗,张飞,牛魔,程咬金,白起,刘禅,庄周,项羽,廉颇,巨灵神,安禄山,猪八戒
====战士:====
狂铁,裴擒虎,铠,孙悟空,哪吒,杨戬,橘右京,亚瑟,雅典娜,夏侯惇,关羽,吕布,韩信,老夫子,达摩,典韦,曹操,钟无艳,墨
赵云,刑天,龙 且
====刺客:====
百里玄策,庞统,花木兰,阿轲（荆轲）,不知火舞,李白,娜可露露,兰陵王,露娜,韩信,宫本武藏,盖聂,红拂
====法师:====
杨玉环,弈星,女娲,周瑜,鬼谷子,半月,干将莫邪,东皇太一,大乔,诸葛亮,貂蝉,张良,安琪拉,不知火舞,姜子牙,武则
王昭君,甄姬,扁鹊,高渐离,嬴政,妲己,小乔
====射手:====
公孙离,百里守约,后羿,刘备,黄忠,马可波罗,成吉思汗,虞姬,李元芳,艾琳,狄仁杰,鲁班七号,孙尚香
====辅助:====
明世隐,梦奇,孙膑,太乙真人,蔡文姬
```

综合练习2：十二星座速配　巨蟹座和十二星座的爱情匹配分值（星座名／速配值）：白羊座/50，；金牛座/90；双子座/70；巨蟹座/80；狮子座/75；处女座/89；天枰座/55；天蝎座/100；射手座/40；摩羯座/60；水瓶座/45；双鱼座/99。利用插入排序法，将12个速配值从小到大排序，运行结果如下：

```
巨蟹座与哪个星座匹配，匹配分数由低到高如下：
40      45      50      55      60
70      75      80      89      90
```

综合练习3：统计数字出现的次数　用户输入0~9范围内任意10个数字，统计各数字出现的次数。运行结果如下：

```
请输入 10 个 0~9 的数组元素：
1 5 6 8 2 1 4 5 8 5
--------------------------------
|   0 出现的次数 0      |
|   1 出现的次数 2      |
|   2 出现的次数 1      |
|   3 出现的次数 0      |
|   4 出现的次数 1      |
|   5 出现的次数 3      |
|   6 出现的次数 1      |
|   7 出现的次数 0      |
|   8 出现的次数 2      |
|   9 出现的次数 0      |
--------------------------------
```

综合练习4：成绩单公布　某班期中考试后，根据“总成绩＝智育成绩*60%+德育成绩*30%+体育成绩*10%”的计算方式，将班级前12名同学的总成绩进行排名。输入每个人的总成绩，输出从高到低排序后的成绩。运行结果如下：

```
输入成绩：
a[0]=95
a[1]=85
a[2]=90
a[3]=77
a[4]=88
a[5]=100
a[6]=96
a[7]=93
a[8]=80
a[9]=79
a[10]=89
a[11]=92
成绩排名如下：
100     96      95      93      92
90      89      88      85      80      79      77
```

综合练习5：平安夜卖苹果　平安夜，父亲推出一车苹果（共2520个），分给6个儿子卖。父亲先按事先写在纸上的数字把苹果分完，每个人拿到的苹果数量都不同。然后他说：“老大，把你的苹果分1/8给老二。老二连同原来的苹果，分1/7给老三。老三连同原来的苹果，分1/6给老四。依此类推，最后老六拿到后，连同原来的苹果分1/3给老大。这

样，你们每个人手中的苹果就一样多了。”那么，兄弟6人原来各有多少苹果呢？编写程序计算一下。输出结果如下：

```
x[1]=240
x[2]=460
x[3]=434
x[4]=441
x[5]=455
x[6]=490
```

综合练习6：谁被@了　QQ群或微信群里，要是找某个人有急事，就会@他。编写程序，输出被@的人员列表。输出结果如下：

```
被@的列表：
明日科技　你被@了
扎克伯格　你被@了
比尔盖茨　你被@了
```

第9章
函数

大型程序一般会被分为若干个程序模块，每个模块实现一个特定功能。C语言中，由函数实现子程序，由子程序实现模块功能。

本章致力于使读者了解函数的概念，掌握函数的定义及调用方式；了解内部函数和外部函数的作用范围，能区分局部变量和全局变量的不同；最后能将函数应用于程序中，将程序分成不同的功能模块。

本章的知识架构及重难点如下：



9.1 函数概述



构成C程序的基本单元是函数，函数中包含着程序的可执行代码。

每个C程序的入口和出口都位于main函数中，但并不需要把所有指令都放在main函数中。一般的做法是将程序划分成若干个模块，每个模块完成一部分功能，不同的程序模块可以由不同的人来完成，从而提高软件开发的效率。

这就好比是盖楼房，一栋摩天大楼是不可能靠一个人完成的，而要靠多部门、多工种之间协力完成。通常是有一个总工程师，在他的指挥下，有部门运输建筑材料，有部门建造楼房主体，还有部门粉刷内外墙涂料。编写程序的道理与盖楼是一样的，主函数就像总工程师一样，控制着整体程序的推进和执行，其中定义的其他函数就好比参与盖楼的多个部门或工种，他们要通过某种调度（函数调用）才能完成特定的功能。

主函数可以调用其他函数，其他函数间也可以相互调用。函数可以有参数和返回值，通过它们实现数据间的传递。在主函数中调用其他函数，这些函数执行完毕之后会返回main函数中。通常把这些被调用的函数称为下层函数。函数调用发生时，立即执行被调用的函数，而调用者则进入等待的状态，直到被调用函数执行完毕。

【例9.1】编写3个函数：做饭，钓鱼，写诗（**实例位置：资源包\TM\sl\09\01**）

在本实例中，定义3个函数来完成做饭、钓鱼、写诗等特定的功能，然后在主函数中调用它们。为了简化函数的功能，这里只让其输出一条提示信息。读者可通过本实例对函数有一个直观的认识。

```
#include<stdio.h>                    /*包含头文件*/
void Cook();                         /*声明 Cook 函数*/
void Fish();                         /*声明 Fish 函数*/
void Poem();                         /*声明 Poem 函数*/
int main()                           /*主函数 main*/
{
    Cook();                          /*调用 Cook 函数*/
    Fish();                          /*调用 Fish 函数*/
    Poem();                          /*调用 Poem 函数*/
    return 0;                        /*程序结束*/
}
void Cook()                          /*自定义 Cook 函数*/
{
    printf("会做饭\n");
}

void Fish()                          /*自定义 Fish 函数*/
{
    printf("会钓鱼\n");
}

void Poem()                          /*自定义 Poem 函数*/
{
    printf("会写诗\n");
}
```

在分析本实例之前，我们先来了解一下什么是C程序源文件、库函数和用户自定义函数。。

☑ 源文件：由一个或者多个函数组成。C语言以源程序为单位进行编译，而不是以函数为单位进行编译。

☑ 库函数：由C语言系统提供，用户无须定义，调用前也不必做类型说明，但需要在程序开始部分包含有该函数原型的头文件。例如，要使用能在控制台显示信息的printf函数，需在程序开始时包含stdio.h头文件；要使用字符串操作函数strlen、strcmp等时，需在程序开始时包含string.h头文件。

☑ 用户自定义函数：用户编写的用来实现特定功能的函数。例如，Cook、Fish和Poem函数都是自定义函数。

在本例程序中，首先包含了stdio.h头文件，然后声明了3个自定义函数，最后在主函数main中调用了这3个函数。在主函数main外，可以看到这3个函数的定义。

运行程序，显示效果如图9.1所示。

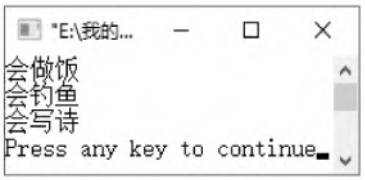


图9.1　在主函数中调用其他函数

编程训练（答案位置：资源包\TM\sl\09\编程训练\）

训练1：盖楼房　编写3个函数，分别实现搬运建筑材料、建造楼房主体、粉刷内外墙功能，并在主函数中调用这3个函数。输出结果如下：

```
执行搬运功能
执行建造功能
执行粉刷功能
```

训练2：写情书　定义一个函数，内容为一封情书，在主函数中调用该函数，将情书内容展示出来。运行结果如下：

```
人生最美好的是相遇
我一生最奢侈的事
就是途中与你相遇
然后相濡以沫，共闻花香，有生之年
只诉温暖不言殇，倾心相遇，安暖相陪
```

9.2　函数的定义和声明

C语言的库函数可以直接调用，如printf输出函数。而自定义函数则必须由用户进行定义，确定其要实现的功能，这样才能被其他函数调用。

9.2.1　函数的定义



一个函数应包括函数头和函数体。定义一个函数的语法格式如下：

```
返回值类型 函数名(参数列表)
{
    函数体                                    /*函数实现特定功能的过程*/
}
```

首先来看一段代码，然后通过分解，了解函数的构成。

```
int AddTwoNumber(int iNum1,int iNum2)          /*函数头部分*/
{
    int result;                                /*定义整型变量*/
    result = iNum1+iNum2;                      /*进行加法操作*/
    return result;                             /*返回操作结果，结束*/
}
```

函数头

函数体

1．函数头

函数头是函数的入口，标志着一段函数代码的开始。函数头包括返回值类型、函数名和参数列表3个部分，如图9.2所示。

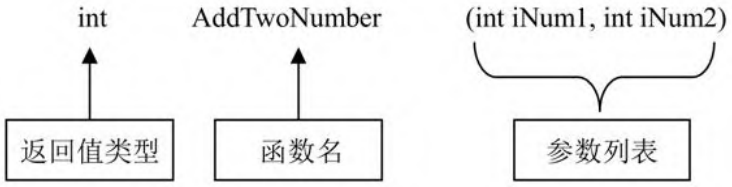


图9.2　函数头组成

☑ 返回值类型：函数返回值的类型，必须是C语言中的某个数据类型。这里函数的返回值将是一个int型值。

☑ 函数名：函数的标识符，在一个C程序中应保持唯一。因为是标识符，所以函数名要遵守标识符命名规则。这里，函数名是AddTwoNumber，可以推测出该函数的功能是两数相加求和。

☑ 参数列表：调用函数时，用于将主调函数中的实际参数复制到该列表对应的形式参数中。可以没有参数，也可以有多个参数。这里定义了两个int变量，表示要相加的两个数。

2．函数体

函数体位于函数头的下方位置，由一对大括号括起来，大括号决定了函数体的范围。函数要实现的特定功能，都是在函数体部分通过代码语句完成的，最后通过return语句返回实现的结果。

在上面的代码中，函数体内首先定义了一个int型变量，用来保存加法的计算结果，之后利用传递进来的参数进行加法操作，并将结果保存在result变量中，最后函数要将所得到的结果进行返回。通过这些语句的操作，实现了求解两数和的特定功能。

在定义函数时会出现以下几种特殊的情况。

☑ 无参函数：没有参数列表的函数。如例9.1中的Cook、Fish、Poem都是无参函数。

☑ 空函数：没有任何内容，也没有什么实际功能的函数。空函数的形式如下：

```
类型说明符  函数名()
{
}
```

实际开发中，有时某个函数还未编好，或者后续要拓展某个函数，这时就会先用一个空函数代替，先占个位置，待后续时机成熟再用编好的函数取代它。

注意，C语言中，函数的定义是互相平行、独立的。也就是说，函数体内不能再包含其他函数的定义。例如，下面的代码是错误的：

```
int main()
{
    void Display()                                  /*错误！不能在函数体内定义另一个函数*/
    {
        printf("I want to show the Nesting function");
    }
    return 0;
}
```

这里，主函数main中定义了一个Display函数，目的是输出一句提示。由于C语言不允许进行嵌套定义，因此编译时会出现如图9.3所示的

错误提示。

```
error C2143: syntax error : missing ';' before '{'
```

图9.3　错误提示

9.2.2　函数的声明



在程序中编写函数时，要先对函数进行声明，再对函数进行定义。函数定义是为了让编译器知道函数的功能，而函数声明是为了让编译器预先知道有这么一个函数，以及函数的名称、参数、返回值类型等信息。

函数声明的一般形式如下:

```
返回值类型  函数名(参数列表);
```

要注意的是，函数声明语句的最后要用分号“;”作为结尾。例如，声明一个函数的代码如下:

```
Int ShowNumber(int iNumber);
```

【例9.2】交换两个数值（实例位置：资源包\TM\sl\09\02）

通过本实例了解函数声明与函数定义的位置，及其在程序中的作用。

```
#include<stdio.h>
void exchange(int a,int b);                          /*声明 exchange 函数*/
int main()
{
    int a=3,b=4;                                     /*定义整型变量*/
    printf("交换之前的值 a=%d,b=%d\n",a,b);          /*提示信息*/
    exchange(a,b);                                   /*调用 exchange 函数*/
    return 0;
}
void exchange(int a,int b)                           /*定义 exchange 函数，用于交换两个数*/
{
    int c;                                           /*交换数值*/
    c=a;
    a=b;
    b=c;
    printf("交换后的值 a=%d,b=%d\n",a,b);            /*输出交换之后的数据*/
}
```

（1）观察上面的程序，可以看到在main函数的开头先进行了exchange函数的声明，声明的作用是告知其函数将在后面进行定义。

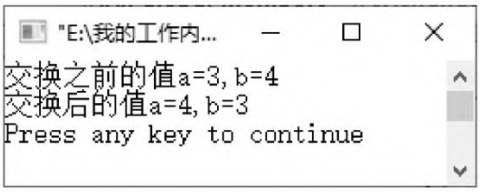


图9.4　函数的定义与声明

（2）在main函数体中，首先定义两个整型变量a、b，之后输出一条提示消息，然后调用exchange函数。

（3）在main函数的定义之后可以看到exchange函数的定义，功能是实现两个数的数值互换。

运行程序，结果如图9.4所示。

注意

如果先定义函数，再调用函数，则不再需要进行函数声明，此时函数定义已包含了函数声明的作用。

编程训练（答案位置：资源包\TM\sl\09\编程训练\）

训练3：输出谚语　编写一个程序，输出“最大的挑战和突破在于用人，而用人最大的突破是信任人”。运行效果如下：

```
最大的挑战和突破在于用人，而用人最大的突破是信任人
```

训练4：打印新年菜单　春节是中国最重要的节日，家家户户都会张灯结彩，丰盛的年夜饭更是必不可少的。定义一个函数，打印2021年新年菜单，再在主函数中调用函数。运行结果如下：

```
-----新年菜单如下-------
 1.凉菜
 2.红烧鱼
 3.煮虾
 4.酱猪蹄
 5.红烧排骨
 6.孜然牛肉
 7.木须柿子
 8.水煮肉片
------------------------
```

9.3　函数参数

多数情况下，主调函数和被调函数之间存在着数据传递关系，这种数据传递是通过函数参数来实现的。函数参数的作用是传递数据给函数使用，函数利用接收到的数据进行具体的操作处理。

9.3.1　形式参数与实际参数



函数的参数分为两种：形式参数和实际参数。同为参数，读者要仔细体会其中的区别。

1．形式参数

声明和定义函数时，函数名后面括号中的参数称为形式参数。这些参数只是定义了类型，在实际参数传入前并没有实际意义，因此叫作形式参数，简称形参。

2．实际参数

调用函数时，函数名后面括号中的参数称为实际参数。调用函数的过程就是真正使用这个函数的过程，此时调用者会传递一些要实际参与运算的参数给被调用函数，这些实际参与运算的参数就是实际参数，简称实参。

下面来看一段代码，加深对形参和实参的理解。

```
int Minus(int iNumber1,int iNumber2)          /*定义 Minus 函数，有两个 int 型参数*/
{
    int iResult;                              /*定义整型变量*/
    iResult=iNumber1-iNumber2;                /*两个 int 型参数相减*/
    return iResult;                           /*返回计算结果*/
}
int main()
{   …
    iResult=Minus(9,4);                       /*调用 Minus 函数，传入实参 9,4，函数执行 9-4 运算*/
    …
}
```

上述代码定义了一个Minus函数，功能是两数相减。这里，iNumber1和iNumber2是形式参数，表示相减的两个整型数，没有具体值（等待实参传入）。主函数中，通过“iResult=Minus（9，4）;”语句调用Minus函数，这里9和4为实际参数，调用函数后，将用4代替iNumber1，用9代替iNumber2，因此Minus函数实际执行的运算的是9−4。

注意，函数参数可以是常量、变量、数组、指针等，也可以是表达式。

9.3.2　数组作函数参数



1．数组元素作为函数参数

数组元素作为函数实参，与普通变量作为函数实参一样，是单向的值传递。

【例9.3】输出数组元素（实例位置：资源包\TM\sl\09\03）

定义一个数组并为其赋值，然后将数组元素作为函数实参进行传递。自定义函数体中，形参得到实参传递的数值后，将其显示输出。

```
#include<stdio.h>
void ShowMember(int iMember);                    /*声明 ShowMember 函数，形参为一个整型数*/

int main()
{
    int iCount[10];                              /*定义一个整型数组*/
    int i;                                       /*定义整型变量*/
    for(i=0;i<10;i++)                            /*for 循环，对数组元素依次赋值*/
    {
        iCount[i]=i;
    }
    for(i=0;i<10;i++)                            /*循环调用 ShowMember 函数*/
    {
        ShowMember(iCount[i]);                   /*函数实参为数组元素*/
    }
    return 0;
}

void ShowMember(int iMember)                     /*自定义 ShowMember 函数，形参为一个整型数*/
{
    printf("Show the member is%d\n",iMember);    /*输出数据*/
}
```

（1）首先进行函数声明，在主函数main中定义一个整型数组和一个整型变量i。

（2）使用for循环语句对数组中的元素依次赋值，在这里，变量i既是循环条件，也是引用数组元素的下标。

（3）通过for循环语句调用ShowMember函数，显示数据。

运行程序，显示效果如图9.5所示。

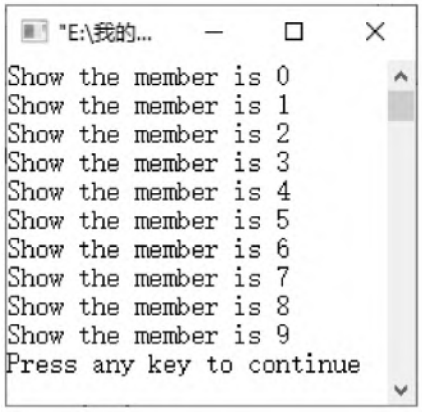


图9.5　数组元素作为函数参数

2．数组名作为函数参数

C语言中，数组名表示的是数组中第一个元素的地址。因此，当数组名作为函数实参时，传递的是数组的地址。这点和数组元素做实参时是不一样的，注意体会其不同。

【例9.4】数组名作为函数参数（实例位置：资源包\TM\sl\09\04）

在本实例中，使用数组名作为函数的实参和形参，实现数组的赋值和输出。

```
#include<stdio.h>
void Evaluate(int iArrayName[10]);                    /*声明赋值函数，形参为一个数组名*/
void Display(int iArrayName[10]);                     /*声明输出函数，形参为一个数组名*/
int main()
{
    int iArray[10];                                   /*定义一个具有 10 个元素的整型数组*/
    Evaluate(iArray);                                 /*调用函数，实现数组元素赋值，数组名为实参*/
    Display(iArray);                                  /*调用函数，实现数组元素输出，数组名为实参*/
    return 0;
}
/*////////////////////////////////////////////////////////////////////////////////////////////*/
/*                                  进行数组元素的输出                                        */
/*////////////////////////////////////////////////////////////////////////////////////////////*/
void Display(int iArrayName[10])                      /*定义输出函数，形参为一个数组*/
{
    int i;                                            /*定义整型变量*/
    for(i=0;i<10;i++)                                 /*在循环语句中执行数组输出*/
    {
        printf("the member number is %d\n",iArrayName[i]);
    }
}
/*////////////////////////////////////////////////////////////////////////////////////////////*/
/*                                  进行数组元素的赋值                                        */
/*////////////////////////////////////////////////////////////////////////////////////////////*/
void Evaluate(int iArrayName[10])                     /*定义赋值函数，形参为一个数组*/
{
    int i;                                            /*定义整型变量*/
    for(i=0;i<10;i++)                                 /*在循环语句中执行数组赋值*/
    {
        iArrayName[i]=i;
    }
}
```

（1）首先对Evaluate函数和Display函数进行声明，在声明语句中可以看到数组名作为形参。

（2）在主函数main中定义一个具有10个元素的整型数组iArray。

（3）调用Evaluate函数，数组名iArray作为函数实参，传递的是数组的地址。在Evaluate函数中，使用数组iArrayName作为形参，接受对应的地址空间，并对数组进行赋值操作。

（4）调用Display函数，将数组输出，可以看到在函数参数中使用的也是数组名称。

运行程序，显示效果如图9.6所示。

3．可变长度数组作为函数参数

数组作为函数参数时，如果未指明数组大小，就属于长度可变的数组作为函数参数。

【例9.5】不使用库函数，实现字符串连接功能（实例位置：资源包\TM\sl\09\05）

在本实例中，不使用编译器提供的库函数，将函数的参数声明为长度可变数组。具体代码如下：

```
#include<stdio.h>                                   /*包含头文件*/
void _strcat(char str1[],char str2[])               /*自定义 strcat 函数，形参为两个长度可变的数组*/
{
    int i,j;                                        /*定义控制变量*/
    for(i=0;str1[i]!='\0';i++);                     /*字符数组 1 中循环*/
        for(j=0;str2[j]!='\0';j++)                  /*字符数组 2 中循环*/
            str1[i+j]=str2[j];                      /*字符串连接*/
    str1[i+j]='\0';                                 /*结束*/
 }
int main()                                          /*主函数 main*/
{
    char str1[100],str2[100];                       /*定义字符数组*/
    printf("请输入字符串 1:\n");                    /*提示信息*/
    gets(str1);                                     /*输入字符串 1*/
    printf("请输入字符串 2:\n");                    /*提示信息*/
    gets(str2);                                     /*输入字符串 2*/
    _strcat(str1,str2);                             /*调用函数连接 2 个字符串*/
    printf("连接之后的字符串是：%s\n",str1);        /*显示连接后的字符串*/
    return 0;                                       /*程序结束*/
}
```

运行程序，显示效果如图9.7所示。

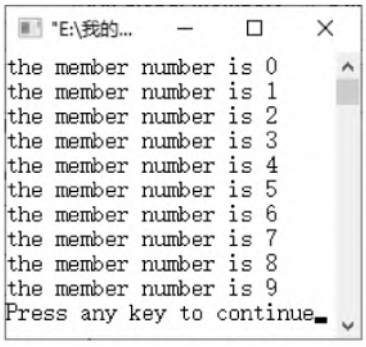


图9.6　数组名作为函数参数

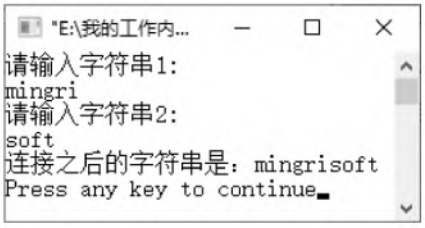


图9.7　可变长度数组作为函数参数

编程训练（答案位置：资源包\TM\sl\09\编程训练\）

训练5：用户登录验证　编写登录函数，函数有两个形参：账号名和密码。如果账号名为“张三”，密码为“123”，则登录成功，否则登录失败。运行结果如下：

```
登录成功
```

训练6：自定义运算函数　编写函数fun（int a[][N]，int n），使数组左下半边角的元素值均乘以n。例如，n=3，a={{1，9，7}，{2，3，8}，{4，5，6}}，运行程序后得到{{3，9，7}，{6，9，8}，{12，15，18}}。运行结果如下：

```
a[0][0]=1
a[0][1]=9
a[0][2]=7
a[1][0]=2
a[1][1]=3
a[1][2]=8
a[2][0]=4
a[2][1]=5
a[2][2]=6
左下半边角元素中的值乘以 n 的结果是：
3       9       7
6       9       8
12      15      18
```

9.4　函数的调用和返回

通过前面的学习，我们已经知道，使用函数的过程就是调用函数的过程。

9.4.1 函数的调用



函数的调用方式有3种，包括语句调用、表达式调用和函数参数调用。

1．语句调用

函数调用作为一个独立语句出现，就称为语句调用。这是最常用的函数调用方式，可以有返回值，也可以没有返回值。例如：

```
Display();                                    /*通过语句调用 Display 函数*/
```

2．表达式调用

当函数调用出现在一个表达式中时，函数必须返回一个确定的值，作为表达式运算的一部分。例如：

```
iResult=iNum3*AddTwoNum(3,5);                 /*函数调用出现在表达式中，返回值参与乘法运算*/
```

这条语句中，函数AddTwoNum的返回值将与iNum3执行乘法，得到的结果赋值给iResult变量。

【例9.6】实现欧姆定律功能（实例位置：资源包\TM\sl\09\06）

在本实例中，定义一个函数，其功能是利用欧姆定律（R=U/I）计算电阻值，并在表达式中调用该函数，使其返回值参与运算。

```
#include<stdio.h>                          /*包含头文件*/
void TwoNum(float iNum1, float iNum2);     /*声明函数，函数进行计算*/
int main()
{
    TwoNum(5,10);                          /*调用函数*/
    return 0;                              /*程序结束*/
}
void TwoNum(float iNum1, float iNum2)      /*定义函数*/
{
    float iTempResult;                     /*定义整型变量*/
    iTempResult=iNum1/iNum2;               /*进行计算，并将结果赋值给 iTempResult*/
    printf("电阻值是%f\n",iTempResult);    /*显示输出电阻值*/
}
```

（1）先对要使用的函数进行声明操作。

（2）在主函数main中，调用TwoNum函数来计算数值5和10的相除。

（3）定义函数TwoNum，定义变量用来存储计算的结果，利用表达式计算欧姆值，并且将运算结果赋值给变量iTempResult。使用printf函数对所得到的结果进行输出显示。运行程序，显示效果如图9.8所示。

3．函数参数调用

函数调用还可以出现在函数参数中。此时，函数的返回值将作为参数使用。例如：

```
iResult=AddTwoNum(10,AddTwoNum(3,5));      /*函数调用出现在函数参数中*/
```

在这条语句中，AddTwoNum函数的功能还是进行两个数相加，然后将相加的结果作为函数的一个参数，继续进行计算。

【例9.7】判断体温是否正常（实例位置：资源包\TM\sl\09\07）

本实例编写getTemperature函数得到体温值，将其返回的体温数据交给judgeTemperature函数，作为参数使用。具体代码如下：

```
#include<stdio.h>                                   /*包含头文件*/
void judgeTemperature(float temperature);           /*声明函数*/
float getTemperature();                             /*声明函数*/

int main()                                          /*主函数 main*/
{
    judgeTemperature(getTemperature());             /*调用 judgeTemperature 函数，实参为 getTemperature
数的返回值*/
    return 0;                                       /*程序结束*/
}
float getTemperature()                              /*定义 getTemperature 函数*/
{
    float temperature;                                  /*定义变量*/
    printf("please input a temperature:\n");            /*输出提示信息*/
    scanf("%f",&temperature);                           /*输入体温*/
    printf("当前体温是：%.1f\n",temperature);           /*输出当前体温值*/
    return temperature;                                 /*返回体温值*/
}

void judgeTemperature(float temperature)            /*定义 judgeTemperature 函数，形参为一个温度值*
{
    if(temperature<=37.3f&& temperature>=36)        /*判断体温值是否正常*/
        printf("体温正常\n");
    else
        printf("体温不正常\n");
}
```

运行程序，显示效果如图9.9所示。

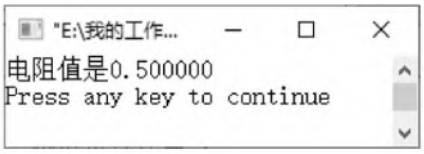


图9.8　表达式调用

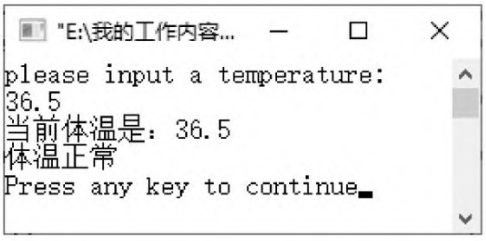


图9.9　函数参数调用

9.4.2　函数的返回

在函数体中常会看到这样一条return语句。

```
return 0;
```

这就是返回语句。return语句的作用有两个，下面进行详细介绍。

1．退出函数，返回主调程序中

C程序中，有两种方法可以终止函数执行，返回到调用函数位置。

- ☑ 函数体中的所有语句都已执行完毕，遇到结束符号“}”后自动返回。
- ☑ 遇到return语句，随即返回。

2．返回一个值，供主调函数使用

用户调用函数时，通常是希望得到一个确定的返回值。该返回值是通过return语句实现的。

返回值也需要约定类型，因此定义函数时，需要明确指定函数返回值的类型。例如：

```
int Max(int iNum1,int iNum2);                /*函数返回值类型为 int*/
double Min(double dNum1,double dNum2);       /*函数返回值类型为 double*/
char Show(char cChar);                       /*函数返回值类型为 char*/
```

如果函数返回值的类型和return语句中表达式的值不一致，则以函数返回值的类型为准。数值型数据可以自动进行类型转换，即函数定义的返回值类型决定最终返回值的类型。

注意，return语句后面的括号可以省略，即return 0和return（0）是相同的。另外，函数也可以没有返回值。例如，返回值类型为void的函数就没有返回值。

【例9.8】返回体温值（实例位置：资源包\TM\sl\09\08）

在本实例中，编写函数getTemperature，返回一个体温值。具体代码如下：

```
#include<stdio.h>                                   /*包含头文件*/
float getTemperature();                             /*声明函数*/
int main()                                          /*主函数 main*/
{
    getTemperature();                               /*调用函数*/
    return 0;                                       /*程序结束*/
}

float getTemperature()                              /*自定义温度函数*/
{
    float temperature;                              /*定义浮点型变量*/
    printf("please input a temperature:\n");        /*输出提示信息*/
    scanf("%f",&temperature);                       /*输入一个浮点型变量*/
    printf("当前体温是：%.1f\n",temperature);       /*输出当前温度*/
    return temperature;                             /*返回温度值*/
}
```

（1）首先为程序声明一个getTemperature函数，在主函数调用自定义getTemperature函数。

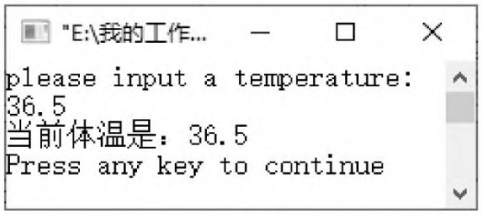


图9.10　返回值类型与return值类型

（2）getTemperature函数的定义中，定义了一个浮点型变量，用户通过提示信息输入一个温度值，最后将温度值返回。在这里可以看到，getTemperature函数中定义的温度值是float型，所以返回的也是float型变量。运行程序，显示效果如图9.10所示。

9.4.3　函数的嵌套调用



C语言虽然不允许进行函数嵌套定义，但却可以函数嵌套调用，即可在一个函数体内可以调用另外一个函数。例如，在main函数中调用Display函数，在Display函数中再调用ShowMessage函数，这种就叫作嵌套调用。调用时，一般从最外层main函数开始，一层层的调用，直到深入最内层的函数。嵌套调用返回时，恰好相反，从最内层的函数开始，一层层返回，直到返回最外层的主调函数中。

这就好比某公司要完成业绩，CEO需要将运营指标交代给总经理，总经理将工作分拆给各部门经理（副经理），部门经理们再将工作分拆给下属职员，职员按照上级指示完成具体工作。之后，职员将完成结果汇报给部门经理，部门经理汇总后将结果汇报给总经理，总经理再汇总后将结果汇报给CEO。

【例9.9】CEO下达工作内容（**实例位置：资源包\TM\sl\09\09**）

在本实例中，利用嵌套函数模拟CEO分派工作的过程。这里简化任务的完成过程，只输出一条语句表示逐级分派任务（即调用函数）的过程。

```
#include<stdio.h>

void CEO();                                    /*声明 4 个函数*/
void Manager();
void AssistantManager();
void Clerk();

int main()
{
    CEO();                                     /*调用 CEO 函数*/
    return 0;
}
void CEO()                                     /*定义 CEO 函数*/
{
    printf("CEO 交代给总经理\n");
    Manager();                                 /*调用 Manager 函数*/
}
void Manager()                                 /*定义 Manager 函数*/
{
    printf("总经理交代给部门经理\n");
    AssistantManager();                        /*调用 AssistantManager 函数*/
}
void AssistantManager()                        /*定义 AssistantManager 函数*/
{
    printf("部门经理交代给各个职员\n");
    Clerk();                                   /*调用 Clerk 函数*/
}
void Clerk()                                   /*定义 Clerk 函数*/
{
    printf("职员开始执行\n");
}
```

（1）首先在程序中声明将要使用的4个函数，其中的CEO代表公司总裁，Manager代表总经理，AssistantManager代表副经理，Clerk代表职员。

（2）main函数下是4个函数的定义。先来看一下CEO函数，通过输出一条信息来表示这个函数的功能和作用，然后嵌套调用Manager函数。Manager和CEO函数运行的步骤是相似的，最后在其函数体内调用AssistantManager函数。在AssistantManager函数中调用Clerk函数。

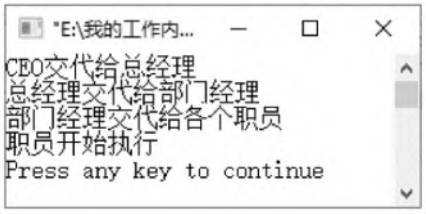


图9.11　函数的嵌套调用

（3）在主函数main中，调用了CEO函数，于是程序的整个流程按照步骤（2）进行，直到“return 0”语句返回，程序结束。

运行程序，显示效果如图9.11所示。

9.4.4 函数的递归调用



所谓递归调用，就是函数自己调用自己。从定义中可以看出，函数递归调用是函数嵌套调用的一种特殊形式。

函数可以直接调用自己，也可以间接调用自己，如图9.12所示。所谓间接调用，就是在递归函数的下层函数中调用自己。

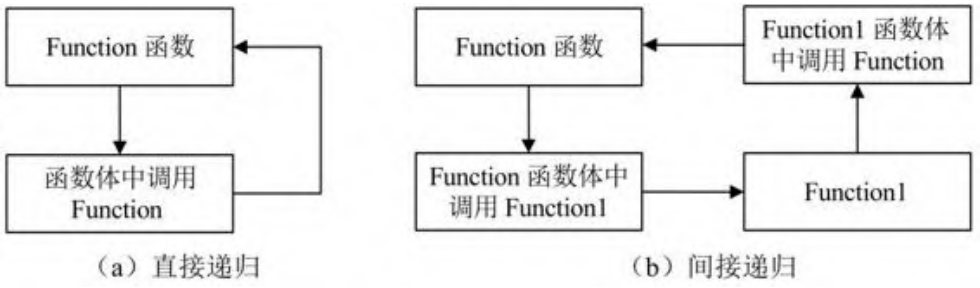


图9.12 递归调用过程

递归之所以能实现，是因为函数的每个执行过程在栈中都有自己的形参和局部变量副本，这些副本和该函数的其他执行过程不发生关系。这种机制是当代大多数程序设计语言实现子程序结构的基础。

【例9.10】倒序输出名单（实例位置：资源包\TM\sl\09\10）

在本实例中，定义一个字符串数组，为其赋值一系列名称，通过递归函数调用，逆序显示名单。注意，本例之所以能逆序输出名单，是因为嵌套函数返回时是从最内层到最外层逐层返回。

```
#include<stdio.h>
void DisplayNames(char** cNameArray);              /*声明递归函数 DisplayNames*/

char* cNames[]=                                    /*定义字符串数组*/
{
      "Aaron",
      "Jim",
      "Charles",
      "Sam",
      "Ken",
      "end"                                        /*设定一个结束标志*/
};

int main()
{
      DisplayNames(cNames);                        /*调用 DisplayNames 函数*/
      return 0;
}

void DisplayNames(char** cNameArray)               /*定义 DisplayNames 函数*/
{
      if(*cNameArray=="end")                       /*判断是否是结束标志*/
      {
            return ;                               /*如果是，函数结束返回*/
      }
      else
      {
            DisplayNames(cNameArray+1);            /*如果不是，递归调用自身*/
            printf("%s\n",*cNameArray);            /*输出字符串*/
      }
}
```

如图9.13所示为程序的调用流程，通过此图读者可对递归调用过程有一个更直观的认识。

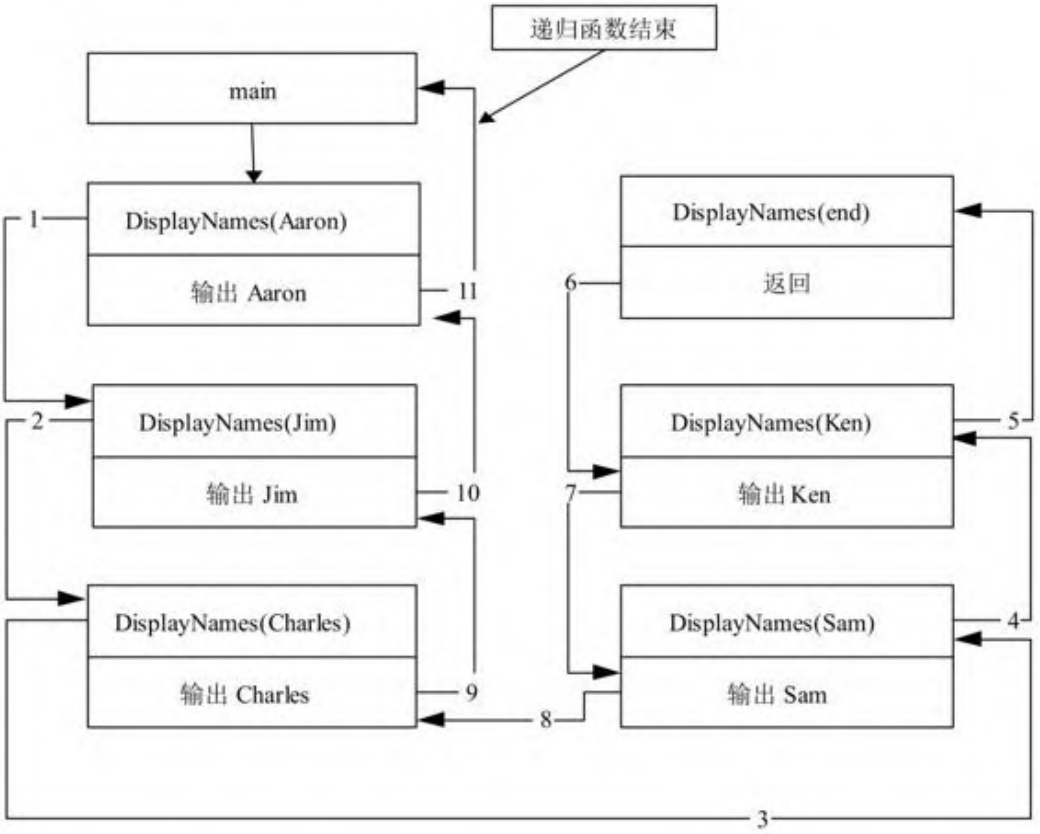


图9.13　程序调用流程图

（1）首先声明递归函数DisplayNames，其参数声明为指针的指针（第10章中将介绍，这里不用深究）。

（2）定义一个全局字符串数组，并且为其进行赋值。其中，字符串“end”作为数组的结尾标志。

（3）在主函数main中调用递归函数DisplayNames。

（4）在DisplayNames的函数体中，通过if语句判断当前要输出的字符串是否是结束字符“end”，如果是，则使用return语句返回。否则，执行下面的else语句，调用递归函数，在函数参数处可以看到传递的字符串数组元素发生改变，传递下一个数组元素。调

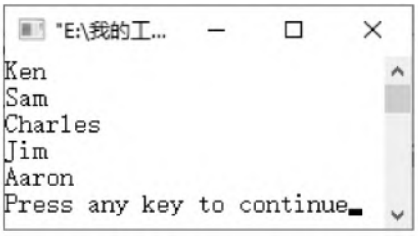


图9.14　函数的递归调用

用递归函数后，仍然要先判断传递进来的字符串是否是数组的结束标志。最后输出字符串数组的元素。

运行程序，显示效果如图9.14所示。

编程训练（答案位置：资源包\TM\sl\09\编程训练\）

训练7：判断某个数是否为回文数　编写函数，判断字符串是否为回文数（即顺读和倒读是一样的）。若是，函数返回1，否则返回0。运行结果如下：

```
请输入一个数：
12321
是回文数
```

训练8：某宝集福活动　每年春节，支付宝都会推出集福活动。自定义一个函数，将2020年支付宝推出的福打印出来，然后在主函数中调用编写的函数，运行结果如下：

```
五福为：
爱国福
富强福
和谐福
友善福
敬业福
附加福为：
全家福
沾气福
```

训练9：魔术师及代表作　自定义一个包含魔术师及其代表作的函数，在主函数中调用这个函数，输出每个魔术师的名字及作品，运行结果如下：

```
1、Jason Latimer：“划时代的近景魔术师”，代表魔术《杯与球》《激光魔术》
2、大卫·科波菲尔：代表作《人体切割》《消失的自由女神》。
3、大卫·布莱恩：擅长跳脱术，曾在一座桥下不吃不喝坚持了44天 。
```

```
4、Daryl：近景魔术大师。出过很多教学光碟 。
5、Michael.Ammar：也是近景魔术大师，出过很多教学 。
```

训练10：判断血压是否正常　正常的血压收缩压是90~140mmHg，舒张压是60~90mmHg。如果收缩压低于90mmHg或舒张压低于60mmHg，属于低血压。如果收缩压高于140mmHg或舒张压高于90mmHg，属于高血压。编写程序，判断舒张压是否正常，运行结果如下：

```
请输入舒张压数值：
88
当前舒张压是：88
血压正常
```

9.5　内部函数和外部函数

函数是C程序中的最小单位，可以把一个或多个函数保存为一个文件，这个文件就称为源文件。当一个源程序由多个源文件组成时，多个文件之间的函数可以相互调用。当然我们也可以通过将其设置为内部函数，禁止其被其他文件调用。因此，根据能不能被其他文件调用，C语言又把函数分为两类：一类是内部函数，另一类是外部函数。

9.5.1　内部函数



定义一个函数，如果该函数只能被所在的源文件使用，那么就称这样的函数为内部函数。内部函数又称为静态函数。使用内部函数，可以使函数只局限在函数所在的源文件中，如果在不同的源文件中有同名的内部函数，则这些同名函数间是互不干扰的。

定义内部函数时，要在函数返回值和函数名前面加上关键字static进行修饰。一般形式如下：

```
static 返回值类型 函数名(参数列表)
```

例如，定义一个功能为加法运算且返回值是int型的内部函数，代码如下：

```
static int Add(int iNum1,int iNum2)
```

在函数的返回值类型int前加上关键字static，可将原来的函数指定成内部函数。

技巧

使用内部函数的好处是，不同开发者编写函数时，不必再担心函数是否会与其他源文件中的函数同名。因为内部函数只在所在源文件中有效，不同源文件中即使有相同的函数名，也没有关系。

【例9.11】显示“Hello MingRi!”（实例位置：资源包\TM\sl\09\11）

在本实例中使用内部函数，通过一个函数对字符串进行赋值，再通过一个函数对字符串进行输出显示。

```
#include<stdio.h>

static char* GetString(char* pString)                /*定义字符串赋值函数*/
{
    return pString;                                  /*返回字符*/
}
static void ShowString(char* pString)                /*定义字符串输出函数*/
{
    printf("%s\n",pString);                          /*显示字符串*/
}

int main()
{
    char* pMyString;                                 /*定义字符串变量*/
    pMyString=GetString("Hello   MingRi!");          /*调用函数，为字符串赋值*/
    ShowString(pMyString);                           /*调用函数，输出显示字符串*/
    return 0;
}
```

在程序中，使用static关键字对函数进行修饰，使其只能在其源文件中进行调用。

运行程序，显示效果如图9.15所示。

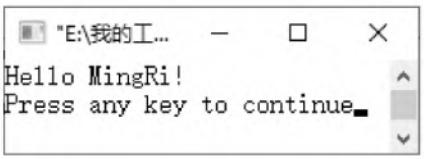


图9.15　内部函数的使用

9.5.2　外部函数



外部函数是可以被其他源文件调用的函数。定义外部函数时，使用关键字extern进行修饰。同样，使用外部函数时，要先用extern声明所用的函数是外部函数。

例如，外部函数的函数头可以写成下面的形式：

```
extern int Add(int iNum1,int iNum2);
```

这样，Add函数就可以被其他源文件调用，进行加法运算。

注意

C语言中，定义函数时如果未指明是内部函数还是外部函数，系统将默认此函数为外部函数。也就是说，定义外部函数时可以省略关键字extern。本书中多数实例使用的函数都为外部函数。

【例9.12】求三角形的内角度数（实例位置：资源包\TM\sl\09\12）

一个三角形的3个内角度数之比为2:3:3，求各内角的度数。使用外部函数编写程序求解。

```
/*////////////////////////////////////////////////////////////////////////////////*/
/*                              cal1.c 文件                                       */
/*////////////////////////////////////////////////////////////////////////////////*/
#include<stdio.h>
extern void GetAngle(int a,int b,int c)                    /*定义 GetAngle 函数*/
{
    float a1=0,b1=0,c1=0;                                  /*定义 3 个变量，用来表示三角形的 3 个内角*/
    a1=(float)180*a/(a+b+c);                               /*求每个内角的度数*/
    b1=(float)180*b/(a+b+c);
    c1=(float)180*c/(a+b+c);
    printf("内角的度数分别是：%.1f,%.1f,%.1f\n",a1,b1,c1);  /*输出内角度数*/
}
```

```
/*////////////////////////////////////////////////////////////////////////////////*/
/*                              Cal.c                                             */
/*////////////////////////////////////////////////////////////////////////////////*/
#include<stdio.h>
extern void GetAngle(int a,int b,int c);                   /*声明外部函数*/
int main()
{
    GetString(2,3,3);                                      /*调用 cal1 文件中的 GetString 函数*/
    return 0;
}
```

主函数main在源文件Cal.c中。首先声明一个函数，其中使用extern关键字说明函数为外部函数。然后在main函数体中调用这个函数。在cal1.c源文件中对

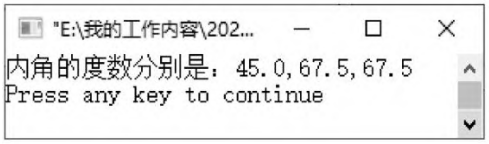


图9.16　外部函数的使用

GetString函数进行定义，通过对传入的参数执行返回操作，完成对变量的赋值功能。

运行程序，显示效果如图9.16所示。

编程训练（答案位置：资源包\TM\sl\09\编程训练\）

训练11：嵌套计算　利用嵌套函数计算s=12!+22!+32!+42!+52!。运行结果如下：

```
计算的结果是：362905
```

训练12：输出名言　编写外部函数并调用，输出名言“我们必须在别人改变之前改变自己”。

9.6　局部变量和全局变量

在讲解局部变量和全局变量的知识之前，先来了解一下作用域。作用域决定了程序中变量的作用范围：局部变量的作用域是局部的，如函数体内；全局变量的作用域是整个程序。下面具体看一下有关局部变量和全局变量的内容。

9.6.1　局部变量



在函数内部定义的变量是局部变量。我们前面学习的实例中绝大多数变量都只是局部变量，这些变量声明在函数内部，无法被其他函数所使用。函数的形式参数也属于局部变量，作用范围仅限于函数内部的所有语句块。

图9.17表示的是不同情况下局部变量的作用域范围。

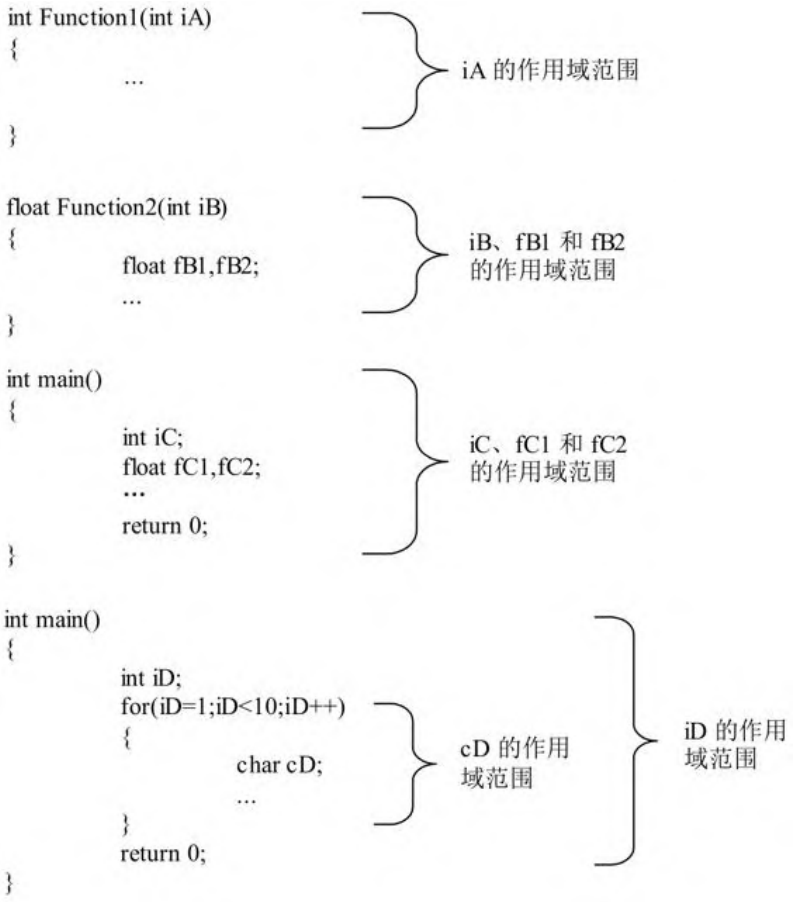


图9.17　局部变量的作用域范围

说明

在语句块内声明的变量仅在该语句块内部起作用，当然也包括嵌套在其中的子语句块。

【例9.13】模拟美团外卖商家的套餐（实例位置：资源包\TM\sl\09\13）

某米线店推出如下套餐：考神套餐13元，单身套餐9.9元，情侣套餐20元。

本实例在不同的位置定义一些变量，并为其赋值来表示变量的所在位置，最后输出显示其变量值，通过输出的信息来观察局部变量的作用范围。

```
#include<stdio.h>                                   /*包含头文件*/

int main()                                          /*主函数 main*/
{
    int iNumber1;                                   /*定义变量*/
    printf("米线店套餐如下：1：考神套餐 2：单身套餐 3：情侣套餐\n");
    if(merchant1=1)
    {
        int iNumber2;                               /*iNumber2 的作用域在 if 语句块中*/
        printf("考神套餐 13 元\n");
        if(iNumber2=2)
        {
            int iNumber3;                           /*iNumber3 的作用域在 if 语句块中*/
            printf("单身套餐 9.9 元\n");
            if(iNumber3=3)
            {
                printf("情侣套餐 20 元\n");
            }
        }
    }
    return 0;
}
```

（1）程序中有3个作用域范围，主函数main是其中最大的作用域范围。因为定义变量iNumber1在main函数中，所以iNumber1的作用范围是整个main函数体。

（2）iNumber2定义在第一个if语句块中，因此它的使用范围就是在第一个if语句块内。变量iNumber3在最内部的嵌套层，因此使用范围只在最里面的if语句块中。

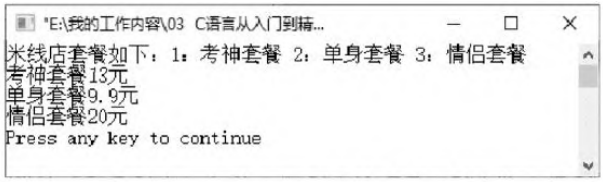


图9.18 局部变量的作用域

局部变量的作用范围由包含该变量的一对大括号所限定，据此可以更好地观察出局部变量的作用域。运行程序，显示效果如图9.18所示。

C语言中，位于不同作用域的变量可以使用相同的标识符，也就是变量名可以相同。如果内层作用域中定义的变量和已经声明的某个外层作用域中的变量有相同名称，在内层中使用这个变量名，此时的变量名表示的是外层变量还是内层变量呢？答案是：内层作用域中的变量将屏蔽外层作用域中的同名变量，直到结束内层作用域为止。这就是局部变量的屏蔽作用。

【例9.14】局部变量的屏蔽作用（实例位置：资源包\TM\sl\09\14）

在本实例中，在不同的语句块内定义了3个相同名称的变量，通过输出的变量值来感受局部变量的屏蔽作用效果。

```
#include<stdio.h>

int main()                                  /*主函数 main*/
{
    int iNumber1=1;                         /*定义第 1 个 iNumber1 变量*/
    printf("%d\n",iNumber1);                /*输出第 1 个 iNumber1 的变量值*/
    if(iNumber1>0)
    {
        int iNumber1=2;                     /*定义第 2 个 iNumber1 变量*/
        printf("%d\n",iNumber1);            /*输出第 2 个 iNumber1 的变量值*/
        if(iNumber1>0)
        {
            int iNumber1=3;                 /*定义第 3 个 iNumber1 变量*/
            printf("%d\n",iNumber1);        /*输出第 3 个 iNumber1 的变量值*/
        }
        printf("%d\n",iNumber1);            /*再次输出第 2 个 iNumber1 的变量值*/
    }
    printf("%d\n",iNumber1);                /*再次输出第 1 个 iNumber1 的变量值*/
    return 0;
}
```

（1）在主函数main中，定义了第一个整型变量iNumber1，将其赋值为1，然后使用printf函数进行输出。可以看到，此时iNumber1的值为1。

（2）使用if语句进行判断，这里使用if语句的目的在于划分出一段语句块。因为位于不同作用域的变量可以使用相同的标识符，所以在if语句块中也定义一个iNumber1变量，并将其赋值为2。再次使用printf函数输出变量iNumber1的操作，观察一下程序的运行结果，发现输出的值为

2。说明值为2的iNumber1在此作用域中将值为1的iNumber1屏蔽掉了。

（3）在if语句中再次进行嵌套，其嵌套语句中定义第3个相同的标识符iNumber1变量。为了进行区分，将其赋值为3。调用printf函数输出变量iNumber1，从程序运行的结果可以看出显示结果为3。说明值为3的iNumber1将值为2与1的两个iNumber1都进行了屏蔽。

（4）在最深层嵌套的if语句结束之后，使用printf函数进行输出，发现此时显示的值为2。说明此时已经不在值为3的iNumber1作用域范围内，而在值为2的iNumber1作用域范围。

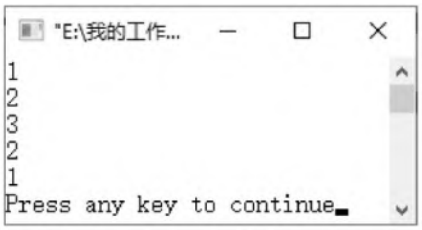


图9.19　局部变量的屏蔽作用

（5）当if语句结束之后，再次输出iNumber1值，此时显示结果为1，说明已离开了值为2的iNumber1的作用域范围，不再对值为1的iNumber1产生屏蔽作用。

运行程序，显示效果如图9.19所示。

9.6.2　全局变量



函数中定义的变量都是局部变量。如果一个变量在所有函数外部声明，这个变量就是全局变量。顾名思义，全局变量可以在程序的任何位置进行访问。

注意

全局变量不属于某个函数，而属于整个源文件。如果外部文件要访问它，则要用extern关键字进行引用修饰。

全局变量可增加函数间的数据联系。同一文件中的所有函数都能引用全局变量的值，因此如果在一个函数中改变了全局变量的值，就能影响到其他函数，相当于各个函数间有了一个直接传递通道。

例如，有一家全国连锁商店，其商品价格是全国统一的。全国各地有很多这样的连锁商店，当进行价格调整时，应该确保每一家连锁商店的价格是相同的。全局变量就像其中所要设定的价格，而函数就像每一家连锁店，当全局变量进行修改时，函数中使用的该变量都将同步被更改。

【例9.15】模拟连锁超市价格调整（实例位置：资源包\TM\sl\09\15）

在本程序中，使用全局变量模拟连锁店的全国价格调整，使用函数表示连锁店，并在函数中输出一条消息，表示连锁店中的价格。

```
#include<stdio.h>
int iGlobalPrice=100;                                   /*设定商品的初始价格*/

void Store1Price();                                     /*声明 4 个函数*/
void Store2Price();
void Store3Price();
void ChangePrice();

int main()
{
    printf("原价格 : %d\n",iGlobalPrice);               /*先显示价格改变之前所有连锁店的价格*/
    Store1Price();                                      /*显示 1 号连锁店的价格*/
    Store2Price();                                      /*显示 2 号连锁店的价格*/
    Store3Price();                                      /*显示 3 号连锁店的价格*/
    ChangePrice();                                      /*调用函数，改变连锁店的价格*/
    printf("修改的价格是: %d\n",iGlobalPrice);          /*给出提示，显示修改后的价格*/
    Store1Price();                                      /*显示 1 号连锁店的当前价格*/
    Store2Price();                                      /*显示 2 号连锁店的当前价格*/
    Store3Price();                                      /*显示 3 号连锁店的当前价格*/
    return 0;
}

void Store1Price()                                      /*定义 1 号连锁店的价格函数*/
{
    printf("1 号连锁店的价格 : %d\n",iGlobalPrice);
}
void Store2Price()                                      /*定义 2 号连锁店的价格函数*/
{
    printf("2 号连锁店的价格 : %d\n",iGlobalPrice);
}
void Store3Price()                                      /*定义 3 号连锁店的价格函数*/
{
    printf("3 号连锁店的价格 : %d\n",iGlobalPrice);
}
void ChangePrice()                                      /*定义更改连锁店价格的函数*/
{
    printf("您想要改价格吗？如果是，改的价格是: ");
    scanf("%d",&iGlobalPrice);
}
```

（1）在程序中，定义了一个全局变量iGlobalPrice来表示全国连锁店的初始商品价格，为了可以形成对比，初始化值为100。还定义了4个函数，其中3个代表连锁店的价格，例如Store1Price代表1号连锁店；ChangPrice函数用来改变全局变量的值，代表对所有连锁店进行调价。

（2）在主函数main中，首先将3家连锁店的初始价格进行显示，之后通过一条信息提示更改iGlobalPrice变量。全局变量被修改后，再次输出将所有连锁店的当前价格。

（3）通过程序的运行结果可以看出，全局变量成为函数间数据联系的渠道，修改一个全局变量后，所有函数中的该变量都会发生改变。

运行程序，显示效果如图9.20所示。



图9.20　使用全局变量模拟价格调整

编程训练（答案位置：资源包\TM\sl\09\编程训练\）

训练13：选择桌面主题　小红经常变换电脑桌面，由于收藏的桌面风格太多，他编写了一个程序，定义style局部变量，利用这个变量将所有风格输出。假设本例中需要输出风景、蓝天白云、奇幻梦境3种风格的壁纸。

训练14：钱包里还剩多少钱　将钱包中的总金额设为全局变量，定义一个pay（int number）付款方法，每次付款之后，都会减少钱包中的总金额。付款3次之后，钱包中还剩多少钱?

9.7　函数应用



编译系统通常会提供一些标准库函数，以供用户快速调用。不同的编译系统，提供的库函数会略有差异。下面介绍ANSI C提供的部分常用的库函数。

9.7.1　数学函数

在程序中经常会进行一些数学相关的运算，这里首先介绍有关数学的常用函数。注意，使用数学函数时，要包括math.h头函数，即要在程序

开始处添加#include<math.h>。

1．abs函数

abs函数的功能是求解整数的绝对值。函数形式如下：

```
int abs(int i);
```

例如，求一个负数绝对值的代码如下：

```
int iAbsoluteNumber;                    /*定义整型变量*/
int iNumber = -12;                      /*定义整型变量，为其赋值-12*/
iAbsoluteNumber=abs(iNumber);           /*调用 abs 函数*/
```

2．labs函数

labs函数的功能是求解长整型数的绝对值。函数形式如下：

```
long labs(long n);
```

例如，求一个长整型绝对值的代码如下：

```
long lResult;                           /*定义长整型变量*/
long lNumber = -1234567890L;            /*定义长整型变量，为其赋值-1234567890*/
lResult= labs(lNumber);                 /*调用 labs 函数*/
```

3．fabs函数

fabs函数的功能是求解实型数的绝对值。函数形式如下：

```
double fabs(double x);
```

例如，求一个实型数绝对值的代码如下：

```
double fFloatResult;                    /*定义实型变量*/
double fNumber = -1234.0;               /*定义实型变量，为其赋值-1234.0*/
fFloatResult= fabs(fNumber);            /*调用 fabs 函数*/
```

【例9.16】判断两人相差几岁（实例位置：资源包\TM\sl\09\16）

在本实例中，使用abs函数计算两人相差的岁数。具体代码如下：

```
#include<stdio.h>                                   /*包含头文件 stdio.h 和 math.h */
#include<math.h>
int main()
{
    int iAbsoluteNumber;                            /*定义整型变量*/
    int iNumber;
    int differ,ab;
    printf("输入年龄：\n");
    scanf("%d,%d",&iAbsoluteNumber,&iNumber);       /*输入两个年龄*/
    differ=iAbsoluteNumber-iNumber;                 /*相差的年龄*/
    ab=abs(differ);                                 /*求相差的绝对值*/
    printf("两个人相差%d 岁 \n",ab);                 /*显示输出相差的年龄*/
    return 0;                                       /*程序结束*/
}
```

运行程序，显示效果如图9.21所示。

4．sin函数

sin函数的功能是求解正弦。函数形式如下：

```
double sin(double x);
```

例如，求一个角正弦值的代码如下：

```
double fResultSin;                                  /*定义实型变量*/
double fXsin = 0.5;                                 /*定义实型变量，为其赋值为 0.5*/
fResultSin = sin(fXsin);                            /*使用正弦函数*/
```

5．cos函数

cos函数的功能是求解余弦。函数形式如下：

```
double cos(double x);
```

例如，求一个角余弦值的代码如下：

```
double fResultCos;                          /*定义实型变量*/
double fXcos = 0.5;                         /*定义实型变量，为其赋值为 0.5*/
fResultCos = cos(fXcos);                    /*调用余弦函数*/
```

6．tan函数

tan函数的功能是求解正切。函数形式如下：

```
double tan(double x);
```

例如，求一个角正切值的代码如下：

```
double fResultTan;                          /*定义实型变量*/
double fXtan = 0.5;                         /*定义实型变量，为其赋值为 0.5*/
fResultTan = tan(fXtan);                    /*调用正切函数*/
```

【例9.17】求梯形顶角及正切值（实例位置：资源包\TM\sl\09\17）

已知一个梯形是由一个正方形和两个等腰直角三角形组成的，求此梯形顶角的度数和它的正切值。

```
#include<stdio.h>
#include<math.h>                            /*包含头文件 math.h*/
int main()
{
    double fResultTan;                      /*用来保存正切值*/
    int Result;
    Result=90+45;                           /*求顶角*/
    fResultTan =tan(Result);                /*调用正切函数*/
    printf("另一个内角是:%d\n",Result);      /*输出顶角结果*/
    printf("正切值是：%f\n",fResultTan);     /*输出顶角的正切值*/
    return 0;
}
```

运行程序，显示效果如图9.22所示。

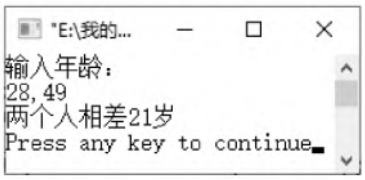


图9.21　数学库函数使用

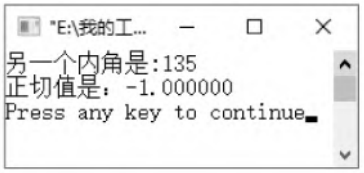


图9.22　使用三角函数

9.7.2　字符函数

下面要介绍的是3个有关字符的函数，使用时要先包含头文件ctype.h。

1．isalpha函数

isalpha函数的功能是检测某个字符是否是字母，如果是字母表中的字母（大写或小写），则返回非0值，否则返回0。函数形式如下：

```
int isalpha(int ch);
```

例如，判断输入的字符是否为字母的代码如下：

```
char c;                                  /*定义字符变量*/
scanf("%c", &c);                         /*输入字符*/
isalpha(c);                              /*调用 isalpha 函数判断输入的字符*/
```

2．isdigit函数

isdigit函数的功能是检测某个字符是否是数字，如果是数字，则函数返回非0值，否则返回0。函数形式如下：

```
int isdigit(int ch);
```

例如，判断输入的字符是否为数字的代码如下：

```
char c;                                  /*定义字符变量*/
scanf("%c", &c);                         /*输入字符*/
isdigit(c);                              /*调用 isdigit 函数判断输入的字符*/
```

3．isalnum函数

isalnum函数的功能是检测某个字符是否是字母或数字，如果是字母表中的某个字母或某个数字，则函数返回非0值，否则返回0。函数形式如下：

```
int isalnum(int ch);
```

例如，判断输入的字符是否为数字或字母的代码如下：

```
char c;                                  /*定义字符变量*/
scanf("%c", &c);                         /*输入字符*/
isalnum(c);                              /*调用 isalnum 函数判断输入的字符*/
```

【例9.18】判断输入字符的类型（实例位置：资源包\TM\sl\09\18）

在本程序中，向控制台输入字符，利用if判断语句和字符函数判断输入的是哪一种类型的字符，然后根据不同的字符类型输出相应提示信息。

```
#include<stdio.h>
#include<ctype.h>                                   /*包含头文件 ctype.h*/

void SwitchShow(char c);

int main()
{
    char cCharPut;                                  /*定义字符变量，用来接收输入的字符*/
    char cCharTemp;                                 /*定义字符变量，用来接收回车符*/

    printf("请第一次输入一个字符:");                  /*消息提示，第一次输入字符*/
    scanf( "%c", &cCharPut);                        /*输入字符*/
    SwitchShow(cCharPut);                           /*调用函数进行判断*/
    cCharTemp=getchar();                            /*接收回车符*/

    printf("请第二次输入一个字符:");                  /*消息提示，第二次输入字符*/
    scanf( "%c", &cCharPut);                        /*输入字符*/
    SwitchShow(cCharPut);                           /*调用函数判断输入的字符*/
    cCharTemp=getchar();                            /*接收回车符*/

    printf("请第三次输入一个字符:");                  /*消息提示，第 3 次输入字符*/
    scanf( "%c", &cCharPut);                        /*输入字符*/
    SwitchShow(cCharPut);                           /*调用函数判断输入的字符*/

    return 0;                                       /*程序结束*/
}

void SwitchShow(char cChar)
{
    if(isalpha(cChar))                              /*判断是否为字母*/
    {
        printf("您输入的是字母 %c\n",cChar);
    }
    if(isdigit(cChar))                              /*判断是否为数字*/
    {
        printf("您输入的是数字 %c\n", cChar);
    }
    if(isalnum(cChar))                              /*判断是否为字母或者数字*/
    {
        printf("您输入的是字母或者数字 %c\n", cChar);
    }
    else                                            /*当字符既不是字母也不是数字时*/
    {
        printf("您输入的是既不是字母也不是数字:%c\n", cChar);
    }
}
```

（1）要使用字符函数，先要引入头文件ctype.h。

（2）在程序中定义了两个字符变量，cCharPut用来在程序中接收用户输入的字符，而cCharTemp的作用是接收用户按Enter键所产生的

回车符。

（3）定义SwitchShow函数，实现在程序中判断字符的功能。SwitchShow函数体中，在3个if语句的判断条件中调用不同的字符函数，根据返回值判断传递的字符参数cChar是哪种情况，并给出不同的提示信息。

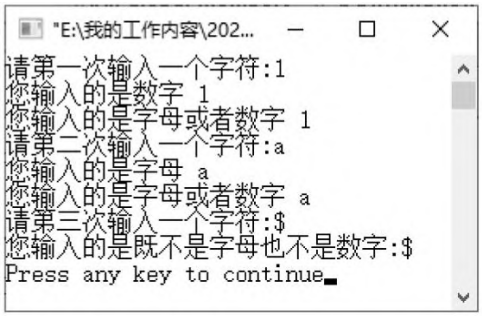


图9.23　使用字符函数判断输入字符

（4）在main函数中调用getchar函数，其作用是获取一个字符。因为用户每次输入完毕后都要按Enter键进行确定，为防止回车符被当做输入的字符，这里调用getchar函数预先提取走回车符。

运行程序，显示效果如图9.23所示。

说明

读者可以尝试将main函数中getchar函数所在行的代码注销掉，运行程序并观察结果，会发现第二个输入操作会被程序跳过。想一想，为什么。

9.7.3　字符串处理函数



编写程序时，经常要对字符和字符串进行操作，如转换字符大小写、求字符串长度等，这些都可以使用字符串函数来解决。C语言标准函数库

中提供了一系列字符串处理函数，合理、有效地使用这些字符串函数，可以大大提高编程效率。本节将介绍常用的字符串处理函数，使用时要先包含头文件string.h。

1．strcpy函数

在字符串处理函数中，strcpy函数用于复制特定长度的字符串到另一个字符串中。其语法格式如下：

```
strcpy(字符数组1,字符数组2)
```

功能：把字符数组2中的字符串复制到字符数组1中，字符串结束标志“\0”也一同复制。复制过去的内容将覆盖字符数组1中的对应内容。

说明

（1）字符数组1要有足够的长度，否则无法装下待复制的字符串。

（2）“字符数组1”必须写成数组名形式；而“字符数组2”可以是数组名，也可以是一个字符串常量，这时相当于把一个字符串赋予该字符数组。

（3）不能用赋值语句将一个字符串常量或字符数组直接赋给一个字符数组。

【例9.19】照葫芦画瓢（实例位置：资源包\TM\sl\09\19）

在main函数体中定义两个字符数组，通过键盘输入获取两个字符串并分别输出，调用strcpy函数将源字符数组中的字符串赋值给目标字符数组，最后输出目标字符数组。

```
#include<stdio.h>
#include<string.h>                                        /*包含头文件 string.h*/

int main()
{
    char str1[30],str2[30];
    printf("输入目标字符串：\n");
    gets(str1);                                           /*输入目标字符串*/
    printf("输入源字符串：\n");
    gets(str2);                                           /*输入源字符串*/
    printf("输出目标字符串：\n");
    puts(str1);                                           /*输出目标字符串*/
    printf("输出源字符串：\n");
    puts(str2);                                           /*输出源字符串*/
    strcpy(str1,str2);                                    /*调用 strcpy 函数实现字符串复制*/
    printf("调用 strcpy 函数进行字符串复制：\n");
    printf("复制字符串之后的目标字符串：\n");
    puts(str1);                                           /*输出复制后的目的字符串*/
    return 0;                                             /*程序结束*/
}
```

运行程序，字符串复制效果如图9.24所示。



图9.24　字符串复制

2．strcat函数

字符串连接就是将一个字符串连接到另一个字符串的末尾，使其组合成一个新的字符串。strcat函数就具有字符串连接的功能，其语法格式如下：

```
strcat(字符数组1,字符数组2)
```

功能：把字符数组2中的字符串连接到字符数组1的字符串后面，并删去字符数组1中原有的串结束标志“\0”。

说明

字符数组1应有足够的长度，以保证能装下连接后的字符串。

【例9.20】显示文件完整路径（实例位置：资源包\TM\sl\09\20）

编写程序，接受用户输入的目录和文件名，然后输出文件的完整路径。

在main函数体中定义两个字符数组，分别存储输入的目录和文件名，调用strcat函数将文件名字符串连接到目录字符串的后面，最后输出目录字符数组。

```
#include<stdio.h>
#include<string.h>

int main()
{
    char str1[30],str2[30];
    printf("输入目录:\n");
    gets(str1);                          /*输入目录字符串*/
    printf("输入文件名:\n");
    gets(str2);                          /*输入文件名字符串*/

    printf("输出目录:\n");
    puts(str1);                          /*输出目录*/
    printf("输出文件名:\n");
    puts(str2);                          /*输出文件名*/
    strcat(str1,str2);                   /*调用 strcat 函数进行字符串连接*/
    printf("文件全路径:\n");
    puts(str1);                          /*输出连接后的字符串，即文件的完整路径*/

    return 0;                            /*程序结束*/
}
```

运行程序，字符串连接效果如图9.25所示。

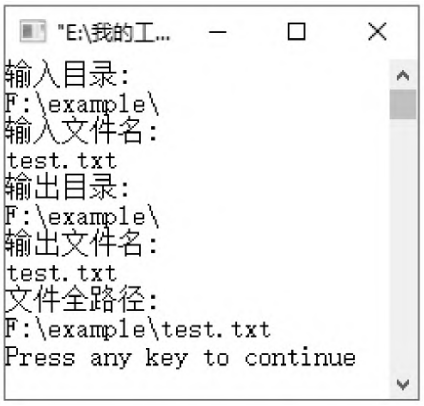


图9.25　字符串连接

说明

字符串复制实质上是用字符数组2中的字符串覆盖字符数组1中的字符串，而字符串连接则不存在覆盖的问题，只是单纯地将字符数组2中的字符串连接到字符数组1中字符串的后面。

3．strcmp函数

字符串比较就是将一个字符串与另一个字符串从首字母开始，按照ASCII码的顺序逐个进行比较。strcmp函数具有在字符串间进行比较的功能，其语法格式如下：

```
strcmp(字符数组1,字符数组2)
```

功能：按照ASCII码顺序比较两个数组中的字符串，并返回比较结果。两个字符串比较时，若出现不同的字符，则以第一个不同字符的比较结果作为整个比较的结果。返回值如下：

- ☑ 字符串1=字符串2，返回值为0。
- ☑ 字符串1>字符串2，返回值为正数。
- ☑ 字符串1<字符串2，返回值为负数。

注意

字符串比较不能使用关系运算符，也不能使用赋值运算符。因此，下面的语句是错误的：

```
if(str[2]==mingri)…
str[2]=mingri;…
```

【例9.21】模拟银行取钱（实例位置：资源包\TM\sl\09\21）

在自动取款机上取钱时，需要输入密码，密码正确方能取到钱。本实例中，正确的密码为574824，只有3次密码输入机会，如果3次都输错了，就提示“请到人工处办理解锁”。

```
#include<stdio.h>
#include<string.h>

int main()
{
    char password[20] = {"574824"};                          /*定义变量，保存正确的密码字符串*/
    char pwstr[20];                                          /*定义变量，保存用户输入的密码字符串*
    int i=1;
    while(i <= 3)                                            /*循环执行 3 次密码输入*/
   {
        printf("输入密码字符串:\n");
        gets(pwstr);                                         /*输入密码字符串*/
            if(strcmp(password,pwstr))                       /*比较两个密码，如果不相同*/
            {
                printf("第%d 次，密码字符串输入错误！\n",i);  /*提示密码输入错误*/
            }
            else                                             /*如果相同*/
            {
                printf("密码正确，请选择服务！\n");           /*输出下一步业务提醒*/
                break;                                       /*退出*/
            }
        i++;
    }
    if(i == 4)
    {
        printf("输入字符串错误 3 次！请到人工处办理解锁\n");  /*如果密码输入错误多于 3 次*/
    }
    return 0;                                                /*程序结束*/
}
```

运行程序，字符串比较效果如图9.26所示。

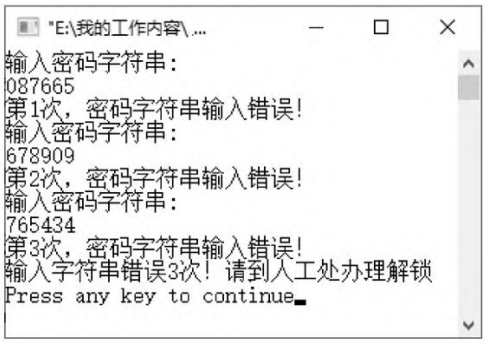


图9.26　字符串比较

4．strupr函数和strlwr函数

字符串的大小写转换需要使用strupr和strlwr函数。strupr函数的语法格式如下：

```
strupr(字符数组名)
```

功能：将数组中存放的字符串的小写字母转换成大写字母，其他字母不变。

strlwr函数的语法格式如下：

```
strlwr(字符数组名)
```

功能：将数组中存放的字符串的大写字母转换成小写字母，其他字母不变。

【例9.22】字符串大小写转换（实例位置：资源包\TM\sl\09\22）

将张三的邮箱地址ZhangSan@MRSOFT.COM全部转换为小写或大写。

在main函数体中定义两个字符数组，分别用来存储要转换的字符串和转换后的字符串，然后根据用户输入的操作指令，判断应调用strupr函

数还是strlwr函数，进行大小写转换。

```
#include<stdio.h>
#include<string.h>

int main()
{
    char text[20]="ZhangSan@MRSOFT.COM",change[20];
    printf("原字符串为:%s\n",text);                    /*输出要转换的字符串*/
    strcpy(change,text);                               /*复制要转换的字符串*/
    strlwr(change);                                    /*字符串转换为小写*/
    printf("转换成小写字母的字符串为:%s\n",change);     /*输出转换后的字符串*/
    strupr(change);                                    /*字符串转换为大写*/
    printf("转换成大写字母的字符串为:%s\n",change);     /*输出转换后的字符串*/
    return 0;                                          /*程序结束*/
}
```

运行程序，字符串大小写转换效果如图9.27所示。



图9.27　字符串大小写转换

5．strlen函数

有时需要知道字符串的长度，虽然通过循环判断字符串结束标志“\0”也能获得字符串的长度，但毕竟实现起来相对烦琐。这时可以使用strlen函数来计算字符串的长度。

strlen函数的语法格式如下：

```
strlen(字符数组名)
```

功能：计算数组中存放的字符串的实际长度（不含字符串结束标志“\0”）。

【例9.23】判断用户输入的密码是否为6位（实例位置：资源包\TM\sl\09\23）

在本实例中的main函数体中定义一个字符数组，用来存储用户输入的字符串，然后调用strlen函数计算字符串长度，并使用if…else…语句判断密码长度是否等于6。

```
#include<stdio.h>
#include<string.h>

int main()
{
        char text[50];                          /*定义字符数组，保存密码字符串*/
        printf("输入一个密码:\n");
        scanf("%s", &text);                     /*输入用户密码*/

        if(strlen(text)==6)                     /*计算密码长度并判断是否等于 6*/
            printf("输入密码是 6 位\n");
        else
            printf("输入密码不是 6 位\n");
        return 0;                               /*程序结束*/
}
```

运行程序，输出效果如图9.28所示。

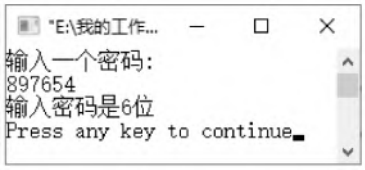


图9.28　计算密码长度

编程训练（答案位置：资源包\TM\sl\09\编程训练\）

训练15：判断地铁两站间行车时间　已知某地铁线路有5站，将从始发站发车行驶到各个站的时间集合{5，10，15，20，25}保存成一个一维数组a，编写程序，判断任意两站之间需要行车多长时间。运行结果如下：

```
请任意输入两站：
3 1
两站之间行车需要10分钟
```

训练16：寻找座位号　使用isdigit函数判断输入的座位号是否是数字，运行效果如下：

结果一：

```
请输入您的位置号：
------------------------------------------------------
|   对不起，您输入的座位号格式不正确    |
|          不能找到您的位置             |
------------------------------------------------------
```

结果二：

```
请输入您的位置号：5
→*→*→*→*→*→*→*→*→*→*→*→*→*→*
  您的位置号是5
  您输入的格式正确，请坐....
→*→*→*→*→*→*→*→*→*→*→*→*→*→*
```

训练17：显示天气预报　使用strcat函数，输出2021年1月23日的实时天气预报。输出结果如下：

```
2021年1月23日 晴，20℃~7℃，微风转西风3~4级
08:00   晴13℃   微风
12:00   晴19℃   微风
16:00   晴18℃   微风
20:00   晴15℃   微风
00:00   晴12℃   微风
```

训练18：更新公告　利用strcpy函数，将包子店的招牌“包子一元一个”换成“包子壹圆壹个”。输出结果如下：

```
原来的招牌内容是：
    包子一元一个
经过处理之后的招牌内容是:
    包子壹圆壹个
```

9.8　实践与练习

（答案位置：资源包\TM\sl\09\实践与练习\）

综合练习1：模拟12306售票系统　12306售票系统全国联网，数据同步。因此，每售出一张票，全国各地的系统显示都会减少一张票数。利用全局变量模拟12306抢票系统，输出效果如下：

```
始发地：上海　目的地：长春　时间：2021年4月10日16：20出发
3个城市剩余的票数分别为：
上海的12306系统剩余票数：99张
北京的12306系统剩余票数：99张
深圳的12306系统剩余票数：99张
我抢到一张票数之后剩余票数：98
我抢到一张票之后3个城市剩余的票数分别为：
上海的12306系统剩余票数：98张
北京的12306系统剩余票数：98张
深圳的12306系统剩余票数：98张
```

综合练习2：为和尚写诗　自定义一个poetry函数，为和尚写一首诗。诗句如下：

```
空门有路不知处
头白齿黄犹念经
何年饮着声闻酒
迄至如今醉未醒
```

综合练习3：一棵松树的梦　在源文件中定义一个全局变量pinetree，并为它赋值。再定义一个christmastree函数，在这个函数里定义名称为pinetree的局部变量，并输出，最后在主函数中调用christmastree函数，并输出全局变量pinetree的值。输出结果如下：

```
下雪了……

============开始做梦……==============

挂上彩灯、礼物……我变成一棵圣诞树@^.^@

============梦醒了……================

我身上落满雪花,我是一棵松树 -_-
```

综合练习4：确定女主角　某导演有一个剧本，需要寻找演员来饰演对应的角色。利用函数的实参和形参知识编写代码，实现为剧本选女主角的功能。输出结果如下：

```
导演选定女主角是：Lucy
→*→*→*→*→*→*→*→*→*→*→*→*→*→*
    Lucy 开始参演李美丽角色
→*→*→*→*→*→*→*→*→*→*→*→*→*→*
```

综合练习5：为C语言归类　要想在腾讯课堂上查看某个课程，一般需要知道其分类。例如，C语言的所属类别从大到小依次为“IT—互联网—编程语言—C”，利用函数嵌套找到C语言课程。效果如下：

```
（1）找到IT分类
（2）IT分类中找到互联网分类
（3）互联网分类中找到编程语言分类
（4）编程语言分类找到C语言课程
```

综合练习6：递归求年龄　甲乙丙丁戊5个人坐在一起聊天。大家猜戊的年龄，他说比丁大2岁；问丁的岁数，他说比丙大2岁；问丙的岁数，他说比乙大2岁；问乙的岁数，他说比甲大2岁；问甲的岁数，他说自己10岁。编写程序求戊的年龄。输出结果如下：

```
------------------------------
   戊的年龄是：18 岁
------------------------------
```

综合练习7：将美元兑换成人民币　编程实现美元兑换人民币功能。美元与人民币之间的汇率经常变更，这里按1美元等于6.28元人民币计算。输出结果如下：

```
您要兑换的美元金额：500

**************************
*  兑换成人民币金额是：3140.00  *
**************************
```

综合练习8：你的心跳正常吗？　利用把函数作为参数使用，编写程序判断输入的心跳数是否是正常心跳数。当心跳数在60~100次／分钟，显示心跳数正常，运行结果如图9.29所示；当心跳数大于100次／分钟或小于60次／分钟时，就显示心跳不正常，运行结果如图9.30所示。

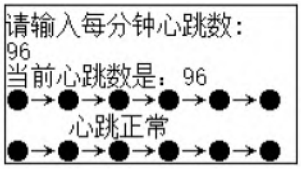


图9.29　心跳正常

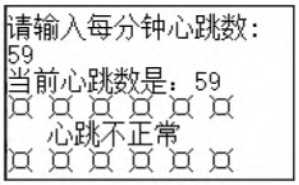


图9.30　心跳不正常

综合练习9：注册明日学院VIP账号　在明日学院注册账号时，账号长度要求为4~12位，使用strlen函数判断注册的明日学院账号是否符合要求。输出结果如下：

```
请输入您想注册的明日学院账号:
mingrisoft
注册成功
```

综合练习10：判断车牌归属地　根据车牌号可以知道一辆车的归属地。编写程序，利用strcmp函数判断车牌的归属地。运行结果如下：

```
车牌号归属地查询:
津 A·12345 这个车牌号的归属地是：天津
沪 A·23456 这个车牌号的归属地是：上海
京 A·34567 这个车牌号的归属地是：北京
```

第10章
指针

指针是C语言的一个重要组成部分，是C语言的核心、精髓所在。用好指针，可以在C语言开发中起到事半功倍的效果。一方面，可以提高程序的编译效率、执行速度，以及动态存储分配；另一方面，可使程序更加灵活，表示和操作各种数据结构更便捷，编写出高质量的应用程序。

本章的知识架构及重难点如下：



10.1　指针的相关概念

指针是C语言的独特之处，其使用起来十分灵活。使用得当，能显著提高某些程序的效率；使用不当，则很容易造成系统错误。许多程序“挂死”往往都是错误地使用指针造成的。

10.1.1　地址与指针



要想弄明白什么是指针，就必须弄清楚数据在内存中是如何存储的，又是如何被读取的。

通常来说，系统会按字节对每个内存单元进行编号，这些内存单元就好比是许多带有编号的小房间，要想使用内存，就需要知道房间编号。例如，定义一个整型变量i需要4个字节，编译器就会为其分配4个内存单元，如图10.1所示，1000~1003就是对应变量i的4个内存单元地址，其内存放的数值是0。假设又定义了一个整型变量j，则j在内存中的起始地址是1004，因为j同样占4个字节，所以1004~1007单元中保存的是j，其值是1。同理，1008~1012、1012~1016、1016~1020也同样是3个整型变量对应的内存空间，如图10.2所示。

可见，编译系统为每个变量都分配了一个能满足其类型大小的内存单元地址，访问该地址就能找到对应变量。那么，程序设计中，又该如何访问变量的地址呢?

前面我们学习过取地址符“&”，其作用是获取变量在内存中的地址。例如，&i表示变量i的内存起始地址1000。通过&i可访问变量i指向的内存单元，进而读取变量i。如图10.3所示，“指向”某个变量地址用箭头表示，就好像旅店住宿时某个房间号牌指向某个房间一样。

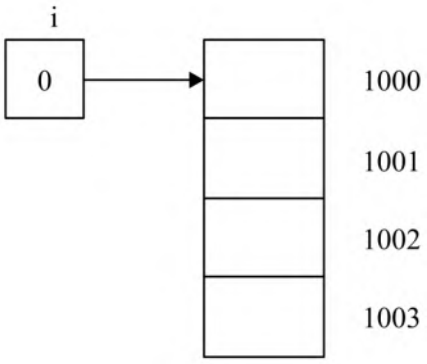


图10.1　变量在内存中的存储

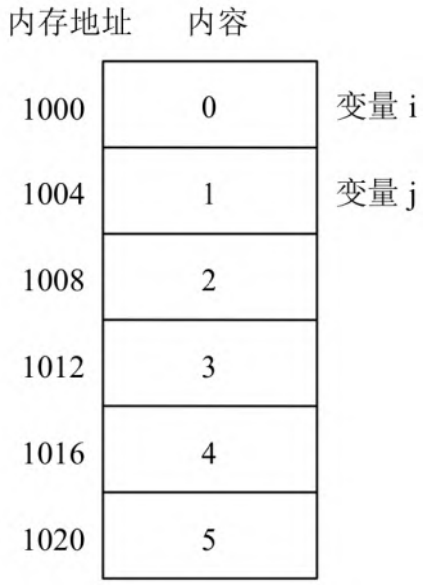


图10.2　变量存放

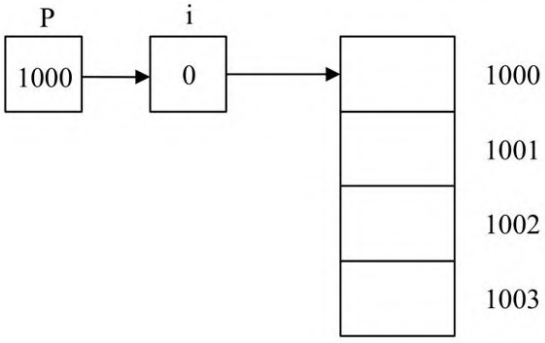


图10.3　指针

为了好记忆，我们将变量的地址形象化地称为该变量的“指针”，意思是通过它能访问以它为地址的内存单元。例如，通过地址1000可以访问变量i的地址，进而读取4个字节得到i的值。

10.1.2　变量与指针



变量的地址是变量和指针之间的连接纽带。所谓“指向”，是通过地址来体现的。因为指针变量通常指向一个变量的地址，所以将一个变量的地址赋给指针变量后，这个指针变量就“指向”了该变量。例如，将变量i的地址&i存放到指针变量p中，p就指向i，其关系如图10.4所示。

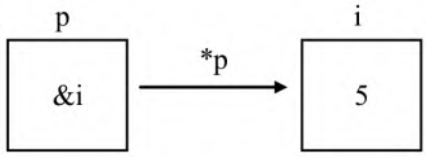


图10.4　地址与指针

程序代码中一般通过变量名对内存单元进行存取操作，但代码编译后，会将变量名转换为该变量在内存中的存放地址，后续对变量值的存取都是通过该地址进行的。例如，对图10.2中的变量i和变量j进行如下求和操作：

```
i+j;
```

计算机内部的存取流程是：根据变量名与地址的对应关系，找到变量i的地址1000，然后从1000开始读取4个字节数据放到CPU寄存器中，再找到变量j的地址1004，从1004开始读取4个字节的数据放到CPU的另一个寄存器中，通过CPU的加法中断计算出结果。

低级语言（如汇编语言）中一般直接通过地址来访问内存单元，高级语言中一般使用变量名来访问内存单元。C语言作为高级语言，仍然提供了通过地址来访问内存单元的方式。

10.1.3　指针变量



一个变量的地址称为该变量的指针。如果有一个变量专门用来存放另一个变量的地址，它就是指针变量。在C语言中有专门用来存放内存单元

地址的变量类型，即指针类型。

1．指针变量的定义

定义指针变量的一般形式如下：

```
类型说明 * 变量名
```

其中，“*”表示该变量是一个指针变量，“变量名”即为定义的指针变量名，“类型说明”表示本指针变量所指向的变量的数据类型。

2．指针变量的赋值

指针变量同普通变量一样，使用前需要先进行定义并赋予具体的值。未经赋值的指针变量不能使用。与其他变量赋值不同，给指针变量赋值只能赋予地址，而不能是其他数据，否则将引起错误。C语言一般用“&变量名”表示某个变量的地址，如&a表示变量a的地址，&b表示变量b的地址。

给一个指针变量赋值可以有以下两种方法。

（1）定义指针变量的同时进行赋值。例如：

```
int a;
int *p=&a;
```

（2）先定义指针变量，之后再赋值。例如：

```
int a;
int *p;
p=&a;
```

注意

注意这两种赋值语句的区别。如果先定义指针变量之后再赋值，赋值时指针变量前不再加“*”。

【例10.1】输出某个数的地址（实例位置：资源包\TM\sl\10\01）

从键盘中输入一个数，利用指针将这个数的地址以十六进制形式输出。实现代码如下：

```
#include<stdio.h>
int main()
{
    int a;                                  /*定义整型变量*/
    int *ipointer1;                         /*定义指针变量*/
    printf("请输入数据：\n");
    scanf("%d",&a);                         /*输入一个数*/
    ipointer1 = &a;                         /*将地址赋给指针变量*/
    printf("转化十六进制为：%x\n",*ipointer1);  /*以十六进制形式输出该数*/
    return 0;                               /*程序结束*/
}
```

程序运行结果如图10.5所示。

3．指针变量的引用

引用指针变量是对变量进行间接访问的一种形式。引用指针变量的形式为“*指针变量”，其含义是引用指针变量所指向的值。

误区警示

没有初始化的指针变量俗称“野指针”，使用时容易产生错误（导致不合法的内存空间）。良好的编程习惯是在定义指针变量时就将其初始化为NULL，由于NULL处禁止写入，所以一旦有错误，可以将错误造成的危害降到最小。

【例10.2】比较两个数的大小（实例位置：资源包\TM\sl\10\02）

使用指针比较两个数的大小，实现代码如下：

```
#include<stdio.h>
int main()
{
    int a=10,b=11;                          /*定义两个整型变量并初始化*/
    int *ipointer1,*ipointer2;              /*定义两个整型指针变量*/
    ipointer1=&a;                           /*将地址赋给指针变量*/
    ipointer2=&b;
    if(*ipointer1>*ipointer2)               /*使用指针比较两数大小*/
        printf("a 的值大\n");
    else
        printf("b 的值大\n");
    return 0;
}
```

程序运行结果如图10.6所示。

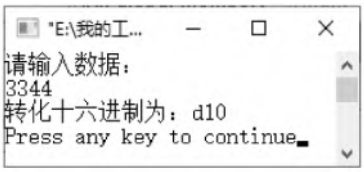


图10.5　输出某个数的地址

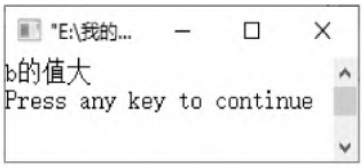


图10.6　使用指针比较两个数的大小

4. “&”和“*”运算符

运算符“&”和“*”都是单目运算符。“&”是取地址运算符，用于返回一个操作数的地址。“*”是指针运算符，用于返回指定地址内保存的变量值。

例如，“p=&i;”是将变量i的内存地址赋给指针变量p。“q=*p;”是将指针p指向的变量，即变量i的值赋给q，假如变量i的值是5，则q的值也是5。

5. “&*”和“*&”的区别

下面通过两条语句来分析“&*”和“*&”运算的区别。如果有如下语句：

```
int a;
p=&a;
```

因为“&”和“*”的优先级相同，按自右而左的方向结合，因此“&*p”先进行“*”运算，“*p”相当于变量a；再进行“&”运算，“&*p”就相当于取变量a的地址。“*&a”先进行“&”运算，“&a”就是取变量a的地址，然后执行“*”运算，“*&a”就相当于取变量a所在地址的值，实际就是变量a。

【例10.3】输出i、j、c的地址（**实例位置：资源包\TM\sl\10\03**）

本实例定义了3个指针变量，使用“&*”计算c=i+j，计算后输出变量i，j，c的地址值。

```
#include<stdio.h>
int main()
{
    long i,j,c;                                  /*定义变量*/
    long *p,*q,*n;                               /*定义指针变量*/
    printf("please input the numbers:\n");
    scanf("%ld,ld",&i,&j);                       /*输入数据*/
    c=i+j;                                       /*实现两数相加*/
    p=&i;                                        /*将地址赋给指针变量*/
    q=&j;
    n=&c;
    printf("%ld\n",&*p);                         /*利用“&*”输出变量 i 的地址*/
    printf("%ld\n",&*q);                         /*利用“&*”输出变量 j 的地址*/
    printf("%ld\n",&*n);                         /*利用“&*”输出变量 c 的地址*/
    return 0;                                    /*程序结束*/
}
```

程序运行结果如图10.7所示。

【例10.4】求水对杯子产生的压强（**实例位置：资源包\TM\sl\10\04**）

杯子的底面积是60平方厘米，杯中装上8厘米高的水，杯子和水的总质量为0.6千克（水的密度是每立方米10^3千克，重力加速度g的值为每千

克10牛顿），利用“*&”计算水对杯子产生的压强。

```
#include<stdio.h>
int main()
{
    double a=1000*10*8*0.001;                    /*定义变量并计算压强*/
    double *p;                                   /*定义指针变量*/
    p=&a;                                        /*将地址赋值给指针变量*/
    printf("根据压强公式：\n");
    printf("水对杯子产生压强:%.1lf Pa\n",*&a);          /*利用“*&”输出压强*/
    return 0;
}
```

程序运行结果如图10.8所示。

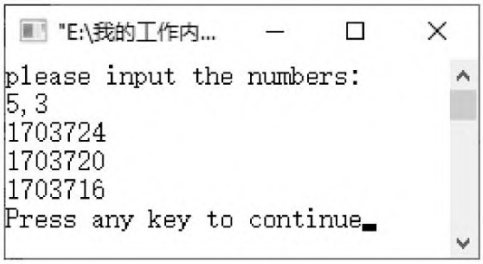


图10.7 “&*”的应用

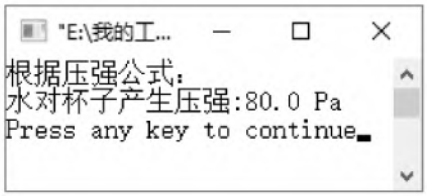


图10.8 “*&”的应用

10.1.4 指针的自增、自减运算



指针的自增自减运算不同于普通变量的自增自减运算，也就是说，并非简单地加1减1，而是按照它所指向的数据类型的直接长度进行增减。

下面通过一个例子，读者仔细揣摩一下。

【例10.5】输出整型变量地址（实例位置：资源包\TM\sl\10\05）

定义指针变量和整型变量，并将整型变量的地址赋值给指针变量，再进行指针自增，输出结果，比较自增前后的数值，看看能得出什么结论。代码如下：

```
#include<stdio.h>
main()
{
    int i;
    int *p;
    printf("please input the number:\n");
    scanf("%d",&i);                          /*输入一个整型数*/
    p=&i;                                    /*将整型变量 i 的地址赋给指针变量*/
    printf("the result1 is: %d\n",p);
    p++;                                     /*指针自增，地址加 1，即增加 4 个字节*/
    printf("the result2 is: %d\n",p);
}
```

程序运行结果如图10.9所示。若将例10.5代码改成如下：

```
#include<stdio.h>
main()
{
    short i;
    short *p;
    printf("please input the number:\n");
    scanf("%d",&i);                          /*输入一个短整型数*/
    p=&i;                                    /*将短整型变量 i 的地址赋给指针变量*/
    printf("the result1 is: %d\n",p);
    p++;                                     /*指针自增，地址加 1，即增加 2 个字节*/
    printf("the result2 is: %d\n",p);
}
```

程序运行结果如图10.10所示。可以看到，指针自增后，两段代码输出的地址并不相同。这是因为：基本整型变量i在内存中占4个字节，指针p是指向变量i的地址的，这里的p++不是简单地在地址上加1，而是指向下一个存放基本整型数的地址。图10.9所示的结果是因为变量i是基本整型，所以执行p++后，p的值增加4（即4个字节）；图10.10所示的结果

是因为i被定义成了短整型，所以执行p++后，p的值增加了2（即2个字节）。

指针自增自减运算，是按照它所指向的数据类型的直接长度进行增或减。可以将例10.5第1段代码用图10.11来形象地表示出来。

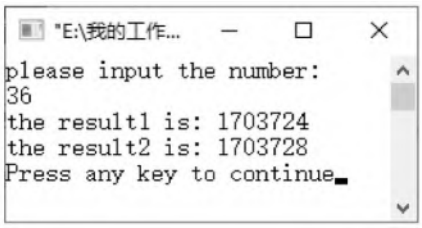


图10.9　整型变量地址输出

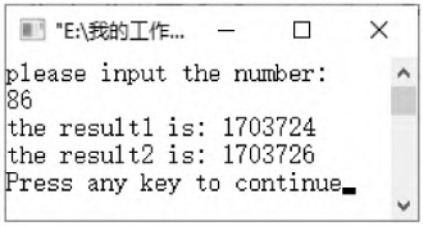


图10.10　短整型变量地址输出

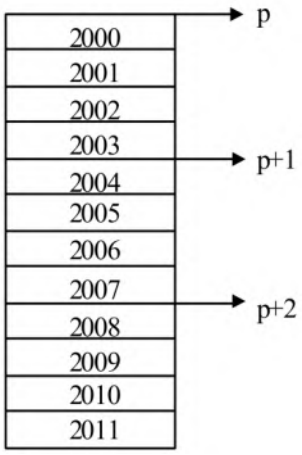


图10.11　指向整型变量的指针

编程训练（答案位置：资源包\TM\sl\10\编程训练\）

训练1：答案究竟在哪儿　小Z是班里成绩最好的同学，大家都喜欢以他的作业为范本核对答案。一天A找Z对答案，Z说答案在Y那里，Y说答

案是10。利用指针模拟这种情况，并将答案输出。

训练2：展示彩灯点亮情况　利用指针自减运算，模拟彩灯的亮灭情况。假设数字0表示灯泡没亮，数字1表示灯泡亮着，有6个灯泡排列成一行组成一个一维数组a={1，0，0，1，0，0}，查找倒数第一个亮着的灯泡位置，并显示该灯泡前一个灯泡是否亮着。运行结果如下：

```
0表示灯灭，1表示灯亮
倒数第一个亮着的灯泡是:倒数第3个
前一个灯泡没亮
```

10.2　数组与指针

使用数组时，系统需要提供一段连续的内存来存储数组中的各元素，如果把数组的地址赋给指针变量，就可以通过指针变量来引用数组。下面就介绍如何用指针来引用一维数组及二维数组元素。

10.2.1　一维数组与指针



定义一维数组时，系统会在内存中为其分配一段存储空间，数组名就是数组在内存中的首地址。若再定义一个指针变量，并将数组的首地址传给指针变量，则该指针就指向了这个一维数组。例如：

```
int *p,a[10];
p=a;
```

这里a是数组名，也就是数组的首地址，将它赋给指针变量p，也就是将数组a的首地址赋给p。也可以写成如下形式：

```
int *p,a[10];
p=&a[0];
```

上面的语句是将数组a中首个元素的地址赋给指针变量p。由于a[0]的地址就是数组的首地址，因此上述两条赋值语句的操作效果完全相同。

【例10.6】输出100以内开始5个数和后5个数（**实例位置：资源包\TM\sl\10\06**）

定义两个指针变量，再定义两个一维数组，将两个数组元素分别赋给两个指针。利用for循环输入两个数组的值（100以内开始5个数和后5个数），再用for循环和指针输出数组值。代码如下：

```
#include<stdio.h>
void main()
{
    int *p,*q,a[5],b[5],i;                    /*定义变量*/
    p=&a[0];                                  /*将数组首地址赋给指针*/
    q=b;
    printf("please input array a:\n");        /*提示输入数组 a*/
    for(i=0;i<5;i++)                          /*利用&a[i]形式输入数组 a*/
        scanf("%d",&a[i]);
    printf("please input array b:\n");        /*提示输入数组 b*/
    for(i=0;i<5;i++)                          /*利用&b[i]形式输入数组 b*/
        scanf("%d",&b[i]);
    printf("array a is:\n");                  /*提示输出数组 a 元素*/
    for(i=0;i<5;i++)
        printf("%5d",*(p+i));                 /*利用*(p+i)形式输出数组 a */
    printf("\n");
    printf("array b is:\n");                  /*提示输出数组 b*/
    for(i=0;i<5;i++)
        printf("%5d",*(q+i));                 /*利用*(q+i)形式输出数组 b */
    printf("\n");
}
```

程序运行结果如图10.12所示。

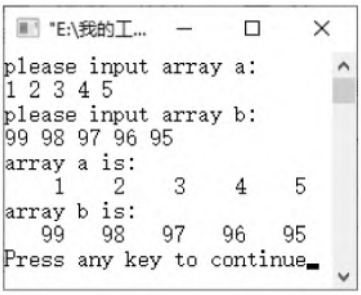


图10.12　输出数组中的元素

如下两条语句，都表示将数组首地址赋给指针变量。

```
p=&a[0];
q=b;
```

☑ p+n与a+n都用于表示数组元素a[n]的地址，即&a[n]。对整个a数组来说，共有5个元素，n的取值为0~4，则数组元素的地址就可以表示为p+0~p+4或a+0~a+4。

☑ *（p+n）和*（a+n）都用于表示数组元素a[n]。语句“printf（"%5d"，*（p+i））;”和“printf（"%5d"，*（q+i））;”表示输出数组a、b中对应的元素。

例10.6中使用指针指向一维数组及通过指针引用数组元素的过程可以通过图10.13和图10.14来表示。

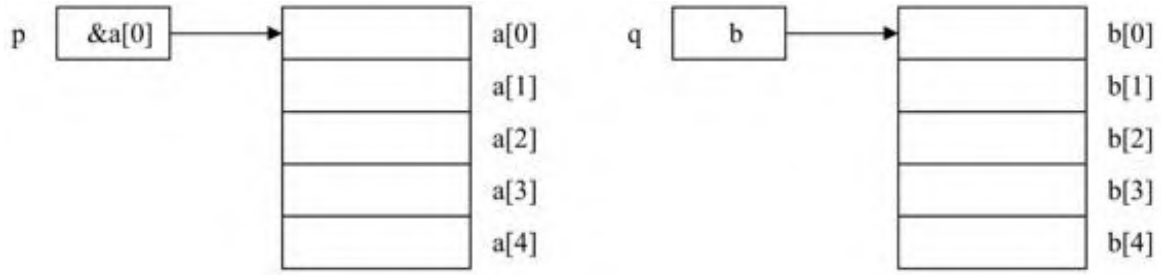


图10.13　指针指向一维数组

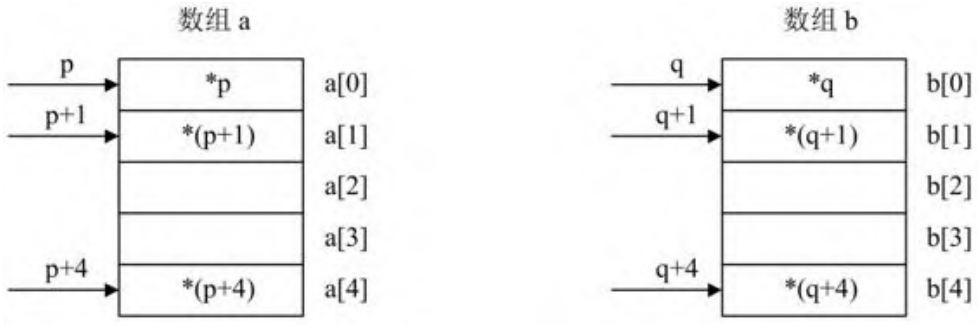


图10.14　通过指针引用数组元素

用a+n表示数组元素的地址，*（a+n）表示数组元素，可以将例10.6中输出数组部分的程序代码改成如下形式（运行结果不变）。

```
printf("array a is:\n");
for(i=0;i<5;i++)
    printf("%5d",*(a+i));                    /*利用*(a+i)形式输出数组 a*/
printf("\n");
printf("array b is:\n");
for(i=0;i<5;i++)
    printf("%5d",*(b+i));                    /*利用*(b+i)形式输出数组 b*/
printf("\n");
```

☑ 表示指针的移动可以使用“++”和“--”这两个运算符。

例如，利用“++”运算符可将数组输出部分的代码改写成如下形式：

```
printf("array a is:\n");
for(i=0;i<5;i++)
    printf("%5d",*p++);                      /*利用*p++形式输出数组 a*/
printf("\n");
printf("array b is:\n");
for(i=0;i<5;i++)
    printf("%5d",*q++);                      /*利用*q++形式输出数组 b*/
printf("\n");
```

还可对例10.6程序再进一步改写，将数组的输入、输出都采用指针自增运算实现。改写后的程序代码如下：

```
#include<stdio.h>
main()
{
    int *p,*q,a[5],b[5],i;
    p=&a[0];
    q=b;
    printf("please input array a:\n");
    for(i=0;i<5;i++)
        scanf("%d",p++);                    /*利用 p++形式输入数组 a*/
    printf("please input array b:\n");
    for(i=0;i<5;i++)
        scanf("%d",q++);                    /*利用 q++形式输入数组 b*/
    p=a;                                    /*注意这行*/
    q=b;                                    /*注意这行*/
    printf("array a is:\n");
    for(i=0;i<5;i++)
        printf("%5d",*p++);                 /*利用*p++形式输出数组 a*/
    printf("\n");
    printf("array b is:\n");
    for(i=0;i<5;i++)
        printf("%5d",*q++);                 /*利用*q++形式输出数组 b*/
    printf("\n");
}
```

比较上面的程序会发现，如果在给数组元素赋值时使用了指针自增运算，而且在输出数组元素时也使用指针变量，则需要加上如下语句：

```
p=a;
q=b;
```

这两行语句的作用是将指针变量p和q重新指向数组a和数组b在内存中的起始位置。若没有该语句，直接使用“*p++”“*q++”的方法进行输出，将会产生错误。

10.2.2 二维数组与指针



定义一个3行5列的二维数组，其在内存中的存储形式如图10.15所示，从中可以看到多种表示二维数组中元素地址的方法。

对于一个m行n列的二维数组，其元素地址的表示方法如下。

- ☑ a表示二维数组的首地址，也表示数组第1行的首地址，a+1表示第2行的首地址，a+m表示第m+1行的首地址。
- ☑ a[0]+n表示数组第1行第n+1个元素的地址，a[m]+n表示第m+1行第n+1个元素的地址。
- ☑ &a[0]表示数组第1行的首地址，&a[m]表示第m+1行的首地址。
- ☑ &a[0][0]既可以表示数组第1行1列的首地址，也可以看作整个数组的首地址。&a[m][n]就是第m+1行n+1列元素的地址。

指针也可表示地址，因此通过指针可以引用二维数组中的元素。

- ☑ *（*（a+m）+n）和*（a[m]+n）含义相同，都表示数组第m+1行第n+1列元素。

【例10.7】输出3*5矩阵（**实例位置：资源包\TM\sl\10\07**）

利用指针对二维数组进行输入和输出，具体代码如下：

```
#include<stdio.h>
main()
{
    int a[3][5],i,j;
    printf("please input:\n");
    for(i=0;i<3;i++)                          /*控制二维数组的行数*/
    {
        for(j=0;j<5;j++)                      /*控制二维数组的列数*/
        {
            scanf("%d",a[i]+j);               /*利用 a[i]+j 形式输入二维数组元素*/
        }
    }
    printf("the array is:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            printf("%5d",*(a[i]+j));          /*利用 a[i]+j 形式输出二维数组元素*/
        }
        printf("\n");
    }
}
```

程序运行结果如图10.16所示。

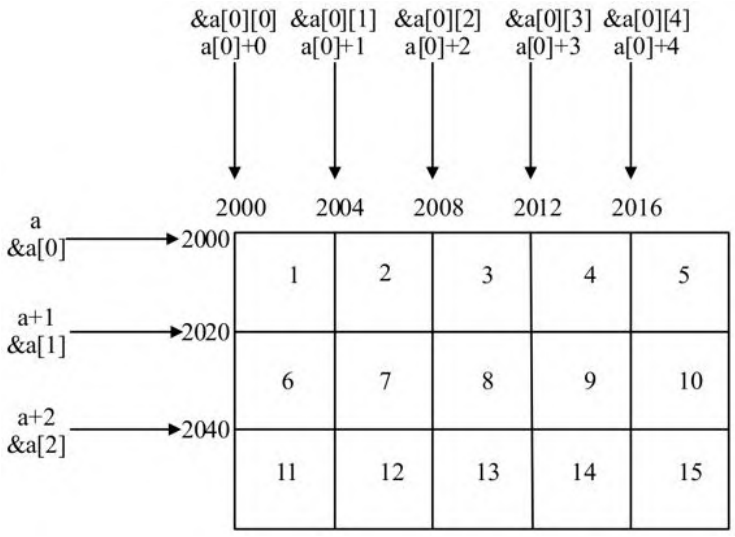


图10.15　二维数组

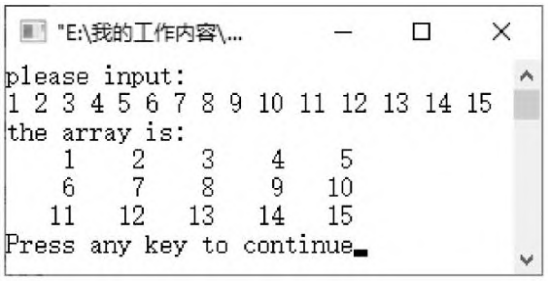


图10.16　二维数组的输入和输出

在运行结果相同的前提下，可将程序改写成如下形式：

```
#include<stdio.h>
main()
{
    int a[3][5],i,j,*p;
    p=a[0];                                  /*定义 p 为第 1 行元素的首地址*/
    printf("please input:\n");
    for(i=0;i<3;i++)                         /*控制二维数组的行数*/
    {
        for(j=0;j<5;j++)                     /*控制二维数组的列数*/
        {
            scanf("%d",p++);                 /*使用 p++形式输入二维数组元素*/
        }
    }
    p=a[0];                                  /*再次定义 p 为第 1 行元素的首地址*/
    printf("the array is:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            printf("%5d",*p++);              /*使用 p++形式输出二维数组元素*/
        }
        printf("\n");
    }
}
```

【例10.8】输出第3行元素（实例位置：资源包\TM\sl\10\08）

将一个3行5列二维数组的第3行元素输出，具体代码如下：

```
#include<stdio.h>
main()
{
    int a[3][5],i,j,(*p)[5];
    p=&a[0];                                    /*定义 p 为第 1 行元素的首地址*/
    printf("请输入 3*5 数组内容 t:\n");
    for(i=0;i<3;i++)                            /*控制二维数组的行数*/
        for(j=0;j<5;j++)                        /*控制二维数组的列数*/
            scanf("%d",*(p+i)+j);               /*使用*(p+i)+j 形式输入二维数组元素*/
    p=&a[2];                                    /*定义 p 为第 3 行元素的首地址*/
    printf("第三行是:\n");
    for(j=0;j<5;j++)
        printf("%5d",*(*p+j));                  /*使用*(p+i)+j 形式输出二维数组元素*/
    printf("\n");
}
```

程序运行结果如图10.17所示。

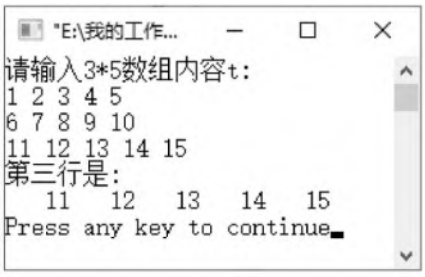


图10.17　输出第3行元素

也可采用*（a+i）+j形式输入二维数组元素，代码如下:

```
scanf("%d",*(a+i)+j);                           /*使用*(a+i)+j 形式输入二维数组元素*/
```

同样可以采用*（a+i）+j形式输出二维数组元素，代码如下:

```
printf("%5d",*(*(a+i)+j));                      /*使用*(a+i)+j 形式输出二维数组元素*/
```

技巧

利用指针引用二维数组的关键是要记住*（a+i）与a[i]是等价的。

10.2.3 字符串与指针



可以通过两种方式访问一个字符串，一是使用字符数组，二是使用字符指针。

【例10.9】使用指针输出字符串（实例位置：资源包\TM\sl\10\09）

利用字符型指针输出字符串“hello mingri”，具体代码如下：

```
#include<stdio.h>
main()
{
    char *string="hello mingri";
    printf("%s",string);                    /*输出字符串*/
}
```

程序运行结果如图10.18所示。首先定义了字符型指针变量string，用字符串常量“hello mingri”为其赋初值。注意，这里并不是把“hello mingri”中的所有字符存放到string中，只是把该字符串中第一个字符的地址赋给指针变量string，如图10.19所示。

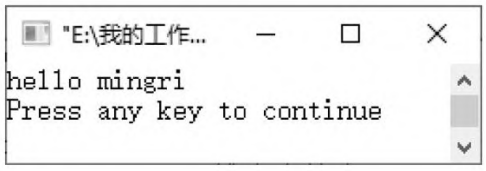


图10.18 字符型指针应用

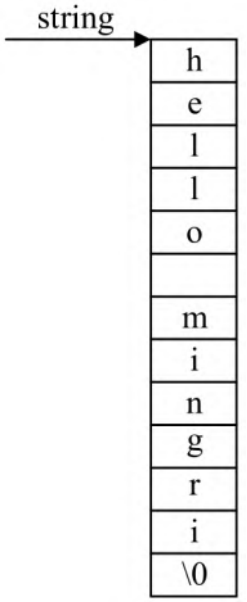


图10.19　字符指针

语句“char *string="hello mingri";”等价于下面两条语句：

```
char *string;
string="hello mingri";
```

【例10.10】实现字符串复制功能（实例位置：资源包\TM\sl\10\10）

不使用string.h函数库中的函数，利用指针实现字符串复制功能，具体代码如下：

```
#include <stdio.h>
void _strcopy(char *str1, char *str2);
int main()
{
    char str1[] = "you are beautiful";          /*定义字符串 1*/
    char str2[] = "";                           /*定义字符串 2*/
    _strcopy(str1, str2);                       /*调用自定义函数_strcopy */
    printf("复制之后的字符串为：%s\n", str2);    /*输出复制之后结果*/
    return 0;
}
void _strcopy(char *str1, char *str2)           /*用户自定义函数，用于实现字符串复制*/
{
    while(*str1 != '\0')                        /*将字符串 1 中的字符逐个复制到字符串 2 中*/
    {
        *str2 = *str1;
        str1++;
        str2++;
    }
    *str2 = '\0';                               /*在字符串 2 的末尾添加结束符 "\0" */
}
```

程序运行结果如图10.20所示。自定义函数中，参数为两个指向字符型数据的指针变量。利用while循环，先将str1指向的内容赋给str2指向的元素，然后str1和str2分别加1，指向下一个元素，直到*str1的值为“\0”为止。

这里有一点需要注意，就是str1和str2的值是同步变化的，如图10.21所示。若str1处在str11的位置，str2就处在str21的位置；若str1处在str12的位置，str2就处在str22的位置。

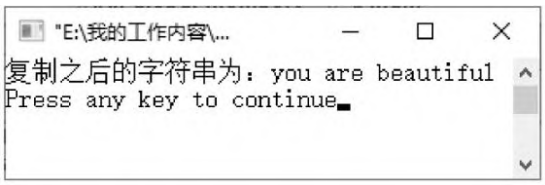


图10.20　复制字符串

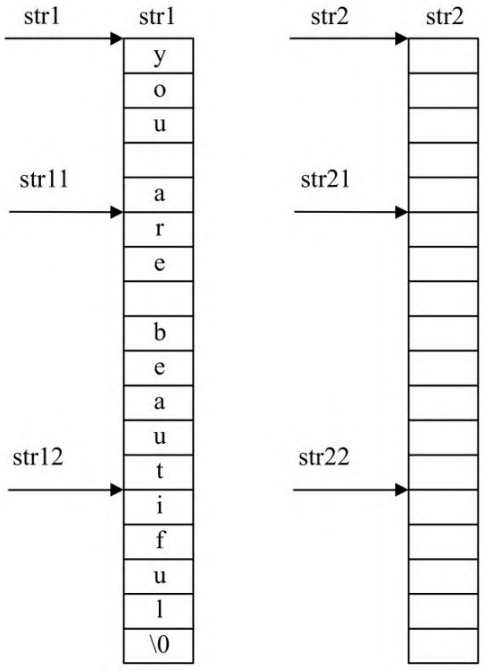


图10.21　str1和str2同步变化

10.2.4　指针数组



字符串数组有别于8.3节中学过的字符数组。字符数组是一个一维数组，而字符串数组是以字符串作为数组元素的数组，可以将其看成一个二维字符数组。下面定义一个简单的字符串数组：

```
char country[5][20]=
{
    "China",
    "Japan",
    "Russia",
    "Germany",
    "Switzerland"
}
```

上述代码中，字符型数组变量country被定义为含有5个字符串的数组，每个字符串的长度都必须小于20（这里要考虑字符串最后的“\0”）。

通过观察上面定义的字符串数组可以发现，像"China"和"Japan"这样的字符串的长度仅为5，加上字符串结束符也仅为6，而内存中却要给它们分别分配一个20字节的空间，空间浪费较大。为了解决这个问题，可以使用指针数组，使每个指针指向所需要的字符常量。这种方法需要在数组中保存字符指针，虽然也占用空间，但要远少于字符串数组占用的空间。

什么是指针数组呢？一个数组，如果其元素均为指针类型数据，则该数组称为指针数组。也就是说，指针数组中的每一个元素都相当于一个指针变量。一维指针数组的定义形式如下：

```
类型名 数组名[数组长度]
```

【例10.11】汉译英填空（实例位置：资源包\TM\sl\10\11）

英语小测，有一题是根据汉语填写英语。语文、数学、英语、化学、生物、物理的英语分别为Chinese、math、English、chemistry、biology和physics，用指针数组输出对应的英文。

```
#include<stdio.h>
int main()
{
    int i;
    char *test[]={"Chinese",
                  "math",
                  "English",
                  "chemistry",
                  "biology",
                  "physics"};                    /*定义一个指针数组，对其中的元素赋初值*/
    printf("答案是:\n");
    for(i=0;i<6;i++)
        printf("%s\n",test[i]);                  /*输出指针数组中的各元素*/
    return 0;
}
```

程序运行结果如图10.22所示。

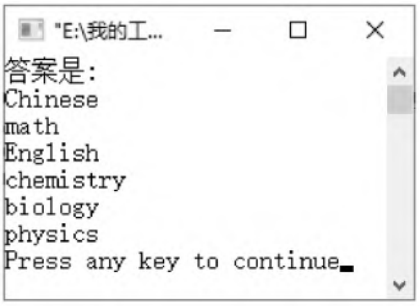


图10.22　输出答案

编程训练（答案位置：资源包\TM\sl\10\编程训练\）

训练3：最受欢迎的座位号　某班级有5*5个座位，输出位置最好的一行座位号。（提示：第2行最受欢迎）。

训练4：输出12个月份　定义指针数组并赋初值，将12个月份输出。运行结果如下：

```
January
February
March
April
May
June
July
August
September
October
November
December
```

10.3　指向指针的指针



一个指针可以指向整型变量、实型变量、字符型变量，当然也可以指向另一个指针，此时称其为指向指针的指针。如图10.23所示，整型变量i的地址是&i，将其值传递给指针变量p1，则p1指向i；同时，将p1的地址&p1传递给p2，则p2指向p1。这里的p2就是指向指针变量的指针。

指向指针的指针定义如下：

```
类型标识符 **指针变量名;
```

例如，“int **p;”的含义为定义一个指针变量p，它指向另一个指针变量，该指针变量又指向一个基本整型变量。由于指针运算符“*”自右至左结合，所以上述定义相当于“int *（*p）;”。

知道如何定义指向指针的指针后，可以将图10.23用图10.24更形象地表示出来。

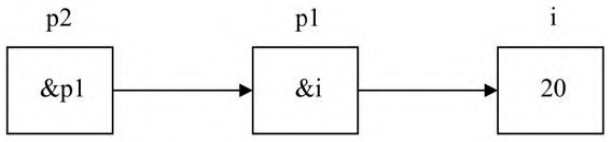


图10.23　指向指针的指针（1）

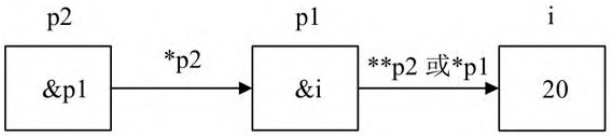


图10.24　指向指针的指针（2）

【例10.12】输出化学周期表中的金属元素（实例位置：资源包\TM\sl\10\12）

利用指向指针的指针，输出化学周期表前20个元素中的金属元素，具体代码如下：

```
#include<stdio.h>
int main()
{
    int i;                                          /*定义循环控制变量*/
    char **p;                                       /*定义一个指向指针的指针变量*/
    char *element[]=                                /*定义一个指针数组，并为其赋初值*/
    {
        "锂",
        "铍",
        "钠",
        "镁",
        "铝",
        "钾",
        "钙"
    };
    for(i=0;i<7;i++)                                /*输出指针数组中的各元素*/
    {
        p=element+i;
        printf("%s\n",*p);
    }
    return 0;
}
```

程序运行结果如图10.25所示。

【例10.13】统计偶数个数（实例位置：资源包\TM\sl\10\13）

利用指向指针的指针输出一维数组中其值为偶数的元素，并统计偶数的个数，具体代码如下：

```
#include<stdio.h>
main()
{
    int a[10],*p1,**p2,i,n=0;                       /*定义数组、指针等为基本整型*/
    printf("please input:\n");
    for(i=0;i<10;i++)                               /*给数组 a 中各元素赋值*/
        scanf("%d",&a[i]);
    p1=a;                                           /*将数组 a 的首地址赋给 p1*/
    p2=&p1;                                         /*将指针 p1 的地址赋给 p2*/
    printf("the array is:");
    for(i=0;i<10;i++)
    {
        if(*(*p2+i)%2==0)                           /*如果是偶数*/
        {
            printf("%5d",*(*p2+i));                 /*输出数组中的偶数元素*/
            n++;                                    /*偶数个数加 1*/
        }
    }
    printf("\n");
    printf("the number is:%d\n",n);
}
```

程序运行结果如图10.26所示。

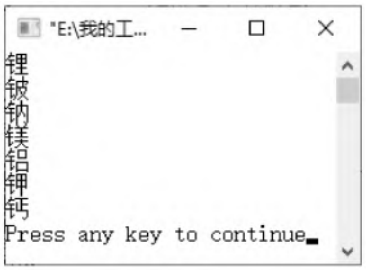


图10.25　输出化学周期表中金属的元素

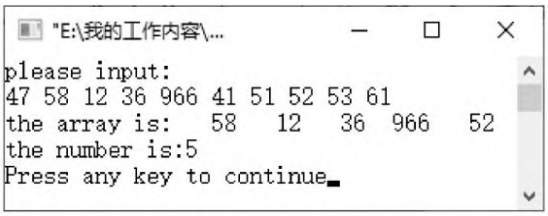


图10.26　输出偶数

程序中，将数组a的首地址赋给指针变量p1，又将指针变量p1的地址赋给p2，要通过这个双重指针变量p2访问数组中的元素，就要一层层地来分析。首先看*p2的含义，*p2指向的是指针变量p1所存放的内容，即数组a的首地址，要想取出数组a中的元素，就必须在*p2前面再加一个指针运算符“*”。

根据前面讲过的指针的用法，还可将例10.14改写成如下形式：

```
#include<stdio.h>
main()
{
    int a[10],*p1,**p2,n=0;                          /*定义数组、指针等为基本整型*/
    printf("please input:\n");
    for(p1=a;p1-a<10;p1++)                           /*指针 p1 从 a 的首地址开始变化*/
    {
        p2=&p1;                                      /*将指针 p1 的地址赋给 p2*/
        scanf("%d",*p2);                             /*通过指针变量给数组元素赋初值*/
    }
    printf("the array is:");
    for(p1=a;p1-a<10;p1++)
    {
        p2=&p1;                                      /*再次将指针 p1 的地址赋给 p2*/
        if(**p2%2==0)                                /*如果是偶数*/
        {
            printf("%5d",**p2);                      /*将数组中的偶数元素输出*/
            n++;                                     /*偶数个数加 1*/
        }
    }
    printf("\n");
    printf("the number is:%d\n",n);
}
```

编程训练（答案位置：资源包\TM\sl\10\编程训练\）

训练5：模拟网上买衣服的场景　小红想要在淘宝上买件衣服，因为她有某卖家的优惠券，所以联系客服找她想要买的衣服。于是客服给她一个链接，找到衣服的价格是559元。模拟场景，找到衣服价格。（提示：使用int **p）。运行结果如下：

```
衣服的价格是：
559
```

训练6：给名字排序　自定义一个排序的函数，在主函数中调用这个函数同时采用指针的指针输出排序后的名字，运行结果如下：

```
一共有5个名字，分别是：
赵小平
周芷若
宇文玥
南乔
楚小乔
```

```
按照姓名排序之后分别是：
楚小乔
南乔
宇文玥
赵小平
周芷若
```

10.4 指针变量作函数参数



通过前面的介绍可知，整型变量、实型变量、字符型变量、数组名和数组元素等均可作为函数参数。此外，指针型变量也可以作为函数参数，这里具体进行介绍。

【例10.14】交换两个变量的值（实例位置：资源包\TM\sl\10\14）

在本实例中，用指针做函数参数，交换两个变量的值。在主函数中，调用自定义函数，利用指针将两个数交换，具体代码如下：

```
#include <stdio.h>
void swap(int *a,int *b)                    /*自定义 swap 函数，形参为两个指针*/
{
    int tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}
main()
{
    int x,y;
    int *p_x,*p_y;
    printf("请输入两个数：\n");
    scanf("%d",&x);                         /*输入两个数*/
    scanf("%d",&y);
    p_x=&x;
    p_y=&y;
    swap(p_x,p_y);                          /*调用 swap*/
    printf("x=%d\n",x);
    printf("y=%d\n",y);
}
```

程序运行结果如图10.27所示。swap函数是用户自定义函数，在main函数中调用该函数交换变量a和b的值，swap函数的两个形参被传入了两个地址值，也就是传入了两个指针变量。在swap函数的函数体内使用整型变量tmp作为中间变量，将两个指针变量所指向的数值进行交换。在main函数内首先获取输入的两个数值，分别传递给变量x和y，调用swap函数将变量x和y的数值互换。

如果swap函数的形参不是指针而是普通整型，调用swap函数后能实现两数互换吗？将前述程序改成如下形式：

```
#include<stdio.h>
void swap(int a,int b)                    /*自定义 swap 函数，形参为两个整数*/
{
    int tmp;
    tmp=a;
    a=b;
    b=tmp;
}
void main()
{
    int x,y;
    printf("请输入两个数：\n");
    scanf("%d",&x);
    scanf("%d",&y);
    swap(x,y);                            /*调用 swap 函数*/
    printf("x=%d\n",x);
    printf("y=%d\n",y);
}
```

程序运行结果如图10.28所示。可见，程序并没有交换x和y的值，这是为什么呢？在函数调用过程中，主调用函数与被调用函数之间有一个数值传递过程，这种数值传递是单向的，只能把实参的值传递给形参。也就是说，在自定义函数体中，即便此时形参的值发生了改变，也无法再传递回来，因此实参的值不会发生变化。这就是为什么普通整型作为函数形参时不能实现x和y值互换的原因。

通过指针传递参数，可以减少值传递带来的开销，也可以使函数调用不产生值传递。

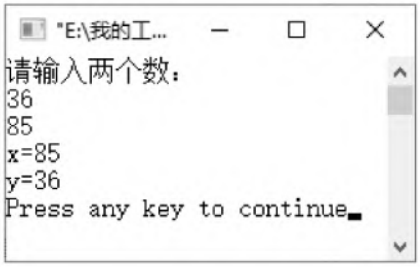


图10.27　指针做函数参数

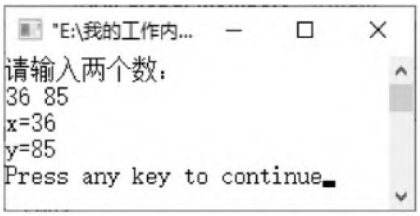


图10.28　普通类型做函数参数

【例10.15】将3个数按降序输出（实例位置：资源包\TM\sl\10\15）

首先定义交换函数swap，参数是指针变量；然后再定义比较大小的函数exchange，参数也是指针变量，并且在exchange函数中嵌套交换函数swap。在主函数中，调用exchange函数将输入的3个数按降序输出，具体代码如下：

```
#include<stdio.h>
void swap(int *p1, int *p2)                          /*自定义交换函数 swap*/
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
void exchange(int *pt1, int *pt2, int *pt3)          /*自定义排序函数 exchange */
{
    if (*pt1 < *pt2)
        swap(pt1, pt2);                              /*调用 swap 函数*/
    if (*pt1 < *pt3)
        swap(pt1, pt3);
    if (*pt2 < *pt3)
        swap(pt2, pt3);
}
int main()
{
    int a, b, c, *q1, *q2, *q3;
    puts("Please input three key numbers you want to rank:");
    scanf("%d,%d,%d", &a, &b, &c);
    q1 = &a;                                         /*将变量的地址赋给指针变量*/
    q2 = &b;
    q3 = &c;
    exchange(q1, q2, q3);                            /*调用 exchange 函数*/
    printf("\n%d,%d,%d\n", a, b, c);
    return 0;
}
```

程序运行结果如图10.29所示。

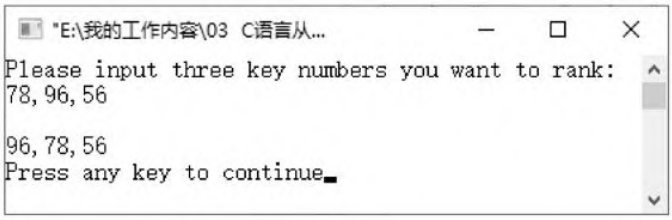


图10.29　嵌套的函数调用

程序创建了一个自定义函数swap，用于交换两个变量的值。还创建了一个exchange函数，其作用是将3个数由大到小排序，在exchange函数中调用swap函数。这里，swap和exchange函数都是以指针变量作为形参。程序运行时，通过键盘输入3个数a、b、c，分别将a、b、c的地址赋给q1、q2、q3，调用exchange函数，将指针变量作为实参，将实参变量的值传递给形参变量，此时q1和pt1都指向变量a，q2和pt2都指向变

量b，q3和pt3都指向变量c；在exchange函数中又调用了swap函数，当执行swap（pt1，pt2）时，pt1也指向了变量a，pt2指向了变量b，这一过程如图10.30所示。

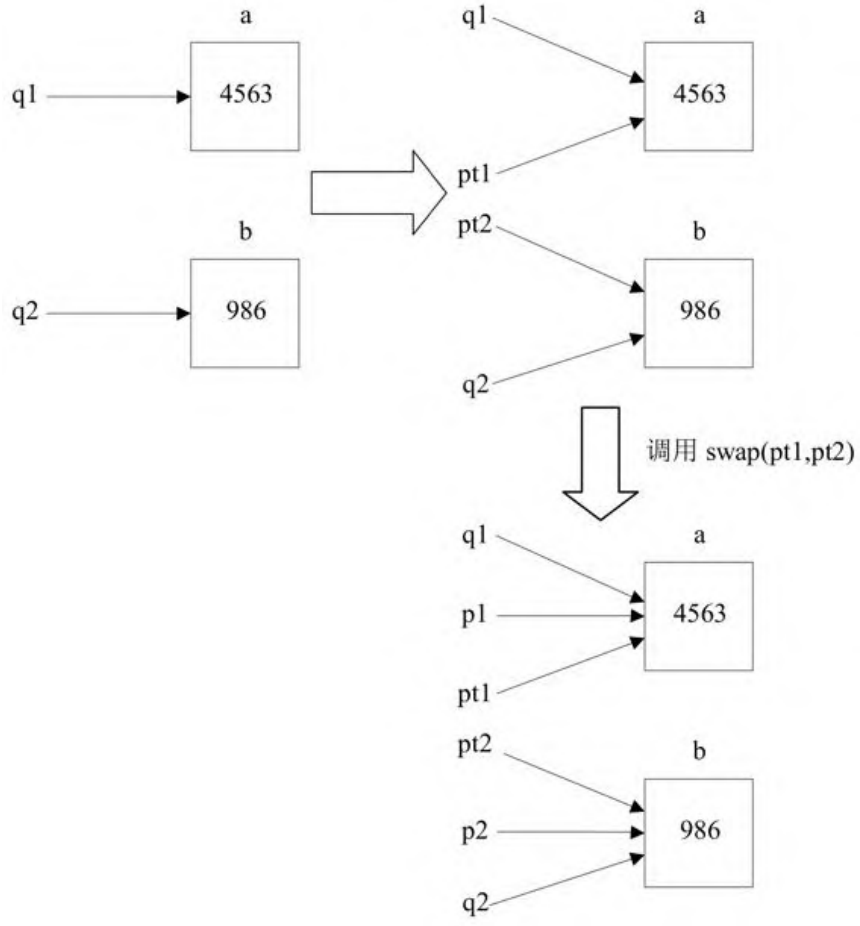


图10.30　嵌套调用时指针的指向情况

C语言中实参变量和形参变量之间的数据传递是单向的“值传递”方式。指针变量作函数参数也是如此，调用函数不可能改变实参指针变量的值，但可以改变实参指针变量所指向变量的值。

下面介绍如何使用指向数组的指针变量作为函数参数。这里，形参和实参均为指针变量。

【例10.16】计算新生总人数（实例位置：资源包\TM\sl\10\16）

利用指针变量作函数参数编写程序，根据输入的各班人数求刚入学的初中新生的总人数。

```
#include<stdio.h>
void SUM(int *p,int n)                          /*自定义 SUM 函数，用于求和*/
{
    int i,sum=0;
    for(i=0;i<n;i++)                            /*循环各班人数并求和*/
        sum=sum+*(p+i);                         /*相加*/
    printf("新生总人数是:%d\n",sum);
}
int main()
{
    int *pointer,a[10],i;
    pointer=a;
    printf("请输入每个班级人数:\n");
    for(i=0;i<10;i++)                           /*输入各班的人数*/
        scanf("%d",&a[i]);
    SUM(pointer,10);                            /*调用 SUM 函数求总人数*/
    return 0;
}
```

程序运行结果如图10.31所示。在自定义函数SUM中使用指针变量作形式参数，在主函数中实际参数pointer是一个指向一维数组a的指针，虚实结合，被调用函数SUM中的形式参数p得到pointer的值，指向了内存中存放的一维数组。

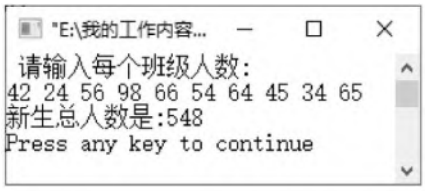


图10.31　输出新生总人数

冒泡排序是C语言中比较经典的例子，也是读者应该牢牢掌握的一种算法，下面分析如何使用指针变量作为函数参数来实现冒泡排序。

【例10.17】实现冒泡排序（实例位置：资源包\TM\sl\10\17）

冒泡排序的基本思想：如果要对n个数进行冒泡排序，则要进行n-1轮比较，在第一轮比较中要从后到前进行n-1次两两比较，在第j轮比较中要进行n-j次两两比较。使用指针实现冒泡法排序，代码如下：

```
#include<stdio.h>
void order(int *p,int n)                          /*自定义冒泡排序函数 order */
{
    int i,t,j;
    for(i=0;i<n-1;i++)
        for(j=0;j<n-1-i;j++)
            if(*(p+j)>*(p+j+1))                   /*判断相邻两个元素的大小*/
            {
                t=*(p+j);                         /*借助中间变量 t 进行值互换*/
                *(p+j)=*(p+j+1);
                *(p+j+1)=t;
            }
    printf("排序后的数组：");
    for(i=0;i<n;i++)
    {
        if(i%5==0)                                /*以每行 5 个元素的形式输出*/
            printf("\n");
        printf("%5d",*(p+i));                     /*输出数组中排序后的元素*/
    }
    printf("\n");
}
main()
{
    int a[20],i,n;
    printf("请输入数组元素的个数：\n");
    scanf("%d",&n);                               /*输入数组元素的个数*/
    printf("请输入各个元素：\n");
    for(i=0;i<n;i++)
        scanf("%d",a+i);                          /*给数组元素赋初值*/
    order(a,n);                                   /*调用 order 函数*/
}
```

程序运行结果如图10.32所示。

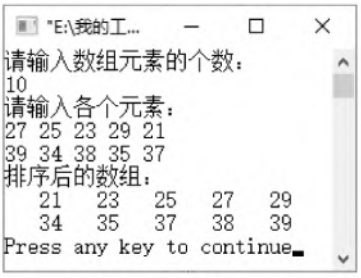


图10.32 冒泡排序结果

前面两个实例都是用一个指向数组的指针变量作函数参数。在10.3节中介绍过指向指针的指针，这里就来通过一个实例介绍如何用指向指针的指针作函数参数。

【例10.18】水果首字母升序排名（实例位置：资源包\TM\sl\10\18）

大福超市员工布置水果区，店长要求按照水果名称升序的顺序摆放。各水果的名称及对应单价如下：

苹果（apple）：3.50　　橘子（tangerine）：2.50　　柚子(grapefriu)：3.00　　香蕉（banana）：2.00

橙子（orange）：2.99　菠萝（pineapple）：4.99　葡萄(grape)：5.00　　火龙果（pitaia）：6.80

编写程序，按照水果名称的首字母将水果升序排序。

```
#include <stdio.h>
#include <string.h>
void sort(char *strings[], int n)                    /*自定义排序函数 sort*/
{
    char *temp;
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (strcmp(strings[i], strings[j]) > 0)  /*比较两个字符串的大小*/
            {
                temp = strings[i];                   /*如果前面的字符比后面的大，则互换*/
                strings[i] = strings[j];
                strings[j] = temp;
            }
        }
    }
}
int main()
{
    int n = 8;
    int i;
    char **p;                                        /*定义字符型、指向指针的指针*/
    char *name[] =
    {
        "apple",
        "tangerine",
        "grapefriu",
        "banana",
        "orange",
        "pineapple",
        "grape",
        "pitaia"
    };
    p = name;
    sort(p, n);                                      /*调用排序函数 sort*/
    printf("排序后的水果单如下：\n");
    for (i = 0; i < n; i++)
        printf("%s\n", name[i]);                     /*输出排序后的字符串*/
    return 0;
}
```

程序运行结果如图10.33所示。

下面将通过一个二维数组使用指针变量作为函数参数的实例来加深读者对该部分知识的理解。

【例10.19】找出数组每行中最大数并求和（实例位置：资源包\TM\sl\10\19）

利用指针作函数参数，找出一个数组每行最大的数，并将这些数相加求和。具体代码如下：

```
#include<stdio.h>
#define N 4
void max(int (*a)[N],int m)                          /*自定义 max 函数，求数组每行的最大元素*/
{
    int value,i,j,sum=0;
    for(i=0;i<m;i++)
    {
        value=*(*(a+i));                             /*将一行中的首个元素赋给 value*/
        for(j=0;j<N;j++)
            if(*(*(a+i)+j)>value)                    /*判断其他元素是否小于 value*/
                value=*(*(a+i)+j);                   /*把比 value 大的数重新赋给 value*/
        printf("第%d 行：最大数是：%d\n",i,value);
        sum=sum+value;
    }
    printf("\n");
    printf("每行中最大数相加之和是：%d\n",sum);
}
main()
{
    int a[3][N],i,j;
    int (*p)[N];
    p=&a[0];
    printf("please input:\n");
    for(i=0;i<3;i++)                                 /*给二维数组元素赋值*/
        for(j=0;j<N;j++)
            scanf("%d",&a[i][j]);
    max(p,3);                                        /*调用 max 函数，指针变量作函数参数*/
}
```

程序运行结果如图10.34所示。

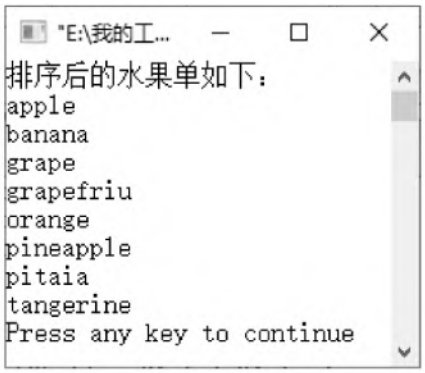


图10.33　字符串排序

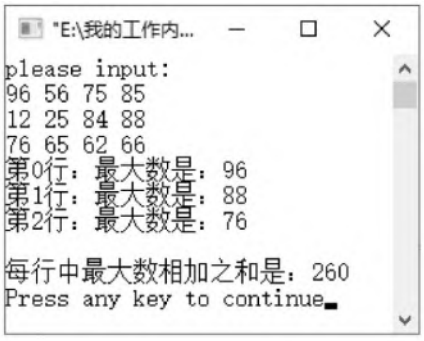


图10.34　输出每行最大的数并求和

现在来总结一下：

（1）数组名就是数组的首地址，因此例10.17中可以将数组名作为实参传递给指针型形式参数。

（2）当形参为数组时，实参也可以为指针变量。可将例10.17改写成如下形式：

```
#include<stdio.h>
void order(int a[],int n)
{
    int i,t,j;
    for(i=0;i<n-1;i++)
        for(j=0;j<n-1-i;j++)
            if(*(a+j)>*(a+j+1))                    /*判断相邻两个元素的大小*/
            {
                t=*(a+j);
                *(a+j)=*(a+j+1);
                *(a+j+1)=t;                        /*借助中间变量 t 进行值互换*/
            }
    printf("排序后的数组：");
    for(i=0;i<n;i++)
    {
        if(i%5==0)                                 /*以每行 5 个元素的形式输出*/
        printf("\n");
        printf("%5d",*(a+i));                      /*输出数组中排序后的元素*/
    }
    printf("\n");
}
main()
{
    int a[20],i,n;
    int *p;
    p=a;                                           /*定义 p 为数组 a 的首地址*/
    printf("请输入数组元素的个数：\n");
    scanf("%d",&n);                                /*输入数组元素的个数*/
    printf("请输入各个元素：\n");
    for(i=0;i<n;i++)                               /*给数组元素赋初值*/
        scanf("%d",p++);                           /*再次定义 p 为数组 a 的首地址*/
    p=a;
    order(p,n);                                    /*调用 order 函数*/
}
```

上述程序中，形参是数组，实参是指针变量。注意倒数第3行语句：

```
p=a;                                               /*再次定义 p 为数组 a 的首地址*/
```

该语句不可少，如果将其省略，则后面调用order函数时，参数p指向的就不再是数组a的首地址，从而导致程序运行出错，这点需要读者加以注意。

编程训练（答案位置：资源包\TM\sl\10\编程训练\）

训练7：汇总一周收纳的钱数　自定义一个用指针变量作参数的函数，功能是将银行每周入账的钱数相加，然后在主函数中调用这个函数，

运行结果如下：

```
请输入一周收纳的钱数：
2345000 677854 78959 567844 356789 56785 678458
面值的总数是:4761689
```

训练8：求长方体的侧面积　输入长、宽、高，求长方体的体积及正面、侧面、顶面面积。要求设计一个函数，参数为长方体长、宽、高的指针。运行结果如下：

```
正面面积是：20
侧面面积是：12
顶面面积是：15
```

10.5　返回指针值的函数



指针变量也可以指向一个函数。函数在编译时会被分配一个入口地址，该入口地址就称为函数的指针。可以用一个指针变量指向函数，然后通过该指针变量调用此函数。

一个函数可以带回一个整型值、字符值、实型值等，也可以带回指针型的数据，即地址。其概念与之前介绍的类似，只是带回的值的类型是指针类型而已。返回指针值的函数简称为指针函数。

定义指针函数的一般形式如下：

```
类型名 *函数名(参数表列);
```

下面的定义中，fun是函数名，调用它以后能得到一个指向整型数据的指针。x和y是函数fun的形式参数，这两个参数均为基本整型。函数名前面有一个“*”，表示此函数是指针型函数。类型说明是int，表示返回的指针指向整型变量。

```
int *fun(int x,int y)
```

【例10.20】求长方形周长（实例位置：资源包\TM\sl\10\20）

输入长和宽，求长方形的周长，使用返回值为指针的函数解决。代码如下：

```
#include<stdio.h>
int *per(int a,int b);
int Perimeter;
void main()
{
    int iWidth,iLength;
    int *iResult;
    printf("请输入长方形的长：\n");
    scanf("%d",&iLength);
    printf("请输入长方形的宽：\n");
    scanf("%d",&iWidth);
    iResult=per(iWidth,iLength);                    /*调用函数 per */
    printf("长方形的周长是：");
    printf("%d\n",*iResult);
}

int *per(int a,int b)                               /*自定义求周长函数 per，返回值为指针*/
{
    int *p;
    p=&Perimeter;
    Perimeter=(a+b)*2;
    return p;                                       /*返回指针*/
}
```

程序运行结果如图10.35所示。

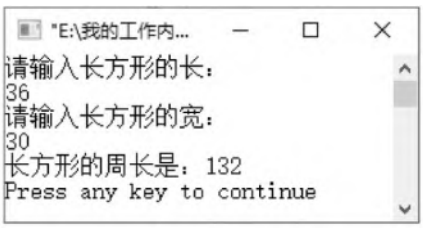


图10.35　求长方形周长

注意，例10.20完全可以用前面学过的更简单的形式来实现，这里只是为了帮助大家更好地理解返回值为指针的函数及其应用。

编程训练（答案位置：资源包\TM\sl\10\编程训练\）

训练9：计算零件总数　师徒两人合做一批零件，徒弟做了9个，比师傅少做21个，问这批零件共有多少个。将数据利用指针返回。运行结果如下：

```
请输入徒弟比师傅少做数值：
21
请输入徒弟做了零件的数值：
9
这批零件总数是:30
```

训练10：植树　三、四年级共植树128棵，四年级比三年级多植20棵，求两个年级各植树多少棵。运行结果如下：

```
请输入总共植树的数量：
128
请输入四年级比三年级多植树的数量：
20
三年级植树的数量是：54
四年级植树的数量是：74
```

10.6　指针数组作main函数的参数



main函数是运行所有程序的入口，由系统调用。当处于操作命令状态下，输入main所在的文件名，系统即调用main函数。

main函数一般是没有参数的。但实际上，main函数也可以是有参函数，其带参形式如下：

```
main(int argc,char *argv[])
```

有两个参数，一个整型和一个指针数组。当一个C源程序经过编译、链接后，会生成扩展名为.exe的可执行文件，该文件可直接在操作系统下运行。对于main函数来说，其实际参数和命令是一起给出的，也就是一个命令行包括命令名和需要传给main函数的参数。命令行的一般形式如下：

```
命令名  参数1 参数2 … 参数n
```

其中，“命令名”就是可执行文件的文件名，命令名和其后参数之间须用空格分隔。

命令行与main函数的参数间存在一定关系。假设命令行为：

```
file1 happy bright glad
```

这里，file1为文件名，就是file1.c经编译、链接后生成的可执行文件file1.exe，其后跟3个参数。以上命令行与main函数中的形式参数关系如下：参数argc记录了命令行中命令与参数的个数，这里包括file1、happy、bright和glad，共4个，指针数组的大小由参数的值决定，这里为char *argv[4]，该指针数组的取值情况如图10.36所示。

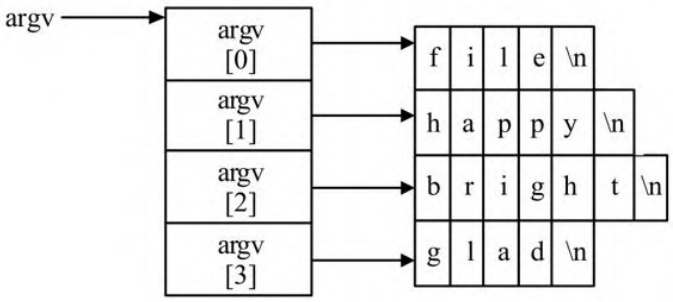


图10.36　指针数组取值

利用指针数组作main函数的形参，可以向程序传送命令行参数。

说明

参数字符串的长度是不定的，参数数量也是任意的，并不规定具体个数。

【例10.21】输出main函数的参数形式（实例位置：资源包\TM\sl\10\21）

```
#include<stdio.h>
main(int argc,char *argv[])                    /*main 函数为带参函数*/
{
    printf("the list of parameter:\n");
    printf("命令名：\n");
    printf("%s\n",*argv);
    printf("参数个数：\n");
    printf("%d\n",argc);

}
```

程序运行结果如图10.37所示。

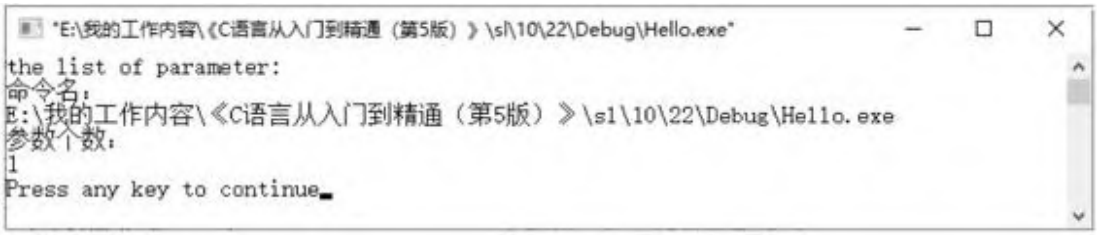


图10.37　输出参数内容

编程训练（答案位置：资源包\TM\sl\10\编程训练\）

训练11：参数的接受度　设计一个程序，该程序接受命令行参数，输出“您好，[输入的参数]”，如果没有参数，则输出“请输入你的名字：”，然后退出程序。

训练12：给程序加权限　利用main函数参数给程序添加权限判断，用户运行程序时需要输入账号和密码，并输出相应的提示。（提示：生成.exe文件，然后使用cmd运行）

10.7　实践与练习

（答案位置：资源包\TM\sl\10\实践与练习\）

综合练习1：2022冬奥会口号　利用两种方式输出字符串：第一种是采用for循环遍历字符数组，输出字符串；第二种是利用指针直接输出字符串。运行效果如下：

```
* * * * * * * * * * * * * * * * * * * * * * ** * * * * * * *
* * * *
2022年重大项目是：2022年北京冬季奥运会
2022年奥运会口号是：纯洁的冰雪，激情的约会
* * * * * * * * * * * * * * * * * * * * * * ** * * * * * * *
* * * *
```

综合练习2：古诗词填空　春眠不觉晓，处处闻啼鸟。___________，花落知多少。利用指针将缺少的诗句输出在控制台上。运行效果如下：

```
答案是：

▼ ▼ ▼ ▼ ▼ ▼ ▼
    夜来风雨声
▲ ▲ ▲ ▲ ▲ ▲ ▲
```

综合练习3：小猪过河　9只小猪渡河，只有一个木筏，一次能载3只猪，只有一只猪会划船。问至少往返几次，9只小猪才能全部渡过河。利用“*&”输出结果。运行效果如下：

```
会划船的猪每次只能载2只猪过去
会划船的猪一直在船上
因此至少4次能全部渡过河
```

综合练习4：计算公路长度　某工程队修路，第一天修了600米，第二天修了全长的20%，第三天修了全长的25%，三天共修了全长的75%，求这条公路全长多少米，同时输出此公里数变量的内存地址。运行结果如下：

```
----------------------------
 公路全长是 2000 米
----------------------------
**************
 它的地址是:1703724
**************
```

综合练习5：统计单词数量　利用指针，统计“I have a dream.”中的单词个数。运行结果如下：

```
□ □ □ □ □ □ □ □ □
     一共有 4 个单词
□ □ □ □ □ □ □ □ □
```

综合练习6：倒序输出数据　输入10个数，存放在数组中。编写程序，将数组中的元素值按照相反顺序存放。运行结果如下：

```
请输入 10 个元素（用空格隔开）：
88 56 45 25 66 95 86 33 16 2

正序输出结果:

   88   56   45   25   66
   95   86   33   16    2
倒序输出结果:

    2   16   33   86   95
   66   25   45   56   88
```

综合练习7：寻找逗号　使用指针寻找字符串“Life is brief，and then you die，you know?”中“，”的位置。运行结果如下：

```
，的位置是：13
，的位置是：31
```

综合练习8：模拟影院售票　某家电影院售票情况如图10.38所示，用指针求出剩余的电影票数。其中，1表示有座，0表示没座，统计1的数量就是统计剩余的电影票数。运行结果如下：

```
请输入电影院售票情况：
0 1 1 0
1 1 1 1
1 0 1 1
```

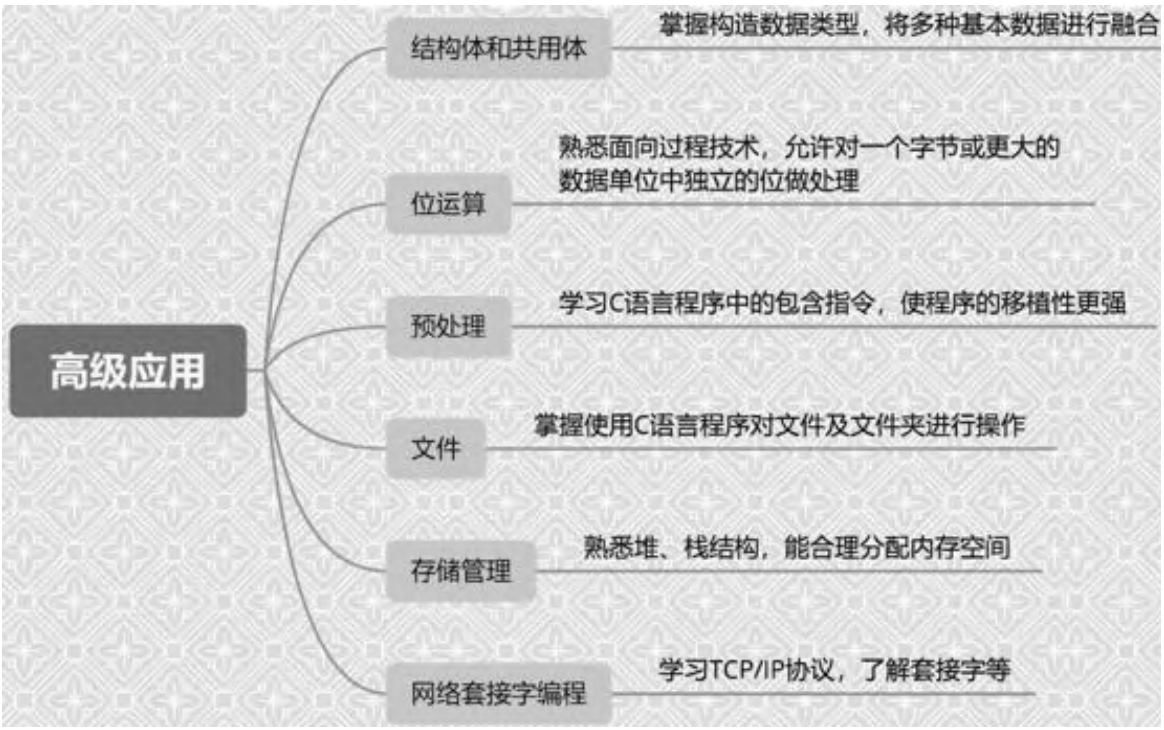
```
1 1 1 1
剩余的票数是：13
```

0	1	1	0
1	1	1	1
1	0	1	1
1	1	1	1

图10.38　电影院售票情况

第3篇
高级应用

本篇介绍了结构体和共用体、位运算、预处理、文件、存储管理和网络套接字编程的内容。学习完本篇知识，读者将能够设计出较复杂的应用程序，并且涉及的范围更广。



第11章
结构体和共用体

迄今为止，我们在程序中用到的都是基本数据类型。但实际开发中，有时简单的变量类型无法满足程序中各种复杂的数据要求，因此C语言还提供了构造类型。构造类型数据是由基本类型数据按照一定规则组成的。

本章致力于使读者了解结构体的概念，掌握结构体和共用体的使用方法，并结合具体实例使大家对结构体和共用体数组、指针有一个更为深刻的理解。

本章的知识架构及重难点如下：



11.1　结构体

前面我们学习过许多基本数据类型，如整型int、字符型char等，还学习了数组这种构造类型。数组中，所有的数据都是同一类型，调用起来

非常方便。

除此以外，有时我们需要定义一些复杂的数据类型，它可能包括多个不同属性，每个属性需要用不同的类型来表示。该怎么实现呢？C语言中，可以把一些有内在联系的不同变量组织起来，封装成一个整体，即定义成一个结构体（structure），以此来表示一种新的数据类型。之后，就可以像处理基本数据类型那样，对结构体类型进行各种操作。

11.1.1 结构体类型的声明



结构体是一种构造类型，它由若干成员组成。其成员可以是一个基本数据类型，也可以是另一个构造类型。声明一个结构体的过程，就是创建一种新的类型名的过程。

声明结构体时使用的关键字是struct，其一般形式如下：

```
struct 结构体名
{
    成员列表
};
```

关键字struct表示声明的是一个结构体，“结构体名”表示要创建的新类型名，大括号中的“成员列表”包括构成该结构体的所有成员。注意，声明结构体时大括号后的分号“;”不能遗漏。

例如，商品一般包括产品名称、形状、颜色、功能、价格和产地等属性，如图11.1所示。很显然，“商品”这种类型并不能使用任何一种基本类型来表示，而需要为其构造一个结构体。

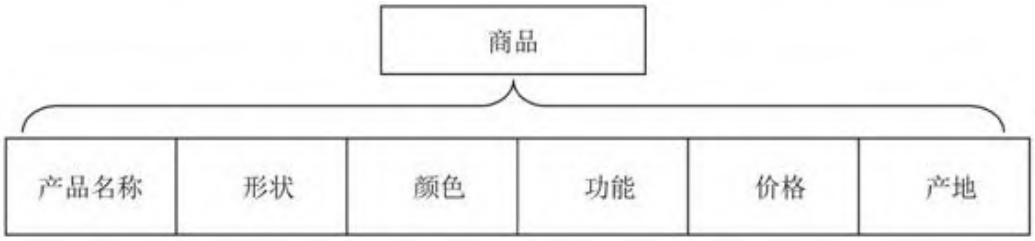


图11.1 “商品”类型

下面来创建“商品（Product）”这种类型，声明代码如下：

```
struct Product                          /*声明商品结构体*/
{
    char cName[10];                     /*产品名称*/
    char cShape[20];                    /*形状*/
    char cColor[10];                    /*颜色*/
    char cFunc[20];                     /*功能*/
    int iPrice;                         /*价格*/
    char cArea[20];                     /*产地*/
};
```

上述代码使用关键字struct声明了一个名为Product的结构体类型，在结构体中定义了6个变量，分别表示产品名称、形状、颜色、功能、价格和产地，并为其设置了相对应的类型。

11.1.2 结构体变量的定义



声明完结构体后，就创建了一种新的类型名，后续就可以使用这种新的类型名定义变量。定义结构体变量的方式有如下3种。

（1）先声明结构体类型，再定义变量。

例如，使用前面声明的Product结构体类型定义两个结构体变量product1和product2，代码如下：

```
struct Product                          /*声明结构体*/
{
    char cName[10];                     /*产品名称*/
    char cShape[20];                    /*形状*/
    char cColor[10];                    /*颜色*/
    char cFunc[20];                     /*功能*/
    int iPrice;                         /*价格*/
    char cArea[20];                     /*产地*/
};
struct Product product1;                /*定义结构体类型变量*/
struct Product product2;
```

struct Product是结构体类型名，而product1和product2是结构体变量名。既然都是使用Product类型定义的变量，那么这两个变量就具有相同的结构。

技巧

在一些大型开发中，为了便于修改和使用，常常将结构体类型的声明放在一个头文件中，这样在其他源文件中如果需要使用该结构体类型，则可以用#include命令将该头文件包含到源文件中。

定义一个基本类型的变量与定义一个结构体类型变量的不同之处在于：定义结构体变量不仅要求指定变量为结构体类型，而且要求指定为某一特定的结构体类型，如struct Product；而定义基本类型的变量时（如整型变量），只需要指定int型即可。

说明

定义结构体变量后，系统就会为其分配内存单元。例如，product1和product2在内存中各占84字节（10+20+10+20+4+20）。

（2）声明结构类型的同时定义结构体变量。例如：

```
struct Product
{
    char cName[10];                    /*产品名称*/
    char cShape[20];                   /*形状*/
    char cColor[10];                   /*颜色*/
    int iPrice;                        /*价格*/
    char cArea[20];                    /*产地*/
}product1,product2;                    /*定义结构体变量*/
```

（3）直接定义结构体类型变量（此时不需要给出结构体名称）。例如：

```
struct
{
    char cName[10];                    /*产品名称*/
    char cShape[20];                   /*形状*/
    char cColor[10];                   /*颜色*/
    int iPrice;                        /*价格*/
    char cArea[20];                    /*产地*/
}product1,product2;                    /*定义结构体变量*/
```

需要注意的是，类型与变量是不同的。例如，只能对变量进行赋值操作，不能对类型进行操作。这就像使用int型定义变量iInt，可以为iInt赋值，但不能为int赋值。编译时，不对类型分配空间，只对变量分配空间。

结构体的成员仍然可以是结构体类型的变量。例如：

```
struct date                            /*日期结构体*/
{
    int year;                          /*年*/
    int month;                         /*月*/
    int day;                           /*日*/
};
struct student                         /*学生结构体*/
{
    int num;                           /*学号*/
    char name[30];                     /*姓名*/
    char sex;                          /*性别*/
    int age;                           /*年龄*/
    struct date birthday;              /*出生日期*/
}student1,student2;
```

以上代码声明了一个date结构体类型，成员包括年、月、日；还声明了一个student结构类型，并且定义了两个结构体变量student1和

student2。在struct student结构体类型中，可以看到有一个成员是表示学生的出生日期，使用的是struct date结构体类型。

11.1.3 结构体变量的引用



定义了结构体类型变量以后，就可以引用该变量，引用形式为“结构变量名.成员名”。例如：

```
product1.cName="Icebox";
product1.iPrice=2000;
```

对结构体变量进行赋值、存取或运算，实质上就是对结构体成员进行操作。上面的赋值语句就是对product1结构体变量中的成员cName和iPrice两个变量进行赋值。

需要注意的是，不能直接将结构体变量作为一个整体进行输入和输出。例如，不能将product1和product2采用如下方式输出：

```
printf("%s%s%s%d%s",product1);
printf("%s%s%s%d%s",product2);
```

如果成员本身又属于一个结构体类型，就需要使用若干个成员运算符“.”找到最低一级的成员，对其进行赋值、存取以及运算操作。例如，对上面定义的student1变量中的出生日期进行赋值：

```
student1.birthday.year=1986;
student1.birthday.month=12;
student1.birthday.day=6;
```

注意

不能使用student1.birthday访问student1变量中的成员birthday，因为birthday本身也是一个结构体变量。

结构体变量的成员可以像普通变量一样，进行各种运算。例如：

```
product2.iPrice=product1.iPrice+500;
product1.iPrice++;
```

因为“.”运算符的优先级最高，所以product1.iPrice++是product1.iPrice成员进行自加运算，而不是先对iPrice进行自加运算。

还可以对结构体变量成员的地址进行引用，也可以对结构体变量的地址进行引用，例如：

```
scanf("%d",&product1.iPrice);                    /*输入成员 iPrice 的值*/
printf("%o",&product1);                          /*输出 product1 的首地址*/
```

【例11.1】展示老师基本信息（实例位置：资源包\TM\sl\11\01）

本实例首先定义一个表示老师的结构体，成员包括姓名、年龄和教龄；然后在主函数中使用该结构体定义一个老师，先赋值，再输出。

```
#include "stdio.h"
#include<string.h>
struct Teacher                                  /*声明教师结构体*/
{
    char name[64];                              /*姓名*/
    int age;                                    /*年龄*/
    int seniority;                              /*教龄*/
 };

int main()
{
    struct Teacher a_teacher;                   /*定义结构体变量*/
    strcpy(a_teacher.name, "明师");             /*将姓名复制给结构体变量*/
    a_teacher.age = 35;                         /*年龄*/
    a_teacher.seniority = 10;                   /*教龄*/
    printf("姓名:%s\n",a_teacher.name);         /*输出结构体变量*/
    printf("年龄:%d\n",a_teacher.age);
    printf("教龄:%d\n",a_teacher.seniority);
    return 0;
}
```

（1）先声明结构体类型Teacher，以表示老师这种特殊的类型，在结构体中定义了有关的成员。

（2）在主函数main中使用struct Teacher定义结构体变量a_teacher，然后根据赋值的信息，引用结构体变量a_teacher中的成员，使用printf函数将其进行输出显示。

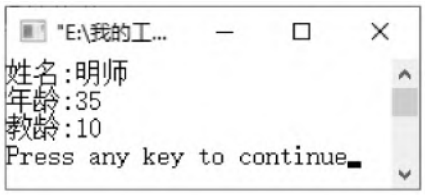


图11.2　引用结构体变量

运行程序，显示效果如图11.2所示。

11.1.4　结构体类型的初始化



结构体类型与其他基本类型一样，也可以在定义结构体变量时指定初始值。例如：

```
struct Student
{
    char cName[20];
    char cSex;
    int iGrade;
} student1={"HanXue","W",3};                    /*定义结构体变量并设置初始值*/
```

定义的变量后面使用等号，初始化值放在大括号中，数据顺序与结构体的成员列表顺序一致。

【例11.2】输出新来学生信息（实例位置：资源包\TM\sl\11\02）

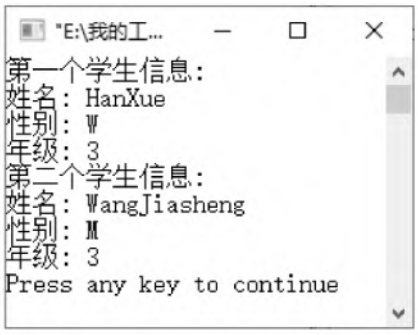
本实例演示了两种初始化结构体的方式，一种是在声明结构体及定义变量的同时进行初始化，另一种是在定义结构体变量后进行初始化。

```
#include<stdio.h>
struct Student                                  /*学生结构体*/
{
    char cName[20];                             /*姓名*/
    char cSex;                                  /*性别*/
    int iGrade;                                 /*年级*/
} student1={"HanXue",'W',3};                    /*方式 1：定义结构体时，直接赋初始值*/

int main()
{
    struct Student student2={"WangJiasheng",'M',3};   /*方式 2：后续再定义变量并赋初始值*/
    printf("第一个学生信息:\n");                  /*以下将第一个结构体中的数据输出*/
    printf("姓名: %s\n",student1.cName);
    printf("性别: %c\n",student1.cSex);
    printf("年级: %d\n",student1.iGrade);
    printf("第二个学生信息:\n");                  /*以下将第二个结构体中的数据输出*/
    printf("姓名: %s\n",student2.cName);
    printf("性别: %c\n",student2.cSex);
    printf("年级: %d\n",student2.iGrade);
    return 0;
}
```

（1）声明结构体时定义student1并对其初始化，待赋值内容放在后面大括号中，每个数据都与结构体的成员数据相对应。

（2）在main函数中使用声明的结构体类型struct Student定义变量student2，并且进行初始化的操作。



（3）最后将两个结构体变量中的成员进行输出，并比较二者数据的区别。

图11.3　结构体类型的初始化操作

运行程序，显示效果如图11.3所示。

误区警示

并不是所有的结构体成员都可以赋值，如果某个成员使用const做了限定，就不可以再对其赋值。

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练1：显示猫的信息　定义一个表示猫的结构体，输出它的年龄、体重、名字等信息。运行结果如下：

```
猫名:糖块
猫龄:1
猫体重:4.90
猫的性别:公
```

训练2：输出某台汽车信息　定义一个表示汽车的结构体，输出车的品牌名、颜色、车长、承载量等信息。运行结果如下：

```
车牌名:路虎
颜色是:黑色
车长:4.85米
可承载5人
```

11.2　结构体数组

当要定义10个整型变量时，可以使用数组的形式。当要定义10个结构体类型变量时，也可以使用数组的形式，这时的数组被称为结构体数

组。结构体数组与普通数组的区别在于：数组中的每个元素都是根据要求定义的结构体类型，而不是基本类型。

11.2.1 定义结构体数组



定义结构体数组的方式与定义结构体变量的方法相同，其一般形式如下：

```
struct 结构体名
{
    成员列表;
}数组名;
```

例如，定义一个学生信息的结构体数组，其中包含5个学生的信息，代码如下：

```
struct Student                          /*学生结构体*/
{
    char cName[20];                     /*姓名*/
    int iNumber;                        /*学号*/
    char cSex;                          /*性别*/
    int iGrade;                         /*年级*/
} student[5];                           /*定义结构体数组*/
```

上述代码中，定义结构体类型的同时定义了结构体数组student[5]。除此以外，也可以先声明结构体类型再定义结构体数组：

```
struct Student student[5];              /*定义结构体数组*/
```

或者直接定义结构体数组（此时不需要给出结构体名称）：

```
struct                                  /*学生结构体*/
{
    …                                   /*成员列表，省略过程*/
} student[5];                           /*定义结构体数组*/
```

上面的代码都定义了一个结构体数组，其中每个元素都为struct Student类型的数据，每个数据中又有4个成员变量，如图11.4所示。

	cName	iNumber	cSex	iGrade
student[0]	WangJiasheng	12062212	M	3
student[1]	YuLongjiao	12062213	W	3
student[2]	JiangXuehuan	12062214	W	3
student[3]	ZhangMeng	12062215	W	3
student[4]	HanLiang	12062216	M	3

图11.4　结构体数组

数组中各数据在内存中的存储是连续的，如图11.5所示。

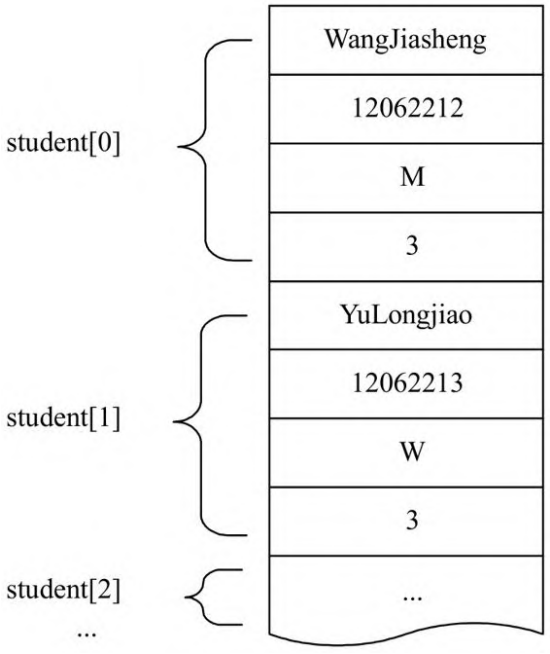


图11.5　数组数据在内存中的存储形式

11.2.2　初始化结构体数组



初始化结构体数组的一般形式如下：

```
struct 结构体名
{
    成员列表;
}数组名={初始值列表};
```

例如，为学生结构体数组进行初始化操作，代码如下：

```
struct Student                                  /*学生结构体*/
{
    char cName[20];                             /*姓名*/
    int iNumber;                                /*学号*/
    char cSex;                                  /*性别*/
    int iGrade;                                 /*年级*/
} student[5]={{"WangJiasheng",12062212,'M',3},
              {"YuLongjiao",12062213,'W',3},
              {"JiangXuehuan",12062214,'W',3},
              {"ZhangMeng",12062215,'W',3},
              {"HanLiang",12062216,'M',3}};     /*定义数组并设置初始值*/
```

为数组进行初始化时，最外层的大括号表示列出的是数组中的元素。因为每个元素都是结构体类型，所以也都使用大括号括起来，其中包含每一个结构体元素的成员数据。

定义数组student时，也可以不指定数组中的元素个数，这时编译器会根据数组后面的初始化值列表中给出的元素个数，来确定数组中元素的个数。例如：

```
student[ ]={…};
```

同样，也可以先声明结构体数组，后续再进行初始化。

【例11.3】显示销售前5名的产品（实例位置：资源包\TM\sl\11\03）

某网站“双11”做促销活动，利用结构体数组编写程序，将销量排行前5名的商品信息输出。代码如下：

```
#include<stdio.h>
struct goods                                        /*商品结构体*/
{
      char cName[20];                               /*品牌名*/
      int iNumber;                                  /*销售量*/
} goods[5]={{"面膜",1458792365},                    /*定义结构体变量并初始化*/
            {"洁面",325656550},
            {"洗发露",324655854},
            {"护发素",256897412},
            {"卸妆膏",155655655}};

int main()                                          /*主函数 main*/
{
      int i;
      for(i=0;i<5;i++)                              /*循环输出数组中的元素数据*/
      {
            printf("NO%d 产品:\n",i+1);
            printf("品牌名是: %s, 销量: %d\n",goods[i].cName,goods[i].iNumber);
            printf("\n");                           /*空格行*/
      }
      return 0;
}
```

（1）声明struct goods结构体类型，同时定义结构体数组goods，并初始化数据。

（2）定义的数组包含5个元素，输出时使用for语句进行循环输出操作。其中，定义变量i为控制循环操作。因为数组的下标是从0开始的，所以为变量i赋值为0。

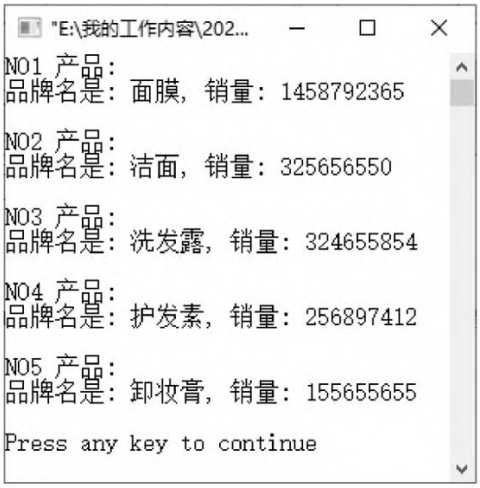


图11.6 输出学生信息

（3）在for语句中，先显示每个学生的输出次序，其中因为i的初值为0，所以要加上1。之后将数组中的元素所表示的数据输出，这时变量i作为数组的下标，然后通过结构体成员的引用得到正确的数据，最后将其输出。

运行程序，显示效果如图11.6所示。

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练3：输出一窝宠物猫的信息　利用结构体数组，输出5只宠物猫的信息。运行结果如下：

```
第1只猫：
猫名：  糖块,猫龄：1元,猫的体重：4.90,猫的性别：公

第2只猫：
猫名：  小点,猫龄：1元,猫的体重：3.50,猫的性别：公

第3只猫：
猫名：  团团,猫龄：1元,猫的体重：3.90,猫的性别：母

第4只猫：
猫名：  琉球,猫龄：1元,猫的体重：3.70,猫的性别：公

第5只猫：
猫名：  点点,猫龄：1元,猫的体重：3.30,猫的性别：母
```

训练4：显示明日科技编著的图书　利用结构体数组，输出目前京东在售，由明日科技出版的5本C语言图书的书名、价格等信息。运行结果如下：

```
第1本图书：
图书名：  零基础学C语言（全彩版）,价格：61.10元

第2本图书：
图书名：  C语言从入门到项目实践（全彩版）,价格：99.80元

第3本图书：
图书名：  玩转C语言程序设计（全彩版）,价格：36.10元

第4本图书：
图书名：  C语言精彩编程200例（全彩版）,价格：69.80元

第5本图书：
图书名：  C语言项目开发实战入门（全彩版）,价格：52.30元
```

11.3 结构体指针

一个指向结构体变量的指针，指向的是结构体变量的起始地址。除此以外，指针变量还可以指向结构体数组以及数组中的元素。

11.3.1 指向结构体变量的指针



定义结构体指针的一般形式如下：

```
结构体类型 *指针名;
```

例如，定义一个指向struct Student结构类型的pStruct指针变量，代码如下：

```
struct Student *pStruct;
```

指针指向的是结构体变量的地址，因此可以使用指针来访问结构体中的成员，方式有两种。

第一种方法是使用成员运算符“.”引用结构成员，形式为“(*pStruct).成员名”。例如，pStruct指针指向student1结构体变量，可以采用如下方式引用其中的成员。

```
(*pStruct).iNumber=12061212;
```

注意，*pStruct一定放在括号内，这是因为成员运算符“.”的优先级最高，如果不使用括号，就会先执行“.”运算然后才是“*”运算。

【例11.4】登记新员工信息（实例位置：资源包\TM\sl\11\04）

某公司招聘新员工后，需要为其安排工位和所属部门。利用结构体类型指针编写程序，将所有新员工的信息输出。

```
#include<stdio.h>                              /*包含头文件*/
struct People                                  /*定义人员结构体*/
{
    char cName[20];                            /*姓名*/
    int iNumber;                               /*职位号*/
    char cS[20];                               /*部门*/
}people={"张伟",14,"开发部"};                  /*对结构体变量初始化*/

int main()
{
    struct People* pStruct;                    /*定义结构体类型指针*/
    pStruct=&people;                           /*指针指向结构体变量*/
    printf("-----信息如下-----\n");            /*提示信息*/
    printf("姓名: %s\n",(*pStruct).cName);     /*使用指针输出结构体成员*/
    printf("职工号: %d\n",(*pStruct).iNumber);
    printf("部门: %s\n",(*pStruct).cS);
    return 0;/*程序结束*/
}
```

（1）首先声明结构体类型People，同时定义变量people并进行初始化。

（2）定义结构体指针变量pStruct，然后执行“pStruct=&people;”使指针指向people变量。

（3）输出消息提示，然后在printf函数中使用指针引用成员变量，将员工信息输出。

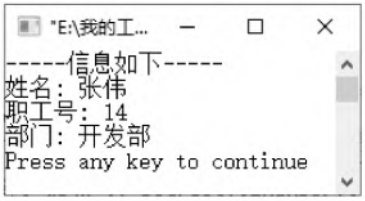


图11.7　使用成员运算符引用结构体变量的成员

运行程序，显示效果如图11.7所示。

第二种方法是使用指向运算符“->”引用结构成员，形式为“pStruct ->成员名”。例如，使用指向运算符引用一个变量的成员：

```
pStruct->iNumber=12061212;
```

注意

在使用“->”引用成员时，要注意分析以下情况。

- ☑ pStruct->iGrade：表示指向的结构体变量中成员iGrade的值。
- ☑ pStruct->iGrade++：表示指向的结构体变量中成员iGrade的值，使用后该值加1。
- ☑ ++pStruct->iGrade：表示指向的结构体变量中成员iGrade的值加1，计算后再进行使用。

总结一下，假如student为结构体变量，pStruct为指向结构体变量的指针，则以下3种引用形式的效果是完全等价的。

- ☑ student.成员名。
- ☑ （*pStruct）.成员名。
- ☑ pStruct->成员名。

【例11.5】打印购票信息（实例位置：资源包\TM\sl\11\05）

在本实例中，定义结构体变量但不对其进行初始化操作，使用指针指向结构体变量并为其成员进行赋值操作。

```
#include<stdio.h>
#include<string.h>

struct Ticket                                   /*定义车票结构体*/
{
    char cName[20];                             /*姓名*/
    int iNumber;                                /*票价*/
    char S[20];                                 /*乘车区间*/
    char cAddress[20];                          /*车次*/
    char c[20];                                 /*开车时间*/
}ticket;

int main()
{
    struct Ticket* pStruct;                     /*定义结构体变量*/
    pStruct=&ticket;                            /*指针指向结构体变量*/
    strcpy(pStruct->cName,"张伟");              /*赋值*/
    pStruct->iNumber=285;
    strcpy(pStruct->S,"长春-北京");
    strcpy(pStruct->cAddress,"D71");
    strcpy(pStruct->c,"2021 年 2 月 20 日 09:08 开");
    printf("姓名: %s\n",ticket.cName);          /*输出结构体成员*/
    printf("票价: %d 元\n",ticket.iNumber);
    printf("乘车区间: %s\n",ticket.S);
    printf("车次: %s\n",ticket.cAddress);
    printf("开车时间: %s\n",ticket.c);
    return 0;
}
```

（1）使用strcpy函数将一个字符串常量复制到成员变量中。使用strcpy函数前，要在程序中包含头文件string.h。

（2）为成员赋值时，使用的是“->”运算符引用的成员变量，在程序的最后使用结构体变量和“.”运算符直接将成员的数据进行输出。

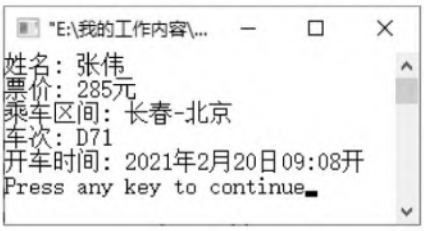


图11.8　使用指向运算符引用结构体变量的成员

运行程序，显示效果如图11.8所示。

11.3.2　指向结构体数组的指针



结构体指针变量指向结构体数组时，指针变量的值就是结构体数组的首地址。还可以直接指向结构体数组中的元素，这时指针变量的值就是该结构体数组元素的首地址。

例如，定义一个结构体数组student[5]，使用结构体指针指向该数组，代码如下：

```
struct Student* pStruct;
pStruct=student;
```

因为数组不使用下标时表示的是数组的第一个元素的地址，所以指针指向数组的首地址。如果想利用指针指向第3个元素，则在数组名后附加下标，然后在数组名前使用取地址符号“&”。

```
pStruct=&student[2];
```

【例11.6】显示5名同学信息（实例位置：资源包\TM\sl\11\06）

在本实例中，使用之前声明的学生结构类型定义结构体数组，并对其进行初始化操作。通过指向该数组的指针，将其中元素的数据进行输出显示。

```
#include<stdio.h>
struct Student                                  /*学生结构体*/
{
    char cName[20];                             /*姓名*/
    int iNumber;                                /*学号*/
    char cSex;                                  /*性别*/
    int iGrade;                                 /*年级*/
} student[5]={{"WangJiasheng",12062212,'M',3},
              {"YuLongjiao",12062213,'W',3},
              {"JiangXuehuan",12062214,'W',3},
              {"ZhangMeng",12062215,'W',3},
              {"HanLiang",12062216,'M',3}};     /*定义数组并设置初始值*/

int main()
{
    struct Student* pStruct;
    int index;
    pStruct=student;
    for(index=0;index<5;index++,pStruct++)
    {
        printf("NO%d 名学生:\n",index+1);       /*首先输出学生的名次*/
        /*使用变量 index 做下标，输出数组中的元素数据*/
        printf("姓名: %s, 学号: %d\n",pStruct->cName,pStruct->iNumber);
        printf("性别: %c, 年级: %d\n",pStruct->cSex,pStruct->iGrade);
        printf("\n");                           /*空格行*/
    }
    return 0;
}
```

（1）定义一个结构体数组student[5]，定义结构体指针变量pStruct指向该数组的首地址。

（2）for循环中，pStruct开始指向数组的首地址，也就是第一个元素的地址，因此使用pStruct->引用的是第一个元素中的成员。使用输出函数显示成员变量表示的数据。

（3）一次循环结束后，循环变量进行自加操作，pStruct也执行自加运算。注意，pStruct++表示pStruct的增加值为一个数组元素大小，即数组元素中的第二个元素student[1]。



图11.9　5名同学信息

运行程序，显示效果如图11.9所示。

注意

（++pStruct）->Number与（pStruct++）->Number的区别在于：前者先执行++操作，使pStruct指向下一个元素的地址，然后取得该元素的成员值；后者先取得当前元素的成员值，再使pStruct指向下一个元素的地址。

11.3.3 结构体作为函数参数



使用函数时，其参数也可以是结构体变量。形式有3种：使用结构体变量作为函数参数；使用指向结构体变量的指针作为函数参数；使用结构体变量的成员作为函数参数。

1．使用结构体变量作为函数参数

结构体变量作为函数实参时，采取的是“值传递”方式，即会将结构体变量所占内存单元的内容全部顺序传递给形参，形参也必须是同类型的结构体变量。例如：

```
void Display(struct Student stu);
```

在形参位置使用结构体变量，但是函数调用期间，形参也要占用内存单元。这种传递方式在空间和时间上开销都比较大。另外，根据函数参数

传值方式，如果在函数内部修改了变量中成员的值，则改变的值不会返回到主调函数中。

【例11.7】求语数外3科平均分（实例位置：资源包\TM\sl\11\07）

在本实例中，声明一个简单的结构体类型表示学生成绩，并初始化。编写一个函数，使得该结构体变量作为函数的参数，在函数中显示此学生成绩并计算语数外3科的平均分，具体代码如下：

```
#include<stdio.h>
struct Student                                  /*学生结构体*/
{
    char cName[20];                             /*姓名*/
    float fScore[3];                            /*分数*/
}student={"SuYuQun",98.5f,89.0,93.5f};          /*定义变量*/

void Display(struct Student stu)                /*形参为结构体变量*/
{
    printf("-----信息如下-----\n");             /*提示信息*/
    printf("姓名: %s\n",stu.cName);             /*引用结构成员*/
    printf("语文: %.2f\n",stu.fScore[0]);
    printf("数学: %.2f\n",stu.fScore[1]);
    printf("英语: %.2f\n",stu.fScore[2]);
    printf("平均分:%.2f\n",(stu.fScore[0]+stu.fScore[1]+stu.fScore[2])/3);  /*计算平均分数*/
}

int main()
{
    Display(student);                           /*调用函数，结构体变量作为实参进行传递*/
    return 0;
}
```

（1）声明一个简单的结构体表示学生分数信息，在其中定义一个字符数组表示名称，再定义一个实型数组表示3个学科的分数。声明结构体的同时定义变量，并进行初始化。

（2）定义一个名为Display的函数，用结构体变量作为函数形参。函数体中，使用参数stu引用结构体中的成员，输出学生的姓名和3个学科的成绩，并在最后通过表达式计算出平均成绩。

（3）在主函数main中，使用student结构体变量作为参数，调用Display函数。

运行程序，显示效果如图11.10所示。

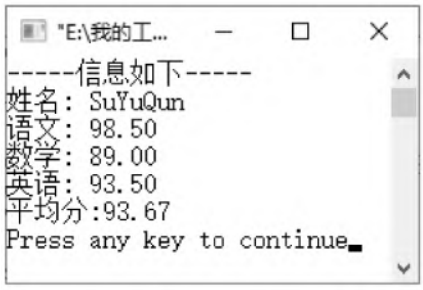


图11.10　结构体变量作函数参数

2．使用指向结构体变量的指针作为函数参数

在使用结构体变量作为函数的参数时，在传值的过程中空间和时间的开销比较大，那么有没有一种更好的传递方式呢？有！就是使用结构体变量的指针作为函数的参数进行传递。

在传递结构体变量的指针时，只是将结构体变量的首地址进行传递，并没有将变量的副本进行传递。例如，声明一个传递结构体变量指针的函数如下：

```
void Display(struct Student* stu)
```

这样使用形参stu指针就可以引用结构体变量中的成员了。需要注意的是，因为传递的是变量的地址，如果在函数中改变成员中的数据，那么返回主调函数时变量会发生改变。

【例11.8】修改英语成绩（实例位置：资源包\TM\sl\11\08）

对例11.7做点小改动，使用结构体变量的指针作为函数参数，并在函数中改动结构体成员的数据。通过前后两次输出，比较二者的区别。

```
#include<stdio.h>
struct Student                                  /*学生结构体*/
{
    char cName[20];                             /*姓名*/
    float fScore[3];                            /*分数*/
}student={"SuYuQun",98.5f,89.0,93.5f};          /*定义变量*/

void Display(struct Student* stu)               /*形参为结构体变量的指针*/
{
    printf("-----信息如下-----\n");             /*提示信息*/
    printf("姓名: %s\n",stu->cName);            /*使用指针引用结构体变量中的成员*/
    printf("英语: %.2f\n",stu->fScore[2]);      /*输出英语的分数*/
    stu->fScore[2]=90.0f;                       /*更改成员变量的值*/
}

int main()
{
    struct Student* pStruct=&student;           /*定义结构体变量指针*/
    Display(pStruct);                           /*调用函数，结构变量作为实参进行传递*/
    printf("修改之后的英语: %.2f\n",pStruct->fScore[2]);  /*输出成员的值*/
    return 0;
}
```

（1）函数参数是结构体变量指针，因此函数体中通过“->”引用成员数据。为了简化操作，只将英语成绩进行输出，并且最后更改成员的数据。

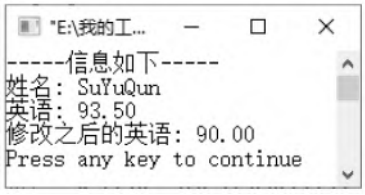


图11.11　结构体变量指针作函数参数

（2）主函数main中，先定义结构体变量指针，并将结构体变量的地址传递给指针，将指针作为函数参数进行传递。函数调用完后，再显示一次变量中的成员数据。通过结果可以看到，在函数中通过指针改变成员的值，在返回主调用函数中值发生变化。

运行程序，显示效果如图11.11所示。

说明

为了直观地看出函数传递的参数是结构体变量指针，定义了一个指针变量指向结构体。实际上，可以直接传递结构体变量的地址作为

函数参数，如“Display（&student）;”。

3．使用结构体变量的成员作为函数参数

该方式与普通变量作为实参是一样的，是传值方式传递（实参与形参类型一致）。例如：

```
Display(student.fScore[0]);
```

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练5：给汽车加油　定义一个汽车结构体，结构体中包含剩余汽油的升数。定义一个加油函数，将汽车作为函数参数，每执行一次该函数，剩余的汽油升数都会增加2。运行结果如下：

```
加油前:12.00
加油后:14.00
加油后:16.00
加油后:18.00
```

训练6：输出某书架信息　定义一个书架结构体。为结构体定义变量初始化赋值，然后使用指针指向该结构变量，最后通过指针引用变量中的成员进行显示。运行结果如下：

```
-----the bookcase's information-----
书架类别是: electric
书架编号: 56
图书编号: 134-467
```

11.4　嵌套的结构体



结构体中的成员不仅可以是基本类型，也可以是结构体类型。例如，定义一个电脑结构体类型，其中的成员包括电源、机箱、显示器和内置零部件。其中，内置零部件又属于一个结构体类型，包括CPU、主板、显卡3个成员。电脑这样的结构体类型就是嵌套的结构体。

【例11.9】输出电脑零件（实例位置：资源包\TM\sl\11\09）

在本实例中，定义两个结构体类型，一个表示电脑零件，一个表示内置零件。其中，内置零件结构体是电脑零件结构中的成员。通过使用电脑零件结构类型表示电脑基本信息内容。

```
#include<stdio.h>
struct in                                          /*电脑内置零件结构体*/
{
    char in1[10];                                  /*内置零件 1*/
    char in2[10];                                  /*内置零件 2*/
    char in3[10];                                  /*内置零件 3*/
};
struct out                                         /*电脑零件结构体*/
{
    char out1[30];                                 /*零件 1*/
    char out2[30];                                 /*零件 2*/
    char out3[30];                                 /*零件 3*/
    struct in ware;                                /*内置零件*/
}computer = { "电源","机箱","显示器",{ "CPU","主板","显卡" } };   /*为结构体变量初始化*/

int main()
{
    printf("外置设备：\n");
    printf("（1）%s\n", computer.out1);            /*以下输出结构体 out 成员数据*/
    printf("（2）%s\n", computer.out2);
    printf("（3）%s\n", computer.out3);
    printf("内置设备：\n");
    printf("（1）%s\n", computer.ware.in1);        /*以下输出结构体 in 成员数据*/
    printf("（2）%s\n", computer.ware.in2);
    printf("（3）%s\n", computer.ware.in3);
    return 0;
}
```

（1）为嵌套结构体类型struct out初始化时要注意，因为内置零件仍然是一个结构体，所以要使用大括号将赋值的数据包含在内。

（2）引用成员结构体变量的成员时，computer.ware表示引用computer变量中的成员ware，computer.ware.in1表示引用computer变量中结构体变量ware的成员in1。

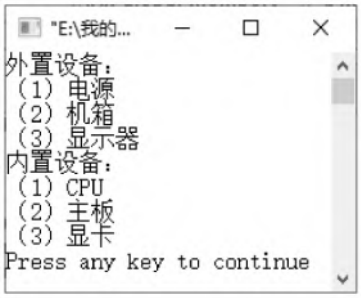


图11.12　包含结构的结构

运行程序，显示效果如图11.12所示。

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练7：桥车的发动机装置　设计一个结构体来表示一辆小轿车，该结构体中再设计一个表示发动机的结构体，最后显示输出发动机的大小以及轿车的颜色。运行结果如下：

```
车子的颜色蓝色
发动机的宽度：40
发动机的高度：60
```

训练8：个人信息暴露啦　定义两个结构体类型，一个表示日期，一个表示个人信息。其中，日期结构体是个人信息结构体的成员。最后通过个人信息结构体类型输出SuYuQun基本个人信息。运行结果如下：

```
-----个人信息如下-----
姓名：SuYuQun
年龄：25
性别：W
生日：1996-12-6
```

11.5　链表

使用数组存放数据非常方便，但由于数组的长度（即包含的元素个数）是固定的，因此当存储不同的元素数量时，就很容易出现问题。如果

向数组中添加的元素数量大于数组大小，则信息无法完全被保存。如果向数组中添加的元素数量小于数组大小，则会造成空间浪费。实际开发中，这种浪费有时是致命的。

这时就希望有另一种存储方式，其存储的元素个数是不受限制的。添加元素时，存储的个数会随之改变，这种存储方式就是链表。

11.5.1 链表概述



链表是一种常见的数据结构。链表中有一个头指针，指向一个变量，这个变量称为元素。在链表中每个元素包括数据部分和指针部分。数据部分用来存放元素所包含的数据，而指针部分用来指向下一个元素。最后一个元素的指针指向NULL，表示指向的地址为空。

如图11.13所示，head就是头指针，指向第一个元素，第一个元素中的指针又指向第二个元素，第二个元素的指针又指向第3个元素，第3个元素的指针指向为空。通过头指针可以查找到链表中的任一元素。

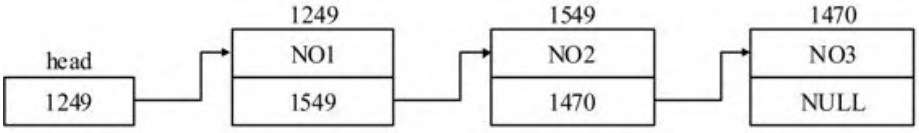


图11.13 链表

这就好比一个幼儿园中，老师拉着第一个小朋友的手，第一个小朋友又拉着第二个小朋友的手，这样下去，幼儿园中的小朋友就连成了一条线。最后一个小朋友没有拉任何人，他的手是空着的，他就好像是链表中的链尾，而老师就是头指针，通过老师可以找到这个队伍中的任何一个小朋友。

注意

链表这种数据结构需要使用指针来实现，因此链表中所有结点都会包含一个指针变量，以保存下一个结点的地址。

例如，设计一个链表表示一个班级，链表中的结点表示学生，代码如下：

```
struct Student
{
    char cName[20];                    /*姓名*/
    int iNumber;                       /*学号*/
    struct Student* pNext;             /*指向下一个结点的指针*/
};
```

可以看到，学生姓名和学号属于数据部分，pNext是指针部分，用来保存下一个结点的地址。

向链表中添加结点时，原最后一个结点的指针将保存新添加的结点地址，新结点的指针将指向空（NULL）。添加完成后，新结点将成为链表中的最后一个结点，如图11.14所示。可见，不用担心链表长度会超出范围。

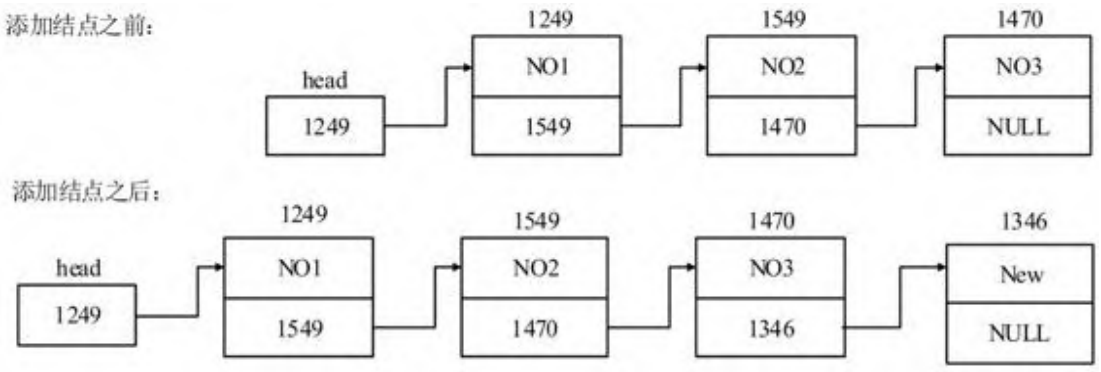


图11.14　结点添加过程

11.5.2　创建动态链表



链表并不是一开始就设定好大小的，而是根据结点多少动态创建的。动态创建结点时，首先要为其分配内存，下面先来了解一下动态分配内存空间需要用到的3个函数。

1．malloc函数

malloc函数用于在内存中动态分配一块size大小的内存空间。函数原型如下：

```
void *malloc(unsigned int size);
```

malloc函数会返回一个指针，该指针指向分配的内存空间。如果出现错误，则返回NULL。

2．calloc函数

calloc函数用于在内存中动态分配n个长度为size的连续内存空间。函数原型如下：

```
void * calloc(unsigned n, unsigned size);
```

calloc函数会返回一个指针，该指针指向动态分配的连续内存空间的首地址。如果出现错误，则返回NULL。

3．free函数

free函数用于释放指针ptr指向的内存区，使该部分内存能被其他变量使用。函数原型如下：

```
void free(void *ptr);
```

free函数无返回值。注意，ptr是最近一次调用calloc或malloc函数时返回的指针。

下面介绍如何建立动态链表。所谓动态链表，就是在程序运行过程中从无到有地建立起一个链表，即一个一个地分配结点的内存空间，然后输入结点中的数据，并建立结点间的相连关系。

仍以前面提到的学生信息为例，假设需要将某班的学生信息存放在一个链表结构中，该怎么实现呢？首先要创建一个结点结构，表示每一个学生。该结点是一个结构体，代码如下：

```
struct Student                                  /*声明学生结构体*/
{
    char cName[20];                             /*姓名*/
    int iNumber;                                /*学号*/
    struct Student* pNext;                      /*指向下一个结点的指针*/
};
```

然后，需要定义一个Create函数，用来创建链表。该函数将返回链表的头指针，代码如下：

```
int iCount;                                                    /*定义全局变量，表示链表长度*/
struct Student* Create()                                       /*创建链表函数 Create*/
{
    struct Student* pHead=NULL;                                /*初始化链表，使头指针为空*/
    struct Student* pEnd,*pNew;
    iCount=0;                                                  /*初始化链表长度*/
    pEnd=pNew=(struct Student*)malloc(sizeof(struct Student)); /*动态分配一块内存空间*/
    printf("please first enter Name ,then Number\n");
    scanf("%s",&pNew->cName);                                  /*输入学生信息*/
    scanf("%d",&pNew->iNumber);
    while(pNew->iNumber!=0)                                    /*当新增加的学号不为空时*/
    {
        iCount++;                                              /*结点数加 1*/
        if(iCount==1)                                          /*如果第一个结点*/
        {
            pNew->pNext=pHead;                                 /*使得指向为空*/
            pEnd=pNew;                                         /*跟踪新加入的结点*/
            pHead=pNew;                                        /*头指针指向首结点*/
        }
        else
        {
            pNew->pNext=NULL;                                  /*新结点的指针为空*/
            pEnd->pNext=pNew;                                  /*原来的尾结点指向新结点*/
            pEnd=pNew;                                         /*pEnd 指向新结点*/
        }
        pNew=(struct Student*)malloc(sizeof(struct Student));  /*再次分配结点内存空间*/
        scanf("%s",&pNew->cName);
        scanf("%d",&pNew->iNumber);
    }
    free(pNew);                                                /*释放没有用到的空间*/
    return pHead;
}
```

（1）在Create函数外部声明一个整型全局变量iCount，表示链表中的结点数量。

（2）Create函数中，先定义3个指针变量，pHead表示头指针，pEnd指向原来的尾结点，pNew指向新创建的结点。然后使用malloc函数分配内存，先让pEnd和pNew都指向第一个分配的内存，然后显示提示信息，输入学生的姓名和学号。

（3）使用while语句进行判断，如果学号不为0，则执行循环语句。首先是iCount++自加运算，表示增加一个结点。然后判断新加入的结点是否是第一个结点，如果是，则执行if语句块中的代码，否则执行else语句块中的代码。

（4）if语句块中，因为第一个结点加入前其中没有结点，所以新结点既为首结点也为末结点，将新结点的指针指向NULL。else语句块中，先将新结点pNew的指针指向NULL，再将原来末结点的指针指向新结点，pEnd指针指向最后一个结点。

（5）一个结点创建完之后，要为其分配内存，然后就可以输入数据。通过while语句再次判断输入的数据是否符合结点要求，当不符合要求时，调用free函数将不符合要求的结点空间释放。

这样，一个链表就通过动态分配内存空间的方式创建完成了。

11.5.3 输出链表



接下来介绍如何将链表中的数据显示输出。定义Print函数，用来输出链表中的数据，代码如下：

```
void Print(struct Student* pHead)                    /*输出结点函数 Print*/
{
    struct Student *pTemp;                           /*循环所用的临时指针*/
    int iIndex=1;                                    /*表示链表中结点的序号*/

    printf("----the List has %d members:----\n",iCount);   /*消息提示*/
    printf("\n");                                    /*换行*/
    pTemp=pHead;                                     /*临时指针得到首结点的地址*/

    while(pTemp!=NULL)
    {
        printf("the NO%d member is:\n",iIndex);
        printf("the name is: %s\n",pTemp->cName);    /*输出姓名*/
        printf("the number is: %d\n",pTemp->iNumber); /*输出学号*/
        printf("\n");                                /*输出换行*/
        pTemp=pTemp->pNext;                          /*移动临时指针到下一结点处*/
        iIndex++;                                    /*进行自加运算*/
    }
}
```

（1）Print函数参数中，pHead表示链表的头结点。函数体中，定义一个临时指针pTemp，用来进行循环操作；定义一个整型变量index，用来表示链表中的结点序号；然后用临时指针变量pTemp保存首结点的地址。

（2）使用while语句显示输出所有结点中保存的数据。每输出一个结点内容后，移动pTemp指针变量，使其指向下一个结点地址。当为最后一个结点时，其指针会指向NULL，循环结束。

【例11.10】显示学生姓名和学号（实例位置：资源包\TM\sl\11\10）

根据上面介绍的有关链表的创建与输出操作，将这些代码整合到一起，编写一个包含学生信息的链表结构，并将链表中的信息显示输出。

```c
#include<stdio.h>
#include<stdlib.h>

struct Student
{
    char cName[20];                                          /*姓名*/
    int iNumber;                                             /*学号*/
    struct Student* pNext;                                   /*指向下一个结点的指针*/
};
int iCount;                                                  /*全局变量表示链表长度*/
struct Student* Create()
{
    struct Student* pHead=NULL;                              /*初始化链表头指针为空*/
    struct Student* pEnd,*pNew;
    iCount=0;                                                /*初始化链表长度*/
    pEnd=pNew=(struct Student*)malloc(sizeof(struct Student));
    printf("请输入名字、学号:\n");
    scanf("%s",&pNew->cName);
    scanf("%d",&pNew->iNumber);
    while(pNew->iNumber!=0)
    {
        iCount++;
        if(iCount==1)
        {
            pNew->pNext=pHead;                               /*使得指向为空*/
            pEnd=pNew;                                       /*跟踪新加入的结点*/
            pHead=pNew;                                      /*头指针指向首结点*/
        }
        else
        {
            pNew->pNext=NULL;                                /*新结点的指针为空*/
            pEnd->pNext=pNew;                                /*原来的尾结点指向新结点*/
            pEnd=pNew;                                       /*pEnd 指向新结点*/
        }
        pNew=(struct Student*)malloc(sizeof(struct Student));/*再次分配结点内存空间*/
        scanf("%s",&pNew->cName);
        scanf("%d",&pNew->iNumber);
    }
    free(pNew);                                              /*释放没有用到的空间*/
    return pHead;
}

void Print(struct Student* pHead)
{
    struct Student *pTemp;                                   /*循环所用的临时指针*/
    int iIndex=1;                                            /*表示链表中结点的序号*/
    printf("----有 %d 个成员:----\n",iCount);                /*消息提示*/
    printf("\n");                                            /*换行*/
    pTemp=pHead;                                             /*指针得到首结点的地址*/
    while(pTemp!=NULL)
    {
        printf("NO%d 成员:\n",iIndex);
        printf("姓名: %s\n",pTemp->cName);                   /*输出姓名*/
        printf("学号: %d\n",pTemp->iNumber);                 /*输出学号*/
        printf("\n");                                        /*输出换行*/
        pTemp=pTemp->pNext;                                  /*移动临时指针到下一个结点*/
        iIndex++;                                            /*进行自加运算*/
    }
}

int main()
{
    struct Student* pHead;                                   /*定义头结点*/
    pHead=Create();                                          /*创建结点*/
    Print(pHead);                                            /*输出链表*/
    return 0;                                                /*程序结束*/
}
```

在main函数中，先定义一个头结点指针pHead，然后调用Create函数创建链表，并将链表的头结点返回给pHead指针变量。利用得到的头结点pHead作为Print函数的参数。

运行程序，显示效果如图11.15所示。

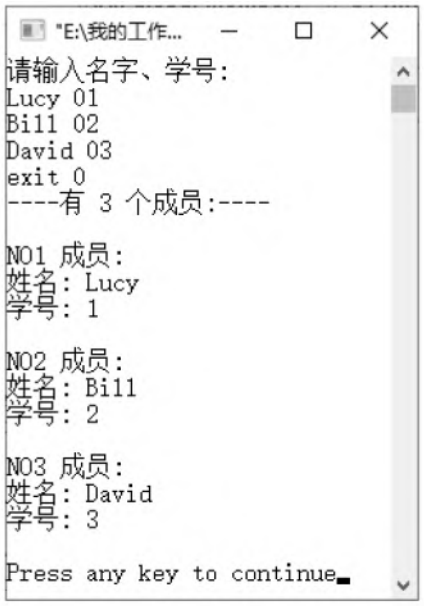


图11.15　创建链表并将数据输出

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练9：北京一日游　张敏北京一日游的观光地点为颐和园、长城和故宫。创建链表，输出这些观光地点。运行结果如下：

```
请输入张敏北京一日游观光的地点：
颐和园 1
长城 2
故宫 3
exit 0
----the List has 3 members:----

the NO1 member is:
地点是: 颐和园
第1个参观

the NO2 member is:
```

```
地点是：长城
第2个参观

the NO3 member is:
地点是：故宫
第3个参观
```

训练10：春节集卡活动　2021年春节前夕，许多App推出了集卡分红利活动。支付宝集齐五福瓜分5亿，快手集卡瓜分2亿，抖音、今日头条和火山小视频联合集卡瓜分20亿；百度集卡瓜分1亿，微博集卡瓜分1亿等。利用链表输出这几款推出集卡活动的App名字，运行结果如下：

```
请输入集卡集福 App 名字：
支付宝 1
快手 2
抖音、今日头条、火山 3
百度 4
微博 5
exit 0
----一共有 5 个 App:----

 NO1 个 App:
名字是: 支付宝
第 1 个

 NO2 个 App:
名字是: 快手
第 2 个

 NO3 个 App:
名字是: 抖音、今日头条、火山
第 3 个

 NO4 个 App:
名字是: 百度
第 4 个

 NO5 个 App:
名字是: 微博
第 5 个
```

11.6　链表的相关操作

本节将对链表的功能进行完善，使其具有插入和删除结点的功能。

11.6.1 插入结点



链表中，可以在头结点位置插入结点，也可以在中间位置插入结点，还可以像创建链表那样在最后插入结点，这3种插入操作的思路是一样的。下面主要介绍第一种插入方式，即在链表的头结点位置插入结点，其插入原理如图11.16所示。

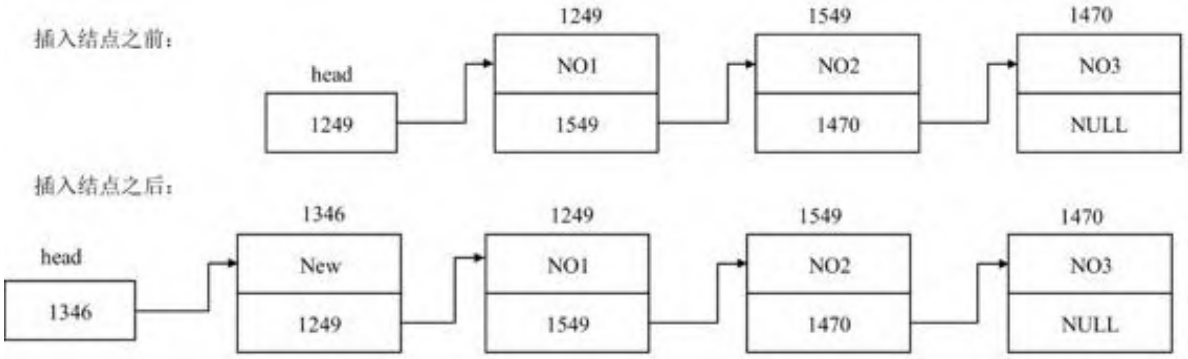


图11.16 插入结点操作

插入结点的过程就好比一群小朋友手拉手连成一条线，这时又来了一个小朋友，他要站在老师和一个小朋友的中间，那么老师就要放开原来的小朋友，拉住新加入的小朋友，这个新加入的小朋友再拉住原来的那个小朋友。

仍以前面创建的学生信息链表为例，设计一个函数Insert，用来向链表中插入结点，代码如下：

```
struct Student* Insert(struct Student* pHead)                    /*插入结点函数 Insert*/
{
    struct Student* pNew;                                        /*指向新分配的空间*/
    printf("----插入第 1 名信息----\n");                          /*提示信息*/
    pNew=(struct Student*)malloc(sizeof(struct Student));        /*分配内存空间，并返回指向该内存空间的指针

    scanf("%s",&pNew->cName);
    scanf("%d",&pNew->iNumber);

    pNew->pNext=pHead;                                           /*新结点指针指向原来的首结点*/
    pHead=pNew;                                                  /*头指针指向新结点*/
    iCount++;                                                    /*增加链表结点数量*/
    return pHead;
}
```

上述代码中，先为要插入的新结点分配内存，然后向新结点中输入数据，这样一个结点就创建完成了。接下来将这个结点插入链表中。首先将新结点的指针指向原来的首结点，保存首结点的地址。然后将头指针指向新结点，完成结点的连接操作，最后增加链表的结点数量。

修改main函数的代码，添加插入结点操作的函数Insert：

```
int main()
{
    struct Student* pHead;                                       /*定义头结点*/
    pHead=Create();                                              /*创建结点*/
    pHead=Insert(pHead);                                         /*插入结点*/
    Print(pHead);                                                /*输出链表*/
    return 0;                                                    /*程序结束*/
}
```

使用Insert函数返回新的头指针，运行程序，显示效果如图11.17所示。

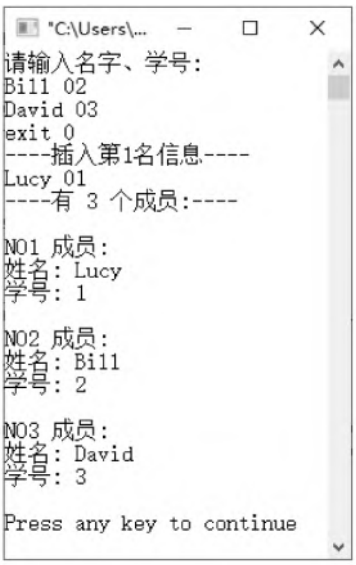


图11.17　链表插入操作

11.6.2　删除结点



如何删除链表中的结点呢？还是通过小朋友手拉手的比喻来想想一些。假设队伍中的一个小朋友想离开了，剩下的人仍需要保持手拉手，方法很简单：只需他两边的小朋友将手拉起来就可以了。

删除链表中某个结点的操作原理如图11.18所示。

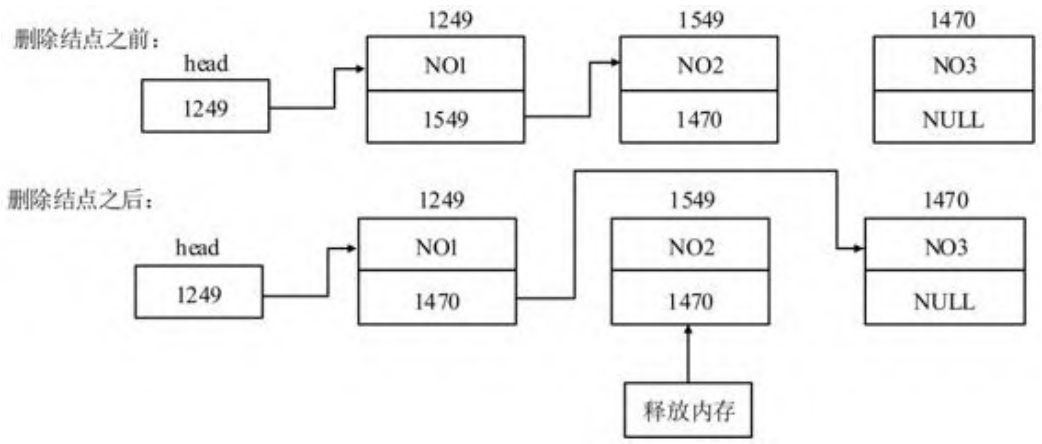


图11.18　删除结点操作

可以发现，要删除一个结点，首先要找到这个结点的位置，例如NO2结点，然后将NO1结点的指针指向NO3结点，最后将NO2结点的内存空间释放掉，这样就完成了结点的删除操作。

在学生信息链表中，编写删除结点操作的函数Delete，代码如下：

```
void Delete(struct Student* pHead,int iIndex)          /*删除结点函数 Delete*/
{
    int i;                                             /*控制循环变量*/
    struct Student* pTemp;                             /*临时指针*/
    struct Student* pPre;                              /*表示要删除结点前的结点*/
    pTemp=pHead;                                       /*得到头结点*/
    pPre=pTemp;

    printf("----删除 NO%d 名成员----\n",iIndex);
    for(i=1;i<iIndex;i++)                              /*for 循环，使 pTemp 指向要删除的结点*/
    {
        pPre=pTemp;
        pTemp=pTemp->pNext;
    }
    pPre->pNext=pTemp->pNext;                          /*连接删除结点两边的结点*/
    free(pTemp);                                       /*释放要删除结点的内存空间*/
    iCount--;                                          /*减少链表中的元素个数*/
}
```

（1）Delete函数有两个参数，pHead表示头指针，iIndex表示待删除结点在链表中的位置。定义整型变量i用来控制循环的次数，然后定义两个指针，分别表示要删除的结点和该结点前的结点。

（2）输出一行提示信息，表示要进行删除操作。之后利用for循环查找要删除的结点，使用pTemp保存要删除结点的地址，pPre保存前一个

结点的地址。找到要删除的结点后，连接该结点两边的结点，然后使用free函数将pTemp指向的内存空间释放。

接下来在main函数中添加代码执行删除操作，将链表中的第二个结点进行删除。

```
int main()
{
    struct Student* pHead;                    /*定义头结点*/
    pHead=Create();                           /*创建结点*/
    pHead=Insert(pHead);                      /*插入结点*/
    Delete(pHead,2);                          /*删除第二个结点的操作*/
    Print(pHead);                             /*输出链表*/
    return 0;                                 /*程序结束*/
}
```

运行程序，可以看到第二个结点中的数据已被删除，显示效果如图11.19所示。



图11.19　删除结点操作

有关链表的操作就讲解到这里。为了方便读者阅读程序，这里将有关链表操作的完整代码给出，希望读者能从整体上对链表有一个清晰的理解。

【例11.11】完整的链表操作（实例位置：资源包\TM\sl\11\11）

本实例中，实现学生信息链表的创建、增加、删除操作，具体代码如下：

```
#include<stdio.h>
#include<stdlib.h>

struct Student                                          /*定义学生结构体*/
{
    char cName[20];                                     /*姓名*/
    int iNumber;                                        /*学号*/
    struct Student* pNext;                              /*指向下一个结点的指针*/
};

int iCount;                                             /*定义全局变量，表示链表长度*/
struct Student* Create()                                /*创建链表函数 Create*/
{
    struct Student* pHead=NULL;                         /*初始化链表，头指针为空*/
    struct Student* pEnd,*pNew;
    iCount=0;                                           /*初始化链表长度*/
    pEnd=pNew=(struct Student*)malloc(sizeof(struct Student));
    printf("请输入姓名、学号：\n");
    scanf("%s",&pNew->cName);
    scanf("%d",&pNew->iNumber);
    while(pNew->iNumber!=0)
    {
        iCount++;
        if(iCount==1)
        {
            pNew->pNext=pHead;                          /*使得指向为空*/
            pEnd=pNew;                                  /*跟踪新加入的结点*/
            pHead=pNew;                                 /*头指针指向首结点*/
        }
        else
        {
            pNew->pNext=NULL;                           /*新结点的指针为空*/
            pEnd->pNext=pNew;                           /*原来的尾结点指向新结点*/
            pEnd=pNew;                                  /*pEnd 指向新结点*/
        }
        pNew=(struct Student*)malloc(sizeof(struct Student));   /*再次分配结点内存空间*/
        scanf("%s",&pNew->cName);
        scanf("%d",&pNew->iNumber);
    }
    free(pNew);                                         /*释放没有用到的空间*/
    return pHead;
}

void Print(struct Student* pHead)                       /*输出结点函数 Print*/
```

```
{
    struct Student *pTemp;                                  /*循环所用的临时指针*/
    int iIndex=1;                                           /*表示链表中结点的序号*/

    printf("----有 %d 名成员:----\n",iCount);               /*消息提示*/
    printf("\n");                                           /*换行*/
    pTemp=pHead;                                            /*指针得到首结点的地址*/

    while(pTemp!=NULL)
    {
        printf(" NO%d 号成员:\n",iIndex);
        printf("姓名: %s\n",pTemp->cName);                  /*输出姓名*/
        printf("学号: %d\n",pTemp->iNumber);                /*输出学号*/
        printf("\n");                                       /*输出换行*/
        pTemp=pTemp->pNext;                                 /*移动临时指针到下一个结点*/
        iIndex++;                                           /*进行自加运算*/
    }
}

struct Student* Insert(struct Student* pHead)               /*插入结点函数 Insert*/
{
    struct Student* pNew;                                   /*指向新分配的空间*/
    printf("----插入第 1 名信息----\n");                    /*提示信息*/
    pNew=(struct Student*)malloc(sizeof(struct Student)); /*分配内存空间，并返回指向该内存空间的指针*/

    scanf("%s",&pNew->cName);
    scanf("%d",&pNew->iNumber);

    pNew->pNext=pHead;                                      /*新结点指针指向原来的首结点*/
    pHead=pNew;                                             /*头指针指向新结点*/
    iCount++;                                               /*增加链表结点数量*/
    return pHead;
}

void Delete(struct Student* pHead,int iIndex)               /*删除结点函数 Delete*/
{
    int i;                                                  /*控制循环变量*/
    struct Student* pTemp;                                  /*临时指针*/
    struct Student* pPre;                                   /*表示要删除结点前的结点*/
    pTemp=pHead;                                            /*得到头结点*/
    pPre=pTemp;

    printf("----删除 NO%d 名成员----\n",iIndex);            /*提示信息*/
    for(i=1;i<iIndex;i++)                                   /*for 循环使得 pTemp 指向要删除的结点*/
    {
        pPre=pTemp;
        pTemp=pTemp->pNext;
    }
    pPre->pNext=pTemp->pNext;                               /*连接删除结点两边的结点*/
    free(pTemp);                                            /*释放要删除结点的内存空间*/
    iCount--;                                               /*减少链表中的元素个数*/
}

int main()
{
    struct Student* pHead;                                  /*定义头结点*/
    pHead=Create();                                         /*创建结点*/
    pHead=Insert(pHead);                                    /*插入结点*/
    Delete(pHead,2);                                        /*删除第二个结点*/
    Print(pHead);                                           /*输出链表*/
    return 0;                                               /*程序结束*/
}
```

说明

链表的产生弥补了数组的一些不足。在数组中插入或删除元素时，需要设计算法来移动数组元素，非常麻烦。但链表也有不足，在链表中查找元素远远不如数组有效率，所以两者的使用都要因地制宜。

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练11：你集到了哪些福　请用链表输出你集到的5个福——和谐福、爱国福、友善福、富强福以及敬业富。

训练12：输出新冠疫情重灾区　新冠肺炎疫情是2020年席卷世界的一场灾难。从中国疫情状况地图上来看，前3名重灾区省份分别是湖北省、浙江省以及广东省。利用链表输出3个重灾区。

11.7　共用体

共用体看起来很像结构体，只不过关键字由struct变成了union。共用体和结构体的区别在于：结构体定义了一个由多个数据成员组成的特殊类型，而共用体定义了一块为所有数据成员共享的内存。

11.7.1　共用体的概念



共用体也称为联合体，它使几种不同类型的变量存放到同一段内存单元中。共用体在某一时刻只能有一个值，即某个数据成员的值。由于所有成员共用同一块内存，因此共用体的大小就等于最大成员的大小。

定义共用体的类型变量的一般形式如下：

```
union 共用体名
{
    成员列表
}变量列表;
```

例如，下面的代码定义一个共用体，包括的数据成员有整型、字符型和实型：

```
union DataUnion
{
    int iInt;
    char cChar;
    float fFloat;
}variable;                                    /*定义共用体变量*/
```

其中，variable为定义的共用体变量，而union DataUnion是共用体类型。共用体也可以像结构体那样，将类型的声明和变量的定义分开：

```
union DataUnion variable;
```

可以看到，共用体和结构体定义变量的方式很相似。需要注意的是，结构体变量的大小是其所包括的所有数据成员大小的总和，其每个成员都有自己的内存单元；共用体的大小则为所包含数据成员中最大内存长度的大小。例如，上面定义的共用体变量variable的大小就与float类型的大小相等。

11.7.2 共用体变量的引用



共用体变量定义完成后，就可以引用其中的数据成员，引用形式为“共用体变量.成员名”。例如，引用前面定义的variable变量中的成员数

据，代码如下：

```
variable.iInt;
variable.cChar;
variable.fFloat;
```

注意，不能在输出语句中直接引用共用体变量，如“printf（"%d"，variable）;”。

【例 11.12】选择一种交通工具回家（实例位置：资源包\TM\sl\11\12）

公司员工下班乘车可以坐公交，也可以坐地铁。设计一个交通工具的共用体，让员工进行选择。在本实例中定义共用体变量，通过定义的显示函数，引用共用体中的数据成员。

```
#include "stdio.h"
#include <string.h>
struct bus                                  /*声明公交车结构体类型*/
{
    char name[64];
};
struct subway                               /*声明地铁结构体类型*/
{
    char name[64];
};
union Transportation                        /*声明公共交通共用体类型*/
{
    struct bus p;
    struct plane c;
};

int main()
{
    union Transportation t;                 /*定义一个共用体*/
    strcpy(t.p.name, "地铁");               /*将相应的名字复制给相应的变量*/
    strcpy(t.c.name, "公交车");
    printf("员工选择%s\n",t.p.name);        /*输出信息*/
    printf("员工选择%s\n",t.c.name);
    return 0;
}
```

在程序中改变共用体的一个成员，其他成员也会随之改变。当给某个特定的成员进行赋值时，其他成员的值也会具有一致的含义，这是因为它

们的值的每一个二进制位都被新值所覆盖。

运行程序，显示效果如图11.20所示。

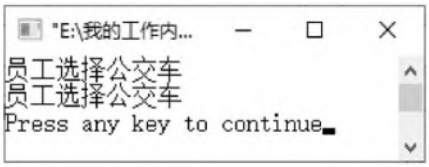


图11.20　使用共用体变量

11.7.3　共用体变量的初始化



定义共用体变量时，可以同时进行初始化操作。同样，初始化值需要放在一对大括号中。

注意，对共用体变量初始化时，只需要一个初始化值就足够了，其类型必须和共用体的第一个成员的类型相一致。如果共用体的第一个成员是一个结构体类型，则初始化值中可以包含多个用于初始化该结构体的表达式。

【例11.13】字符型和整型的亲密关系（实例位置：资源包\TM\sl\11\13）

在本实例中，在定义共用体变量的同时进行初始化操作，并将引用变量的值输出。

```
#include<stdio.h>
union DataUnion                          /*声明共用体类型*/
{
    int iInt;                            /*成员变量*/
    char cChar;
};

int main()
{
    union DataUnion Union={97};          /*定义共用体变量，并进行初始化*/
    printf("iInt: %d\n",Union.iInt);     /*输出成员变量数据*/
    printf("cChar: %c\n",Union.cChar);
    return 0;
}
```

运行程序，显示效果如图11.21所示。

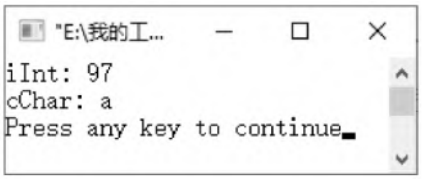


图11.21　共用体变量的初始化

说明

共用体变量可以赋值吗？其实是可以的，只是赋的值仍然是一个相同类型的共用体类型。引用共用体变量名也可以得到一个值，这个值就是共用体对象本身的值。

11.7.4　共用体类型的数据特点



在使用共用体类型时，需要注意以下特点。

- ☑ 同一内存虽然可用来存放不同类型的成员，但每次只能存放一种类型，而不能同时存放所有类型。也就是说，共用体中只有一个成员能起作用，其他成员不起作用。
- ☑ 共用体变量中，起作用的永远是最后存放的成员。存入新成员后，原有的成员将失去作用。
- ☑ 共用体变量的地址和它的各成员地址是一样的。
- ☑ 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值。

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练13：模拟美团订餐　午休吃午饭，公司员工准备在美团上订餐，在米饭、面条、水饺3家店中进行选择，最终因为当天是节日，决定吃饺子。编程模拟此场景。

训练14：罐头种类　将玻璃罐头瓶设为一个共用体，该瓶可以装黄桃、椰子和山楂。定义共用体变量，通过定义的显示函数，引用共用体中的数据成员。运行结果如下：

```
这个罐头瓶装山楂
```

11.8　枚举类型



实际问题中，有些变量会被限定在一个有限范围内，如一周有7天，一年有12个月等。如果把这些变量声明为整型、字符型或其他类型，显然是不妥的。为此，C语言提供了枚举类型。

枚举类型是一种基本数据类型，不能再被分拆。利用关键字enum可以声明枚举类型，并使用枚举类型定义变量。一个枚举变量包含一组相关的标识符，每个标识符都对应一个整数值，称为枚举常量。

例如，定义一个枚举类型变量，其中每个标识符都对应一个整数值：

```
enum Colors(Red,Green,Blue);
```

Colors就是定义的枚举类型变量，括号中第一个标识符对应着数值0，第二个对应于1，依此类推。

注意

每个标识符都必须是唯一的，而且不能采用关键字或当前作用域内其他相同的标识符名。

在定义枚举类型的变量时，可以为某个特定的标识符指定其对应的整型值，紧随其后的标识符对应的值依次加1。例如：

```
enum Colors(Red=1,Green,Blue);
```

这样的话，Red的值为1，Green为2，Blue为3。

【例11.14】选择喜欢的颜色（实例位置：资源包\TM\sl\11\14）

在本实例中，通过定义枚举类型观察其使用方式，其中每个枚举常量在声明的作用域内都可以看作一个新的数据类型。

```
#include<stdio.h>

enum Color{Red=1,Blue,Green} color;                    /*定义枚举变量，并初始化*/
int main()
{
    int icolor;                                        /*定义整型变量*/
    printf("1 代表红色，2 代表蓝色，3 代表绿色。\n");
    printf("请输入您要选择的数字：");
    scanf("%d",&icolor);                               /*输入数据*/
    switch(icolor)                                     /*判断 icolor 值*/
    {
    case Red:                                          /*枚举常量，Red 表示 1*/
        printf("选择红色\n");
        break;
    case Blue:                                         /*枚举常量，Blue 表示 2*/
        printf("选择蓝色\n");
        break;
    case Green:                                        /*枚举常量，Green 表示 3*/
        printf("选择绿色\n");
        break;
    default:
        printf("???\n");
        break;
    }
    return 0;
}
```

程序中首先定义了一个枚举变量。初始化时为第一个枚举常量赋值为1。Red赋值为1后，之后的枚举常量就会依次加1。通过switch语句判断输入的数据与这些标识符是否符合，然后执行case语句中的操作。

运行程序，显示效果如图11.22所示。

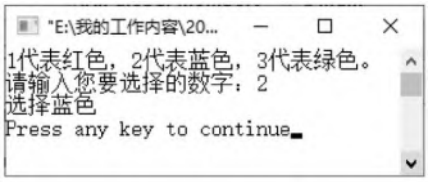


图11.22　使用枚举类型

编程训练（答案位置：资源包\TM\sl\11\编程训练\）

训练15：给一年四季打分　定义枚举类型，代表一年中的4个季节，给“季节”枚举类型分别赋值，并用整型格式输出4个季节的值。

训练16：果盘里的水果　将果盘定义成一个枚举类型，输出枚举变量的数值，并根据用户的输入，判断用户想吃的水果类型。运行结果如下：

```
0代表西瓜,1代表芒果,2代表葡萄,3代表橘子,4代表苹果
请输入你想吃的水果代表的数字：
1
你想吃芒果
```

11.9　实践与练习

（答案位置：资源包\TM\sl\11\实践与练习\）

综合练习1：长春三日游　李明听说长春很美，就准备来个长春三日游。利用链表输出李明长春三日游逛的景点，输出效果如下：

```
请输入李明长春三日游观光的地点：
南湖公园 1
伪满皇宫博物馆 2
亚泰海洋馆 3
exit 0
----有 3 个地点----

第 NO1 地点是：
 南湖公园
第 1 个参观

第 NO2 地点是：
 伪满皇宫博物馆
第 2 个参观

第 NO3 地点是：
 亚泰海洋馆
第 3 个参观
```

综合练习2：无人商店产品基本信息　**定义一个结构体数组，输出无人商店产品基本信息。输出效果如下：**

```
第1种产品：
名字是：康师傅方便面，单价是：2.50元

第2种产品：
名字是：农夫山泉，单价是：2.00元

第3种产品：
名字是：玉米肠，单价是：3.00元

第4种产品：
名字是：可比克薯片，单价是：3.00元

第5种产品：
名字是：蒙牛核桃奶，单价是：2.50元
```

综合练习3：身份证信息揭秘　**使用结构体指针变量输出身份证背后的数据信息，输出效果如下：**

```
第1 个人：
姓名：王囧，出生日期：19991212
性别：男，地址：吉林省长春市

第2 个人：
```

```
姓名：李果，出生日期：19940505
性别：女，地址：河北省北京市

第3 个人：
姓名：张多，出生日期：20001111
性别：男，地址：山东省济南市

第4 个人：
姓名：赵紫轩，出生日期：19900306
性别：女，地址：辽宁省大连市

第5 个人：
姓名：钱小欠，出生日期：19920506
性别：男，地址：江苏省苏州市
```

综合练习4：输出手机基本信息　**利用结构体输出手机的基本信息，包括手机名称、官方报价、主屏尺寸、CPU型号以及电池容量等信息，输出效果如下：**

```
产品名称：VIVO NEX 双面屏
官方报价：4998元
主屏尺寸：6.39寸
CPU型号：高通 骁龙845
电池容量：3500mAh
```

综合练习5：打印某月电子商品销售明细　**利用结构体数组打印出某月的商品销售情况，包括商品编号、商品名称以及销售数量，输出效果如下：**

```
5月份的商品销售明细如下：
商品编号：T0001  商品名称：笔记本电脑  销售数量：2台

商品编号：T0002  商品名称：华为荣耀6X  销售数量：10台

商品编号：T0003  商品名称：iPad  销售数量：2台

商品编号：T0004  商品名称：华为荣耀V9  销售数量：20台

商品编号：T0005  商品名称：MacBock  销售数量：5台
```

综合练习6：选票系统　设计一个候选人的选票程序。假设有3个候选人，每一次输入要选择的候选人姓名，最后输出每个人的得票结果。输出效果如下：

```
请输入唱票过程：
张三
张三
张三
刘六
苏武
刘六
刘六
苏武
苏武
张三
张三：4票
刘六：3票
苏武：3票
```

综合练习7：输出面包信息　市面上面包种类有很多，也分很多品牌。用结构体变量输出面包的基本信息（品牌名、零售价和重量），同时用包含的另一个结构体输出面包的口味（原味、紫薯味和水果味），输出效果如下：

```
基本信息：
品牌：桃李
零售价：5元
重量：350g
口味种类：
口味1：原味
口味2：紫薯味
口味3：水果味
```

综合练习8：2021年火爆游戏排名　利用链表输出2021年火爆游戏排行前3名，输出效果如下：

```
请输入 2021 年火爆游戏前 3 名：
龙武 2 1
英雄联盟 2
地下城与勇士 3
exit 0
----有 3 个成员----

第 NO1 成员是：
  龙武 2
第 1 名

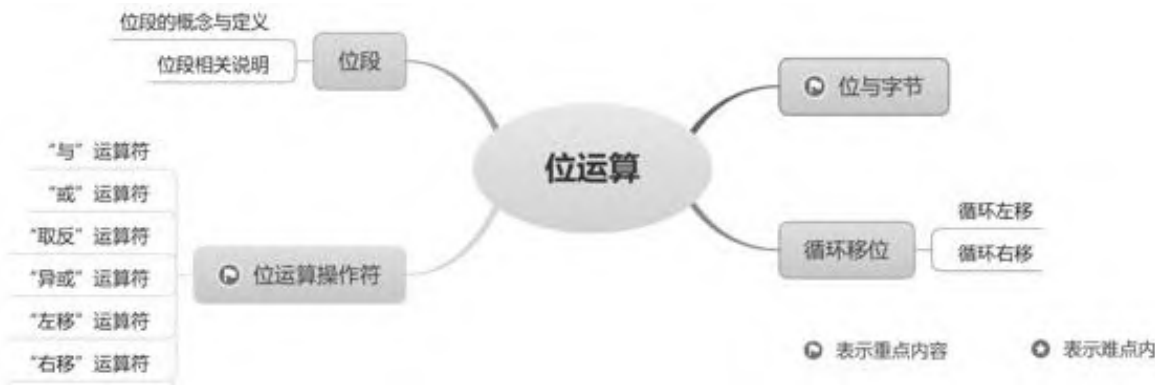
第 NO2 成员是：
  英雄联盟
第 2 名

第 NO3 成员是：
  地下城与勇士
第 3 名
```

第12章
位运算

C语言可用来代替汇编语言，完成大部分的系统开发及底层驱动开发工作，其中一个重要原因就是C语言完全支持按位运算。位运算是C语言的一个特点，这个特点使C语言的应用更加广泛。

本章的知识架构及重难点如下：



12.1 位与字节



数据在内存中是以二进制的形式存放的。位是计算机存储数据的最小单位，一个二进制位可以表示0、1两种状态，多个二进制位组合起来便可表示多种信息。

一个字节通常由8位二进制数组成（有的计算机系统由16位组成），因为本书中所使用的运行环境是Visual C++ 6.0，所以定义一个基本整型数据时，它在内存中占4个字节，也就是32位；如果定义一个字符型，

则在内存中占一个字节，也就是8位。不同的数据类型占用的字节数不同，因此占用的二进制位数也不同。

12.2 位运算操作符

C语言既具有高级语言的特点，又具有低级语言的功能，C语言完全支持按位运算，而且能像汇编语言一样编写系统程序。前面讲过的都是以字节为基本单位进行运算的，本节将介绍如何在位一级进行运算。按位运算也就是对字节或字中的实际位进行检测、设置或移位。如表12.1所示为C语言提供的6类位运算符。

表12.1 位运算符

位运算符	含义	位运算符	含义
&	按位与	^	按位异或
|	按位或	<<	左移
~	取反	>>	右移

12.2.1 按位与运算符



按位与运算符“&”是双目运算符，功能是使参与运算的两个数对应的二进位相与，即对应的两个二进位均为1时，结果为1，否则为0，如表12.2所示。

表12.2 按位与运算符

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

例如，89&38的计算过程如下：

```
  0000000001011001    十进制数 89
& 0000000000100110    十进制数 38
------------------
  0000000000000000    十进制数 0
```

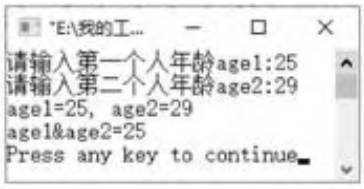
可以发现，按位与运算的一个用途就是清零，即要想原数中为1的位置为0，只需使其与对应位置为0的数相与即可。另一个用途是取一个数中的某些特定位。例如，要取22的后5位，使其与后5位均是1的数进行相与即可。同样，要取后4位，就与后4位都是1的数相与即可。

【例12.1】年龄值进行与运算（实例位置：资源包\TM\sl\12\01）

本实例中，定义两个变量，分别代表两个人的年龄，将这两个变量进行相与运算，具体代码如下：

```
#include<stdio.h>
void main()
{
    unsigned result;                          /*定义无符号变量*/
    int age1, age2;                           /*定义变量*/
    printf("请输入第一个人年龄 age1:");          /*提示输入年龄 1*/
    scanf("%d",&age1);                        /*输入年龄 1*/
    printf("请输入第二个人年龄 age2:");          /*提示输入年龄 2*/
    scanf("%d",&age2);                        /*输入年龄 2*/
    printf("age1=%d, age2=%d", age1, age2);   /*显示年龄*/
    result = age1&age2;                       /*计算相与运算的结果*/
    printf("\nage1&age2=%u\n", result);       /*输出计算结果*/
}
```

程序运行结果及两个年龄的相与运算过程如图12.1所示。




```
  0000000000011001    十进制数 25
& 0000000000011011    十进制数 29
------------------
  0000000000011001    十进制数 25
```

图12.1　年龄相与运算

误区警示

初学者很容易将“&”与“&&”运算符混淆，下面总结一下它们的区别。

“&&”是逻辑与运算符，相当于生活中的“与”，如我与你。其含义为：两个条件同时成立，逻辑与运算的结果为真，否则为假。

“&”是位运算符，且为双目运算符，当两个位都为1时，结果才为1，否则为0。

12.2.2 按位或运算符



按位或运算符“|”是双目运算符，功能是使参与运算的两个数对应的二进位相或，即只要对应的两个二进位有一个为1，结果就为1，如表12.3所示。

表12.3 “或”运算符

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

例如，17|31的计算过程如下：

```
  0000000000010001    十进制数 17
| 0000000000011111    十进制数 31
-------------------
  0000000000011111    十进制数 31
```

可以发现，十进制数17对应的二进制数的后5位是10001，十进制数31对应的二进制数的后5位是11111，将这两个数执行按位或运算之后，得到的结果是31，也就是17对应二进制数的后5位中0变成了1。因此，可以总结出这样一个规律：要想使一个数的后6位全为1，只需和63进行按位或运算即可；同理，要想使一个数的后5位全为1，只需和31进行按位运算即可。

技巧

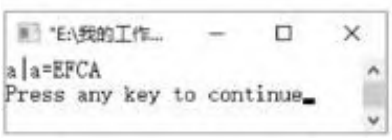
> 同理，要想将某几位变为1，只需与这几位是1的数执行相或操作便可。

【例12.2】0xEFCA与本身进行按位或运算（**实例位置：资源包\TM\sl\12\02**）

本实例中，将十六进制数0xEFCA与本身进行按位或运算，代码如下：

```
#include<stdio.h>
int main()
{
    int a=0xEFCA,result;                    /*定义变量*/
    result = a|a;                           /*计算或运算的结果*/
    printf("a|a=%X\n", result);             /*输出结果*/
    return 0;                               /*程序结束*/
}
```

程序运行结果和按位或计算过程如图12.2所示。为了方便观察，这里只给出每个数据的后16位。




```
  1110111111001010    十六进制数 0xEFCA
| 1110111111001010    十六进制数 0xEFCA
------------------
  1110111111001010    十六进制数 0xEFCA
```

图12.2　按位或运算

误区警示

同样，初学者也很容易将“|”与“||”运算符混淆，下面总结一下它们的区别。

“||”是逻辑运算符，相当于生活中的“或者”，如我或者你。其含义为：两个条件中有一个成立，逻辑或运算的结果就是“真”。

“|”是位运算符，且为双目运算符，当两个位都为0时，结果才为0，否则为1。

12.2.3 按位取反运算符



取反运算符“~”为单目运算符，具有右结合性。其功能是对参与运算的数的各二进位按位求反，即将0变成1，1变成0。例如，~86是对86进行按位求反，计算过程如下：

```
    00000000000000000000000001010011
~                   ↓
    11111111111111111111111110101100
```

【例12.3】将输入的数取反运算（实例位置：资源包\TM\sl\12\03）

本实例中，输入一个数并赋给变量a，计算~a的值，最后以八进制形式输出。代码如下：

```
#include<stdio.h>
main()
{
    unsigned result;                    /*定义无符号变量*/
    int a;
    printf("please input a:");
    scanf("%d",&a);
    printf("a=%d", a);
    result = ~a;                        /*求 a 的反*/
    printf("\n~a=%o\n", result);
}
```

程序运行结果和取反计算过程如图12.3所示。注意，这里最后是以八进制的形式输出的。

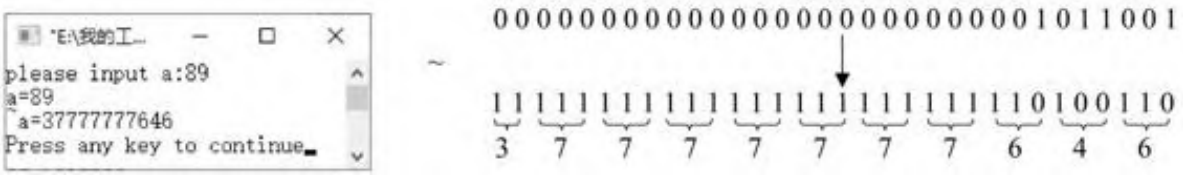


图12.3　按位取反运算

12.2.4　按位异或运算符



按位异或运算符“^”是双目运算符。其功能是对参与运算的两个数对应的二进位相异或，即当对应的两个二进位数相异时结果为1，否则结果为0，如表12.4所示。

表12.4　“异或”运算符

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

例如，107^127的计算过程为：

```
  0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1
^ 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
-----------------------------------
  0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
```

从上面算式可以看出，异或操作的一个主要用途就是能使特定的位翻转。例如，要想将107的后7位翻转，只需与一个后7位都是1的数进行异或操作即可。

异或操作的另一个主要用途，就是在不使用临时变量的情况下实现两个变量值的互换。

例如，x=9，y=4，将x和y的值互换可用如下方法实现：

```
x=x^y;
y=y^x;
x=x^y;
```

其具体运算过程如下：

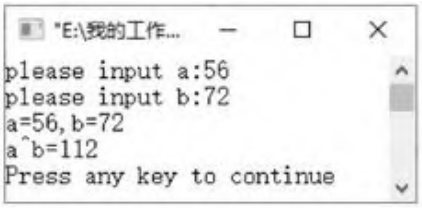
```
  0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 (x)
^ 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 (y)
-------------------------------------
  0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 (x)
^ 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 (y)
-------------------------------------
  0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 (y)
^ 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 (x)
-------------------------------------
  0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 (x)
```

【例12.4】求两个数异或值（**实例位置：资源包\TM\sl\12\04**）

本实例中，要求输入两个数，分别赋予变量a和b，计算a^b的值。代码如下：

```
#include<stdio.h>
main()
{
    unsigned result;                    /*定义无符号数*/
    int a, b;
    printf("please input a:");
    scanf("%d",&a);
    printf("please input b:");
    scanf("%d",&b);
    printf("a=%d,b=%d", a, b);
    result = a^b;                       /*求 a 与 b 按位异或的结果*/
    printf("\na^b=%u\n", result);
}
```

程序运行结果和计算过程如图12.4所示。




```
  0000000000111000
^ 0000000001001000
------------------
  0000000001110000
```

图12.4　按位异或运算

技巧

> 按位异或运算经常被用到一些简单的加密算法中。

12.2.5　左移运算符



左移运算符“<<”是双目运算符，其功能是把“<<”左边的运算数的各二进位全部左移若干位，由“<<”右边的数指定移动的位数，高位丢弃，低位补0。

例如，假设a=39，那么a在内存中的存储情况如图12.5所示。

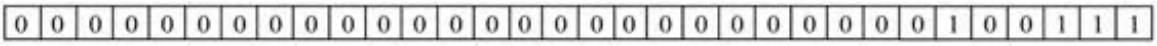


图12.5　39在内存中的存储情况

a<<2表示把a的各二进位向左移动两位，此时内存中的存储情况如图12.6所示。可见，a左移两位后由原来的39变成了156。

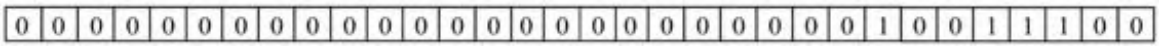


图12.6　39左移两位后

说明

仔细观察可发现，将a左移一位相当于a乘以2，将a左移两位相当于a乘以4，但这种简捷计算只限于移出位不包含1的情况。若是将十进制数64左移两位（假设64以一个字节即8位存储），则移位后的结果将为0（01000000→00000000），这是因为64在左移两位时将1移出了。

【例12.5】将15进行左移（**实例位置：资源包\TM\sl\12\05**）

本实例中，将15先左移两位，将其左移后的结果输出，在这个结果的基础上再左移3位，并将结果输出，代码如下：

```
#include<stdio.h>
main()
{
    int x=15;
    x=x<<2;                          /*x 左移 2 位*/
    printf("左移 2 位的结果:%d\n",x);
    x=x<<3;                          /*x 左移 3 位*/
    printf("再左移 3 位的结果:%d\n",x);
}
```

程序运行结果如图12.7所示。

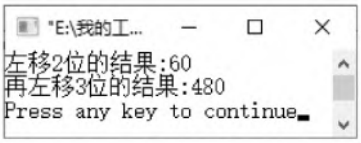


图12.7　左移运算

来分析下例12.5的移位过程。首先，15在内存中的存储情况如图12.8所示。

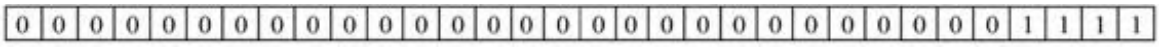


图12.8　15在内存中的存储情况

15左移两位后变为60，其存储情况如图12.9所示。

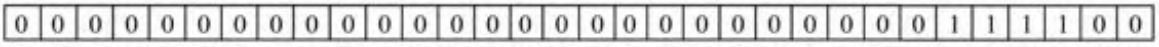


图12.9　15左移两位后变为60

60左移3位变成480，其存储情况如图12.10所示。

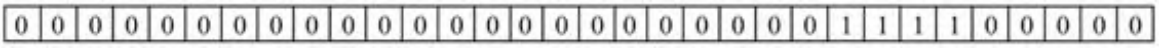


图12.10　60左移3位后变为480

12.2.6　右移运算符



右移运算符“>>”是双目运算符，其功能是把“>>”左边的运算数的各二进位全部右移若干位，“>>”右边的数指定移动的位数。

例如，a>>2即把a的各二进位向右移动两位，假设a=00000110，右移两位后为00000001，a由原来的6变成了1。

说明

对于有符号数，右移时需要注意符号位的问题。当为正数时，最高位一般补0；当为负数时，最高位是补0还是补1，取决于编译系统的规定。移入0的称为“逻辑右移”，移入1的称为“算术右移”。

【例12.6】将30和-30右移（**实例位置：资源包\TM\sl\12\06**）

本实例中，将30和-30分别右移3位，将所得结果输出，在所得结果的基础上再分别右移两位，并将结果输出。

```
#include<stdio.h>
int main()
{
    int x=30,y=-30;
    x=x>>3;                                    /*x 右移 3 位*/
    y=y>>3;                                    /*y 右移 3 位*/
    printf("x 右移 3 位，y 右移 3 位的结果:%d,%d\n",x,y);
    x=x>>2;                                    /*x 右移 2 位*/
    y=y>>2;                                    /*y 右移 2 位*/
    printf("x 再右移两位，y 再右移两位:%d,%d\n",x,y);
    return 0;
}
```

程序运行结果如图12.11所示。

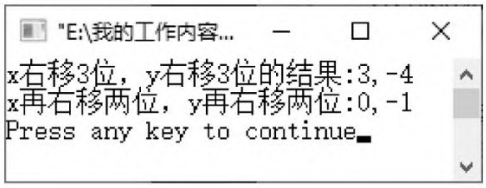


图12.11　右移运算

一起分析下例12.6的移位过程。首先，30在内存中的存储情况如图12.12所示。

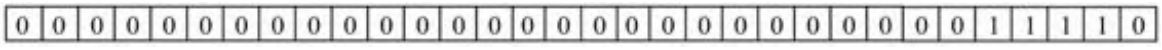


图12.12　30在内存中的存储情况

-30在内存中的存储情况如图12.13所示。



图12.13　-30在内存中的存储情况

30右移3位变成3，其存储情况如图12.14所示。

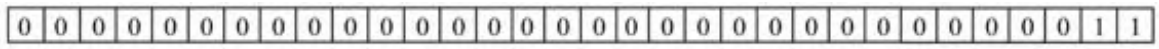


图12.14　30右移3位变为3

-30右移3位变成-4，其存储情况如图12.15所示。



图12.15　-30右移3位变为-4

3右移两位变成0，而-4右移两位则变成-1，如图12.16所示。



图12.16　-4右移两位变为-1

从上面的过程中可以发现，在Visual C++ 6.0中进行的负数右移，实质上就是算术右移。

编程训练（答案位置：资源包\TM\sl\12\编程训练\）

训练1：计算8&9　定义两个变量a和b，并分别赋值为8和9，利用按位与运算计算8&9的值。

训练2：将9取反　在控制台上输入9，将9取反。

12.3　循环移位

循环移位就是将移出的低位放到该数的高位，如图12.17所示；或者将移出的高位放到该数的低位，如图12.18所示。

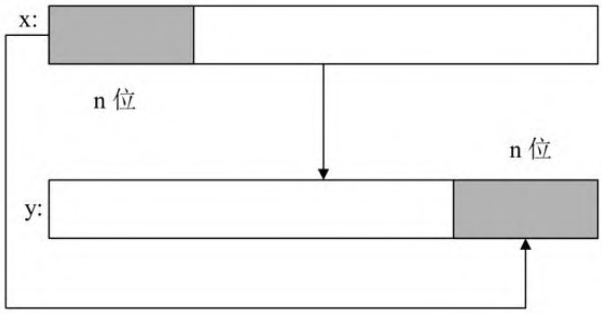


图12.17　循环左移的过程

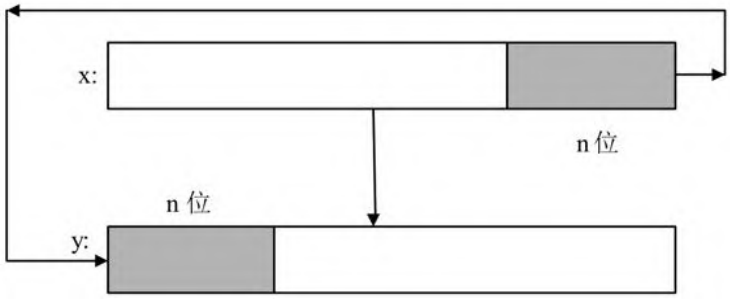


图12.18　循环右移的过程

首先介绍循环左移过程，其实际运算过程分为3步：首先将x的左端n位先放到z的低n位中，实现语句如下：

```
z=x>>(32-n);
```

然后将x左移n位，其右面低n位补0，实现语句如下：

```
y=x<<n;
```

最后将y、z进行按位或运算，实现语句如下：

```
y=y|z;
```

【例12.7】循环左移（实例位置：资源包\TM\sl\12\07）

编程实现循环左移，要求从键盘中输入一个八进制数，然后输入要移位的位数，最后将移位的结果显示在屏幕上。具体代码如下：

```
#include <stdio.h>
left(unsigned value, int n)                    /*自定义左移函数*/
{
    unsigned z;
    z = (value >> (32-n)) | (value << n);      /*循环左移的实现过程*/
    return z;
}
int main()
{
    unsigned a;
    int n;
    printf("请输入一个数:\n");
    scanf("%o", &a);                           /*输入一个八进制数*/
    printf("请输入要移位的位数（>0）:\n");
    scanf("%d", &n);                           /*输入要移位的位数*/
    printf("循环左移的结果:%o\n", left(a, n));  /*将左移后的结果输出*/
    return 0;
}
```

程序运行结果如图12.19所示。

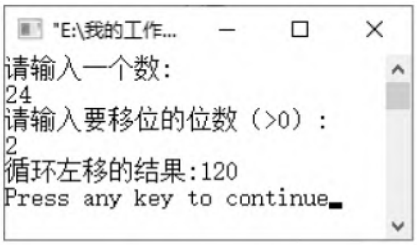


图12.19　循环左移

循环右移的实际运算过程也分为3步：首先将x的右端n位先放到z的高n位中，实现语句如下：

```
z=x<<(32-n);
```

然后将x右移n位，其左端高n位补0，实现语句如下：

```
y=x>>n;
```

最后将y与z进行按位或运算，实现语句如下：

```
y=y|z;
```

【例12.8】循环右移（实例位置：资源包\TM\sl\12\08）

编程实现循环右移，要求从键盘中输入一个八进制数，然后输入要移位的位数，最后将移位的结果显示在屏幕上。代码如下：

```
#include <stdio.h>
right(unsigned value, int n)                    /*自定义右移函数*/
{
    unsigned z;
    z = (value << (32-n)) | (value >> n);       /*循环右移的实现过程*/
    return z;
}
void main()
{
    unsigned a;
    int n;
    printf("请输入一个数:\n");
    scanf("%o", &a);                            /*输入一个八进制数*/
    printf("输入要移位的位数（>0）:\n");
    scanf("%d", &n);                            /*输入要移位的位数*/
    printf("右移后的结果是:%o\n", right(a, n));  /*将右移后的结果输出*/
}
```

程序运行结果如图12.20所示。

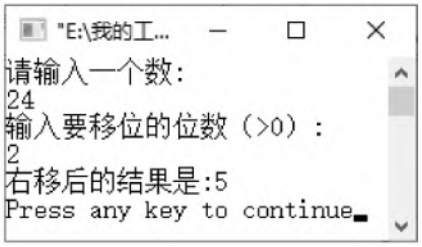


图12.20　循环右移

编程训练（答案位置：资源包\TM\sl\12\编程训练\）

训练3：将520循环左移5位　在控制台上输出520，并输出将其循环左移5位后的结果。

训练4：将1314循环右移13位　在控制台上输出1314，并输出将其循环右移13位后的结果。

12.4　位段

12.4.1　位段的概念与定义



位段是一种特殊的结构体类型，其所有成员的长度均以二进制位为单位进行定义，结构体中的成员被称为位段。位段的一般定义形式如下：

```
struct 结构体名
{
    类型  变量名 1:长度;
    类型  变量名 2:长度;
    …
    类型  变量名 n:长度;
}
```

一个位段必须是int、unsigned或signed中的一种。例如，CPU的状态寄存器按位段类型定义如下：

```
struct status                          /*状态寄存器位段*/
{
    unsigned sign:1;                   /*符号标志*/
    unsigned zero:1;                   /*零标志*/
    unsigned carry:1;                  /*进位标志*/
    unsigned parity:1;                 /*奇偶溢出标志*/
    unsigned half_carry:1;             /*半进位标志*/
    unsigned negative:1;               /*减标志*/
} flags;
```

显然，对CPU的状态寄存器而言，使用位段类型仅需1个字节即可。又如：

```
struct packed_data
{
    unsigned a:2;
    unsigned b:1;
    unsigned c:1;
    unsigned d:2;
}data;
```

可以发现，这里a、b、c、d分别占2位、1位、1位、2位，如图12.21所示。

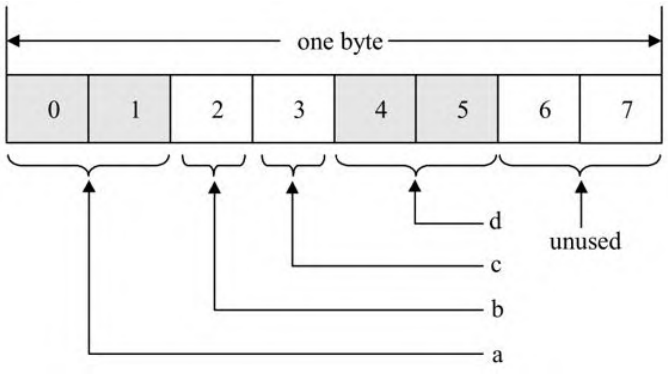


图12.21　占位情况

12.4.2　位段相关说明



前面介绍了什么是位段，这里针对位段的应用，有以下几点说明。

☑ 位段类型和位段变量的定义，以及位段成员的引用，均与结构体类型和结构体变量相同。

☑ 在下面定义的位段结构中，各位段都只占用一个二进制位：

```
struct attribute
{
    unsigned font:1;
    unsigned color:1;
    unsigned size:1;
    unsigned dir:1;
};
```

如果某位段需要表示多于两种状态，也可将该位段设置为占用多个二进制位。例如，字体大小有4种状态，则可将上面的位段结构改写成如下形式：

```
struct attribute
{
    unsigned font:1;
    unsigned color:1;
    unsigned size:2;
    unsigned dir:1;
};
```

☑ 某一位段要从另一个字节开始存放，可写成如下形式：

```
struct status
{
    unsigned a:1;
    unsigned b:1;
    unsigned c:1;
    unsigned :0;
    unsigned d:1;
    unsigned e:1;
    unsigned f:1
}flags;
```

原本a、b、c、d、e、f这6个位段是连续存储在一个字节中的。由于加入了一个长度为0的无名位段，因此其后的3个位段从下一个字节开始存储，一共占用两个字节。

☑ 可以使各位段占满一个字节，也可以不占满一个字节。例如：

```
struct packed_data
{
    unsigned a:2;
    unsigned b:2;
    unsigned c:1;
    int i;
}data;
```

存储形式如图12.22所示。

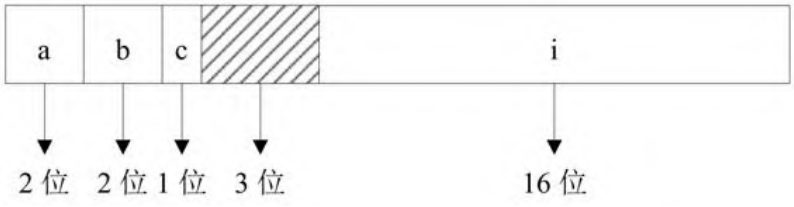


图12.22　不占满一个字节的情况

- ☑ 一个位段必须存储在一个存储单元（通常为1字节）中，不能跨两个存储单元。如果本单元不够容纳某位段，则从下一个单元开始存储该位段。
- ☑ 可以用“%d”“%x”“%u”“%o”等格式字符，以整数形式输出位段。
- ☑ 在数值表达式中引用位段时，系统自动将位段转换为整型数。

12.5　实践与练习

（答案位置：资源包\TM\sl\12\实践与练习\）

综合练习1：流水灯设计　某电子专业同学做毕业设计，要求流水灯的4个灯循环点亮。利用位运算编写程序，输出灯亮时的值（1表示亮，0表示灭）。运行结果如下：

```
请输入流水灯初始化状况（十六进制）：
3df
请输入流水灯要移位的位数（>0）：
2
--------------------------------
    流水灯移位结果 f7c
--------------------------------
```

综合练习2：密码二次加密　用户创建完新账户后，服务器为保护用户隐私，使用异或运算对用户密码进行二次加密，计算公式为“加密数据

＝原始密码^加密算子”。已知加密算子为整数79，计算用户密码459137经过加密后的值是多少。运行结果如下：

```
请输入原始密码和加密算子：
459137 79
☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆
☆    经过加密后的值是:459214      ☆
☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆
```

综合练习3：将两人体重进行相与运算　定义两个变量，分别代表两个人的体重（单位kg），将这两个变量进行相与运算。运行结果如下：

```
请输入第一个人体重:49
请输入第二个人体重:62
-*-*-*-*-*-*-*-*-*-*-*-
  weight1=49,weight2=62
-*-*-*-*-*-*-*-*-*-*-*-
@*@*@*@*@*@*@*@*@*@*@*@*
   weight1&weight2=48
@*@*@*@*@*@*@*@*@*@*@*@*
```

综合练习4：输出A的ASCII码值并取反　输入大写字母A，输出它的ASCII值及ASCII值取反的值。运行结果如下：

```
请输入字符 cChar:
A
cChar 的 ASCII 值为：65
cChar 的 ASCII 值取反为：-66
```

综合练习5：计算1028 % 8　使用位移运算和算数运算符，计算1028 % 8的结果。运行结果如下：

```
----------------------
  计算的结果等于 4
----------------------
```

综合练习6：计算a>>64、a>>65和a>>1　声明int型变量a，输入a的值，在控制台中输出a>>64、a>>65和a>>1的结果。运行结果如下：

```
请输入x的值：  73
x>>64的值是:73
```

```
x>>65的值是:36
x>>1的值是:18
```

综合练习7：0xEFCA与0按位或运算　将16进制数字0xEFCA与0进行按位或运算，输出结果，并观察结果与例12.2结果，可得出什么结论？运行结果如下：

```
→*→*→*→*→*→*→*→*→*→*
        a|0=EFCA
→*→*→*→*→*→*→*→*→*→*
```

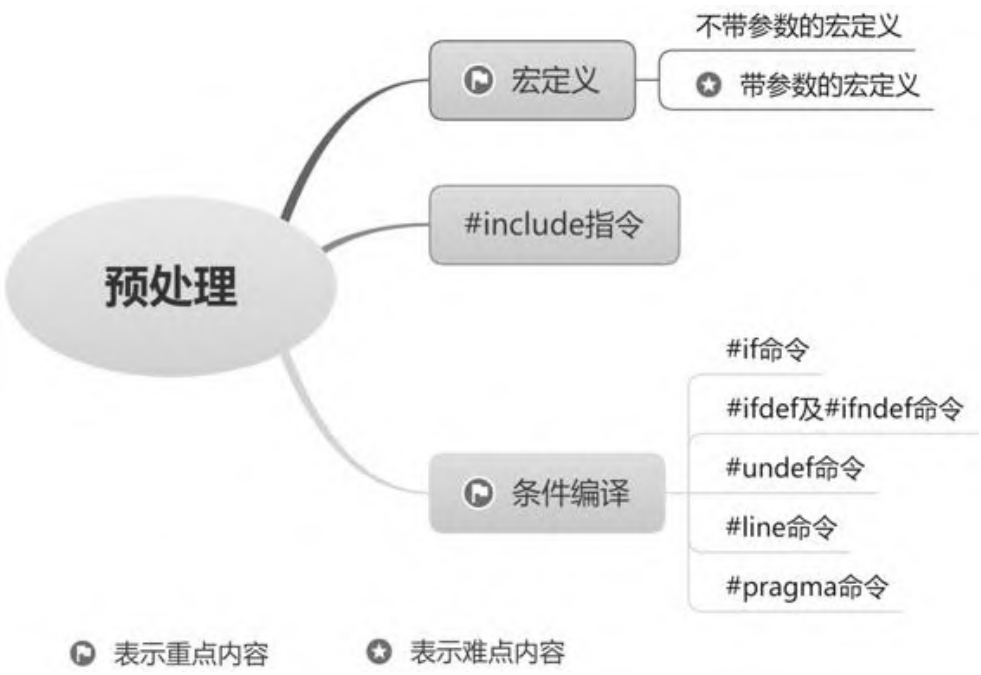
综合练习8：将50和–50右移　将50和-50分别右移3位，输出结果。在所得结果基础上再分别右移2位，继续输出结果。运行结果如下：

```
&&&&&&&&&&&&&&&&&&&&&&&&&
&     50、-50 右移 3 位的结果分别为:6,-7      &
&   再此基础上又右移 2 位的结果分别为:1,-2   &
&&&&&&&&&&&&&&&&&&&&&&&&&
```

第13章
预处理

预处理是C语言特有的功能，包括宏定义、条件编译等。使用预处理功能可使程序的修改、阅读、移植和调试更加方便，也更利于实现模块化程序设计。

本章的知识架构及重难点如下：



13.1 宏定义

宏定义是预处理命令的一种，它提供了一种可以替换源代码中字符串的机制。简单来说，宏定义指令#define用来定义一个标识符和一个字符串，以这个标识符来代表这个字符串，在程序中每次遇到该标识符时就用所定义的字符串替换它。宏定义的作用相当于给指定的字符串起了一个别名。

说明

C语言并没有规定#define必须写在函数外面，只是规定这条命令必须单独占一个完整的逻辑行。其作用范围是#define出现的位置到所在源文件结束，或到相应的#undef预处理指令处。如果需要#define预处理命令仅在某个函数内有效，完全可以把它写在函数内。

13.1.1 不带参数的宏定义



首先来讲解不带参数的宏定义，其一般形式如下：

```
#define  宏名  字符串
```

- ☑ “#”表示这是一条预处理命令。
- ☑ 宏名是一个标识符，必须符合C语言中对标识符的限定。宏名要简单且意义明确，一般习惯用大写字母表示，以便与变量名相区别。
- ☑ 字符串可以是常数、表达式、格式字符串等。

例如，下面宏定义的作用是在程序中用PI替代3.14159。编译预处理时，每当在源程序中遇到PI，就自动用3.14159代替。注意，宏定义不是C语句，不需要在行末加分号。

```
#define PI 3.14159
```

使用#define进行宏定义的好处是：当需要改变一个常量时，只需改变#define命令行，整个程序的常量都会改变，大大提高了程序的灵活性。

宏名定义后，即可成为其他宏名定义中的一部分。例如，下面代码定义了正方形的边长SIDE、周长PERIMETER及面积AREA的值。

```
#define  SIDE  5
#define  PERIMETER  4*SIDE
#define  AREA  SIDE*SIDE
```

前面强调过，宏替换是以字符串代替标识符。因此，可定义一个标准的邀请语：

```
#define  STANDARD  "You are welcome to join us."
printf(STANDARD);
```

编译程序时，遇到标识符STANDARD，就用"You are welcome to join us."替换。

关于不带参数的宏定义，有以下几点需要注意。

☑ 如果在字符串中出现宏名，则不进行替换。例如：

```
#include<stdio.h>
#define TEST "this is an example"
main()
{
    char exp[30]="This TEST is not that TEST";          /*定义字符数组并赋初值*/
    printf("%s\n",exp);
}
```

上述代码的输出结果如图13.1所示。可见，字符串中的两处TEST并没有用"this is an example"替换。

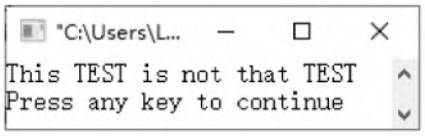


图13.1　字符串中含有宏名

☑ 如果字符串多于一行，可在行末用反斜杠“\”进行续行。

☑ #define命令出现函数外时，宏名的有效范围为定义命令之后到此源文件结束。

注意

通常会将所有#define放到文件开始处或独立的文件中，而不是将它们分散到整个程序中。

☑ 可以用#undef命令终止宏定义的作用域，例如：

```
#include<stdio.h>
#define TEST "this is an example"
main()
{
    printf(TEST);
    #undef TEST
}
```

☑ 宏定义是一种预处理命令，不同于变量定义，它只做字符替换，而不分配内存空间。

13.1.2 带参数的宏定义



带参数的宏定义，不仅要进行字符串替换，还要进行参数替换。其一般形式如下：

```
#define 宏名(参数表)字符串
```

【例13.1】两个数混合运算（实例位置：资源包\TM\sl\13\01）

本实例中，要求对两个数实现乘法和加法混合运算，代码如下：

```
#include<stdio.h>
#define MIX(a,b) ((a)*(b)+(b))                    /*宏定义，求两个数的混合运算*/
main()
{
    int x=5,y=9;
    printf("x,y:\n");
    printf("%d,%d\n",x,y);
    printf("the min number is:%d\n",MIX(x,y));    /*宏定义调用*/
}
```

程序运行结果如图13.2所示。可见，编译程序时，printf语句将被替换为如下形式：

```
printf("the min number is: %d",((a)*(b)+(b)));
```

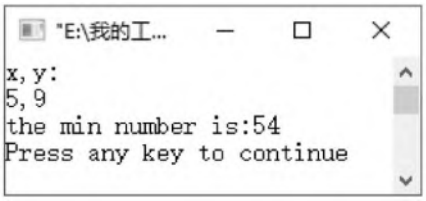


图13.2　混合运算

用宏替换代替实在的函数，好处是可以提升代码速度（不再存在函数调用），代价是由于重复编码，增加了程序长度。

对于带参数的宏定义，有以下几点需要注意。

☑ 宏定义时，参数要加括号。如不加括号，则结果可能是正确的，也可能是错误的。

例13.1中，如果宏定义语句改为：

```
#define MIX(a,b) (a*b+b)                          /*宏定义，求两个数的混合运算*/
```

则main函数中，x=10，y=9时调用MIX（x，y），可以输出正确结果99；x=10，y=3+4时调用MIX（x，y），执行情况为（10*3+4+3+4），将输出错误结果为41。因此，宏定义时一定要给参数加上括号。

☑ 宏扩展必须使用括号来保护表达式中优先级低的操作符，以确保调用时能达到预期效果。

例13.1中，如果宏扩展外没有加括号，则调用5*MIX（x，y），会被扩展为5*（a）*（b）+（b），而本意是希望得到5*（（a）*（b）+（b）），显然出现了错误。

☑ 对带参数的宏的展开，只是用宏名后括号内的实参字符串代替#define命令行中的形参。

☑ 宏定义时，宏名与带参数的括号之间不要加空格，否则会将空格以后的字符都作为替代字符串的一部分。

☑ 在带参宏定义中，形式参数不分配内存单元，因此不必做类型定义。

编程训练（答案位置：资源包\TM\sl\13\编程训练\）

训练1：用宏计算长方形周长　将长方形的长、宽用宏定义为6、5，再用宏定义计算此长方形周长，最后在主函数中输出周长的标识符。

训练2：计算三角形的的面积　利用带参数的宏定义表达三角形的底和高对应的面积。

13.2　#include命令



使用#include命令可将其他源文件的内容包含进来，即将其他文件包含到本文件之中。被读入的源文件必须用双引号或尖括号括起来。例如，下面两行代码均表示包含了标准输入／输出库文件stdio.h，后续可直接使用其中的函数而不用再定义。

```
#include "stdio.h"
#include <stdio.h>
```

双引号和尖括号的区别如下。

☑ 尖括号：此为标准方式，系统会直接到存放C库函数头文件所在的目录中寻找要包含的文件。

☑ 双引号：系统先在用户当前目录中寻找要包含的文件，若找不到，再到存放C库函数头文件所在的目录中寻找要包含的文件。file1.cfile2.cfile1.c

通常情况下，包含库函数时，使用尖括号可以节省查找时间；包含用户自定义文件时，使用双引号比较快捷。用户自己编写的文件通常保存在当前目录中，如果文件不在当前目录中，双引号内还可以给出文件路径。

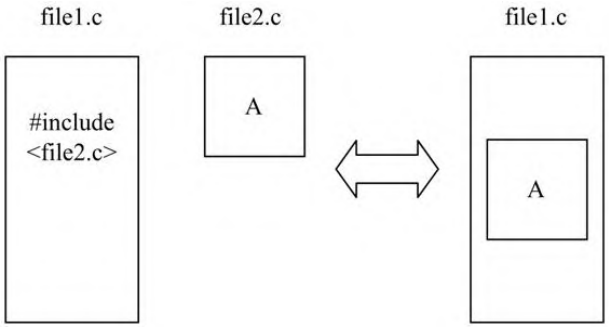


图13.3　文件包含

将文件直接嵌入#include命令内也是可行的，这种方式称为嵌套的嵌入文件。嵌套层次依赖于具体实现，如图13.3所示。

注意

Visual C++ 6.0环境中，是可以省略#include指令的。但现代C语言开发，提倡在使用前交代清楚每个标识符的作用，所以建议开发者合理添加预处理命令#include指令。

【例13.2】一年长一岁（实例位置：资源包\TM\sl\13\02）

（1）文件age.h

```
#define P printf                    /*定义头文件*/
#define D "%d"
#define A (age+1)
```

（2）文件age.c

```
#include "stdio.h"                  /*包含头文件*/
#include "age.h"                    /*包含自定义头文件*/
int main()                          /*主函数 main*/
{
    int age = 3;                    /*定义变量*/
    age=A;                          /*引用头文件包括内容*/
    P("年龄是"D"岁\n",age);         /*输出结果*/
    return 0;                       /*程序结束*/
}
```

程序运行结果如图13.4所示。

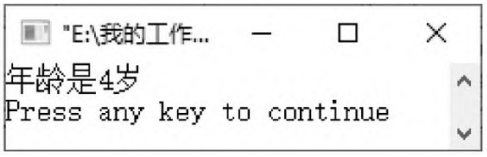


图13.4　一年长一岁

经常用在文件头部的被包含文件称为标题文件或头文件，一般以.h为后缀。一般情况下，将如下内容放到.h文件中。

- ☑ 宏定义。
- ☑ 结构、联合和枚举声明。
- ☑ typedef声明。
- ☑ 外部函数声明。
- ☑ 全局变量声明。

使用文件包含为实现程序修改提供了方便。当需要修改某些参数时，不必逐个修改程序，只需修改一个文件（头部文件）即可。

关于文件包含，有以下几点需要注意。

- ☑ 一个#include命令只能指定一个被包含的文件。
- ☑ 文件包含是可以嵌套的，即在一个被包含文件中还可以包含另一个被包含文件。

☑ 若age.c中包含文件age.h，则预编译后两者会成为一个文件。如果age.h中有全局静态变量，则该全局变量在age.c文件中也有效，这时不需要再用extern声明。

编程训练（答案位置：资源包\TM\sl\13\编程训练\）

训练3：自定义头文件，输出两个数之和　自定义一个sum.h文件，在文件中定义两个变量，然后在sum.c文件中计算这两个变量之和。

```
请输入a、b值：
123 456
a+b=579
```

训练4：输出“中国疫情，武汉加油”　自定义一个ComeOn.h文件，在文件中宏定义printf输出“中国疫情，武汉加油”，然后在ComeOn.c文件中输出结果。

13.3　条件编译

C语言预处理器提供了条件编译功能。一般情况下，源程序中的所有行都会参加编译。如果希望其中一部分内容只在满足一定条件时才进行编译，这时就需要使用条件编译命令。使用条件编译可以非常方便地处理程序的调试版本和正式版本，同时还会增强程序的可移植性。

13.3.1　#if命令



#if的基本含义是：如果#if命令后的参数表达式为真，则编译#if到#endif之间的程序段，否则跳过这段程序。#endif命令用来表示#if段的

结束。

#if命令的一般形式如下：

```
#if 常数表达式
    语句段
#endif
```

如果常数表达式为真，则该段程序被编译，否则跳过该段程序（不编译）。

【例13.3】与50进行比较（实例位置：资源包\TM\sl\13\03）

本实例中，用#if和#define实现与50进行比较。如果等于50，就计算i=i+50；如果大于50，就计算i++；如果小于50，就计算i--。具体代码如下：

```
#include<stdio.h>
#define NUM 50
main()
{
    int i=0;
    #if NUM>50                          /*判断 NUM 是否大于 50*/
        i++;
    #endif
    #if NUM==50
        i=i+50;
    #endif
    #if NUM<50
        i--;
    #endif
    printf("目前的值是::%d\n",i);
}
```

程序运行结果如图13.5所示。若将语句“#define NUM 50”改为“#define NUM 10”，则运行结果如图13.6所示；若改为“#define NUM 100”，运行结果如图13.7所示。

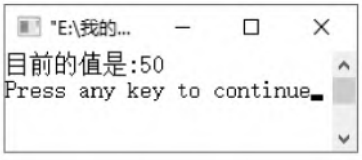


图13.5 #if应用

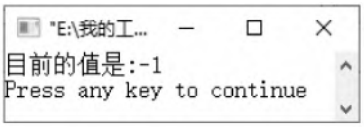


图13.6　NUM为10时

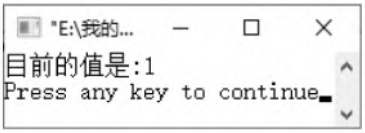


图13.7　NUM为100时

#else的作用是为#if为假时提供另一种选择，其作用和前面讲过的条件判断中的else相近。

【例13.4】判断是否是能被9527整除的偶数（**实例位置：资源包\TM\sl\13\04**）

本实例使用#if与#else语句实现判断功能，代码如下：

```
#include "stdio.h"                                          /*包含头文件*/
#define iInput 19054                                        /*定义宏，用 iInput 替代 19054*/
int main()
{
    printf("当前是：%d\n", iInput);                          /*显示宏定义的数据*/
    #if iInput>0                                            /*判断条件，如果是正数*/
        if(iInput % 9527==0 && iInput %2 == 0)              /*如果能被 9527 整除且为偶数*/
            printf("能被 9527 整除的偶数\n");                /*输出信息*/
    #else
        #if iInput<0                                        /*如果是负数*/
            printf("当前为负数：%d\n", iInput);              /*输出结果*/
        #else                                               /*如果是其他情况*/
            printf("不能被 9527 整除的偶数，此时数据为：%d\n", iInput);  /*显示不满足条件的数据*/
        #endif
    #endif
    return 0;                                               /*程序结束*/
}
```

程序运行结果如图13.8所示。

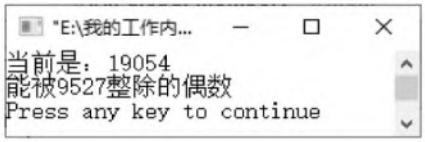


图13.8　#else应用

#elif指令用来建立“如果……或者如果……”这样阶梯状多重编译操作选择，与多分支if语句中的else if类似。

#elif的一般形式如下：

```
#if 表达式 1
    语句段 1
#elif 表达式 2
    语句段 2
#elif 表达式 3
    语句段 3
    ...
#elif 表达式 n
    语句段 n
#endif
```

【例13.5】判断输入的体温是否为正常体温（实例位置：资源包\TM\sl\13\05）

本实例中，利用#elif判断输入的体温是否为正常体温。代码如下：

```
#include<stdio.h>
#define NUM 50
main()
{
    int i=0;
    #if NUM>50
        i++;
    #elif NUM==50
        i=i+50;
    #else
        i--;
    #endif
    printf("i is:%d\n",i);
}
```

13.3.2 #ifdef及#ifndef命令



在#if条件编译命令中，需要判断符号常量所定义的具体值。但有时并不需要判断具体值，只需要知道这个符号常量是否被定义了，这时可以

采用另一种条件编译的方法，即#ifdef与#ifndef命令，分别表示“如果有定义”及“如果无定义”。

#ifdef的一般形式如下：

```
#ifdef 宏替换名
    语句段
#endif
```

其含义是：如果宏替换名已被定义过，则对语句段进行编译；如果未定义#ifdef后面的宏替换名，则不对语句段进行编译。#ifdef可与#else连用，形式如下：

```
#ifdef 宏替换名
    语句段 1
#else
    语句段 2
#endif
```

其含义是：如果宏替换名已被定义过，则对语句段1进行编译；如果未定义#ifdef后面的宏替换名，则对语句段2进行编译。

#ifndef的一般形式如下：

```
#ifndef 宏替换名
    语句段
#endif
```

其含义是：如果未定义#ifndef后面的宏替换名，则对语句段进行编译；如果定义#ifndef后面的宏替换名，则不执行语句段。同样，#ifndef也可以与#else连用，一般形式如下：

```
#ifndef 宏替换名
    语句段 1
#else
    语句段 2
#endif
```

其含义是：如果未定义#ifndef后面的宏替换名，则对语句段1进行编译；如果定义#ifndef后面的宏替换名，则对语句段2进行编译。

【例13.6】模拟银行叫号服务（实例位置：资源包\TM\sl\13\06）

去银行办理业务时，需要先取号，如号码为100号，则需等到叫100号时才可以办理业务；如果没有取号，则不能办理业务。使用#ifdef和#ifndef编写程序，模拟此场景。

```
#include<stdio.h>
#define STR "100 号"                          /*定义宏常量*/
int main()
{
    #ifdef STR                                /*取到号码*/
        printf(STR);                          /*输出叫号信息*/
        printf("能办理业务\n");                /*输出提示信息*/
    #endif
    printf("\n");
    #ifndef ABC                               /*没取号码，表示没被定义*/
        printf("没取到号不能办理业务\n");       /*输出提示信息*/
    #endif
        return 0;
}
```

程序运行结果如图13.9所示。

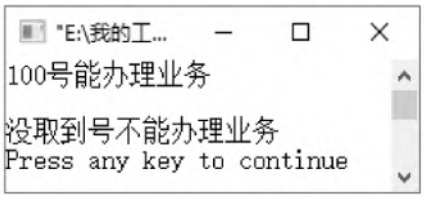


图13.9　模拟银行叫号

13.3.3　#undef命令



使用#undef命令可以删除事先定义好的宏定义。其一般形式如下：

```
#undef 宏替换名
```

使用#undef命令可将宏名限制在特定的代码段中。例如：

```
#define MAX_SIZE 100
char array[MAX_SIZE];
#undef MAX_SIZE
```

上述代码中，首先使用#define定义标识符MAX_SIZE，然后使用#undef删除宏定义。也就是说，遇到#undef语句之前，MAX_SIZE的定义都是有效的。

13.3.4 #line命令



#line命令用于显示_LINE_与_FILE_的内容。_LINE_用于存放当前编译行的行号，_FILE_用于存放当前编译的文件名。#line命令的一般形式如下：

```
#line 行号["文件名"]
```

其中，行号为任一正整数，可选的文件名为任意有效文件标识符。行号为源程序中当前行号，文件名为源文件的名字。#line命令主要用于调试及其他特殊应用。

【例13.7】输出当前行号（实例位置：资源包\TM\sl\13\07）

本实例中，利用#line输出当前行号，代码如下：

```
#line 100 "13.7.C"
#include<stdio.h>
int main()
{
    printf("1.当前行号：%d\n",__LINE__);
    printf("2.当前行号：%d\n",__LINE__);
    return 0;
}
```

程序运行结果如图13.10所示。

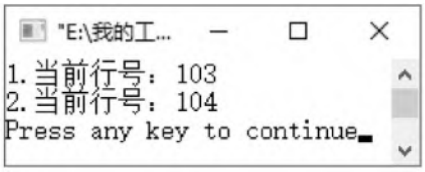


图13.10　输出当前行号

13.3.5　#pragma命令



1．#pragma命令

#pragma命令用于设定编译器状态，或指示编译器完成一些特定动作。一般形式如下：

```
#pragma 参数
```

☑ message参数：在编译信息窗口中输出的相应信息。

☑ code_seg参数：设置程序中函数代码存放的代码段。

☑ once参数：保证头文件被编译一次。

2．预定义宏名

ANSI标准说明了以下5个预定义宏替换名。

☑ __LINE__：当前被编译代码的行号。

☑ __FILE__：当前源程序的文件名称。

☑ __DATE__：当前源程序的创建日期。

☑ __TIME__：当前源程序的创建时间。

☑ __STDC__：判断当前编译器是否为标准C。其值为1，是标准C，否则不是标准C。

宏名的书写比较特别，书写时两边都要由下画线构成。如果编译不是标准的，则可能仅支持以上宏名中的几个，或根本不支持。编译程序有时还提供其他预定义的宏名。

编程训练（答案位置：资源包\TM\sl\13\编程训练\）

训练5：电报译文　输入一行电报，任选两种方式输出：一种是原文输出，另一种是将字母变为其下一个字母，如a变成b，b变为c，……，y变成z，z变成a，其他字符不变。用条件编译方法实现，用#define命令来控制是否要译成密码。运行结果如下：

```
请输入电报：
Oh My God
输出：
Pi Nz Hpe
```

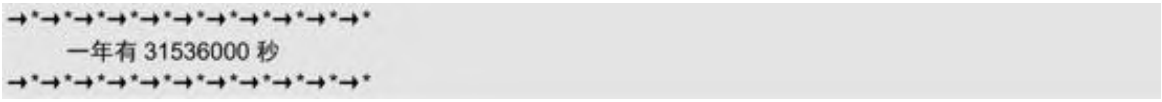
训练6：秘密电报　输入一行报文，将报文源码输出，或输出相同个数的“*”。运行结果如下：

```
请输入信息：
hello
加密之后的消息为：
*****
```

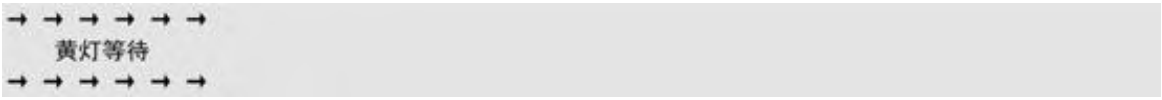
13.4　实践与练习

（答案位置：资源包\TM\sl\13\实践与练习\）

综合练习1：一年有多少秒　用预处理命令声明一个常量，表示一年有多少秒（按每年365天计算），然后输出该信息。运行结果如下：



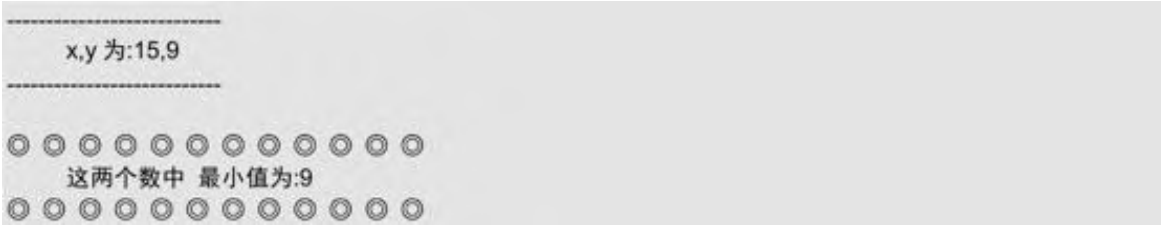
综合练习2：模拟交通灯场景　利用#if…#else编程，输入某个数字，打印出当前交通灯的状态。如输入数字1，输出“红灯停”；输入数字2，输出“绿灯行”；输入数字3，输出“黄灯等待”。运行结果如下：



综合练习3：求两数乘积　输入两个整数，求其乘积，用带参的宏实现。运行结果如下：

```
请输入两个数：
59  64
乘积结果为:3776
```

综合练习4：求最小值　定义一个带参数的宏，比较数值15和9，并返回最小值，运行结果如下：



综合练习5：一年长一岁　自定义头文件age.h，将printf定义一个符号常量，将%d定义一个符号常量，将（age+1）再定义一个符号常量，最后在age.c中输出年龄。运行结果如下：

```
#-#-#-#-#-#-#-#-#-#-#
      年龄是 4 岁
#-#-#-#-#-#-#-#-#-#-#
```

综合练习6：你的成绩合格吗？　小红的考试成绩为91分，利用#if编写程序，判断小红是否及格（及格分是60分）。运行结果如下：

```
及格了
☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆
    当前分数:91
☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆
```

综合练习7：摄影收费标准　某摄影工作室制定的收费标准如下：清纯型235元，异域风情照399元，双人照（如姐妹、情侣照）599元，婚纱照1999元。利用#elif模拟此价格标准，运行结果如下：

结果1：

```
@-@-@-@-@-@-@-@-@-@
    婚纱照
@-@-@-@-@-@-@-@-@-@
```

结果2：

```
卍 卍 卍 卍 卍
    清纯型
卍 卍 卍 卍 卍
```

结果3：

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
  双人照（姐妹照，情侣照）
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

结果4：

```
----------------
  异域风情
----------------
```

综合练习8：自定义文件输出谚语　自定义文件编写程序，使其输出效果如下：

★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★
今日的事情，尽心，尽力，尽意去做了。
无论成绩如何，都应该高高兴兴地上床恬睡
★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★

第14章
文件

文件是一组相关数据的有序集合，是程序设计中的一个重要概念。通常情况下，使用计算机主要是在使用文件。要实现数据处理，往往也需要通过文件的形式来完成。本章就来介绍如何将数据写入文件以及如何从文件中读取数据。

本章的知识架构及重难点如下：



14.1 文件概述



文件是一组相关数据的有序集合，这个数据集有一个名称，叫作文件名。

通常情况下，使用计算机主要是在使用文件。输入和输出指的是从标准输入设备（键盘）输入，由标准输出设备（显示器或打印机）输出。磁盘作为信息载体，用于保存中间结果或最终数据。在使用一些字处理工具时，打开文件可将磁盘的信息输入内存，关闭文件可将内存的数据输出到磁盘。这时，输入、输出是针对文件系统的。

所有文件都可以通过流进行输入、输出操作。与文本流和二进制流相对应，文件可以分为文本文件和二进制文件两大类。文本文件也称为ASCII文件，保存时每个字符对应一个字节，存放对应的ASCII码。二进制文件按二进制编码方式保存文件内容。

按文件内容看，可分为源文件、目标文件、可执行文件、头文件和数据文件等。

从用户角度（或所依附的介质）看，文件可分为普通文件和设备文件两种。

- ☑ 普通文件：指驻留在磁盘或其他外部介质上的一个有序数据集。
- ☑ 设备文件：指与主机相连的各种外部设备，如显示器、打印机、键盘等。操作系统中把外部设备也看作一个文件来进行管理，把它们的输入、输出等同于对磁盘文件的读和写。

C语言中，文件操作都是由库函数来完成的。本章将介绍主要的文件操作函数。

14.2 文件基本操作

文件的基本操作包括文件的打开和关闭。除标准的输入、输出文件外，其他所有文件都必须先打开再使用，使用后还必须关闭该文件。

14.2.1 文件指针



文件指针是一个指向文件有关信息的指针，这些信息包括文件名、状态和当前位置，保存在一个结构体变量中。使用文件时需要在内存中为其分配空间，用来存放文件的基本信息。该结构体类型是由系统定义的，C语言规定该类型为FILE型，其声明如下：

```
typedef struct
{
    short level;
    unsigned flags;
    char fd;
    unsigned char hold;
    short bsize;
    unsigned char *buffer;
    unsigned ar *curp;
    unsigned istemp;
    short token;
}FILE;
```

上述代码中，使用typedef定义了一个FILE结构体类型，编写程序时可直接使用FILE类型来定义变量。注意，定义变量时不必将结构体内容全部给出，只需写成如下形式（fp是一个指向FILE类型的指针变量）：

```
FILE *fp;
```

说明

不可以通过定义FILE类型变量来操作文件。FILE型数据对象的位置由库函数确定，C语言中只能通过FILE *类型的指针来操作文件。

14.2.2 文件的打开



fopen函数用来打开一个文件，打开文件的操作就是创建一个流。fopen函数的原型在stdio.h中，其调用形式一般如下：

```
FILE *fp;
fp=fopen(文件名,使用文件方式);
```

其中，“文件名”是将要被打开文件的文件名，“使用文件方式”是指对打开文件进行的读写方式（见表14.1）。

表14.1 使用文件方式

文件使用方式	含 义
r（只读）	打开一个文本文件，只允许读数据
w（只写）	打开或建立一个文本文件，只允许写数据
a（追加）	打开一个文本文件，并在文件末尾写数据
rb（只读）	打开一个二进制文件，只允许读数据
wb（只写）	打开或建立一个二进制文件，只允许写数据
ab（追加）	打开一个二进制文件，并在文件末尾写数据
r+（读写）	打开一个文本文件，允许读和写
w+（读写）	打开或建立一个文本文件，允许读写
a+（读写）	打开一个文本文件，允许读，或在文件末追加数据
rb+（读写）	打开一个二进制文件，允许读和写
wb+（读写）	打开或建立一个二进制文件，允许读和写
ab+（读写）	打开一个二进制文件，允许读，或在文件末追加数据

例如，以只读方式打开名为123的文本文档文件，代码如下：

```
FILE *fp;
fp= fopen ("123.txt","r");
```

如果使用fopen函数打开文件成功，则返回一个有确定指向的FILE类型指针；若打开失败，则返回NULL。文件打开失败的原因通常是以下3个方面：

☑ 指定的盘符或路径不存在。

☑ 文件名中含有无效字符。

☑ 以r模式打开一个不存在的文件。

14.2.3 文件的关闭



文件使用完毕后，应使用fclose函数将其关闭。fclose函数的原型也在stdio.h中，调用形式如下：

```
fclose(文件指针);
```

正常完成关闭文件操作时，fclose函数的返回值为0，否则返回EOF。例如：

```
fclose(fp);
```

说明

在程序结束之前应关闭所有文件，这样做的目的是防止因为没有关闭文件而造成的数据丢失。

14.3 文件的读写

打开文件后，可对文件进行读出或写入的操作。C语言提供了丰富的文件操作函数，本节将对其进行详细介绍。

14.3.1 fputc函数



fputc函数用于把一个字符写到磁盘文件（fp所指向的文件）中。其一般形式如下：

```
ch=fputc(ch,fp);
```

其中，ch是要写入的字符，可以是一个字符常量，也可以是一个字符变量。fp是文件指针变量，如果函数写入成功，则返回值就是写入的字符；如果写入失败，则返回EOF。

【例14.1】将名人语录写入文件中（实例位置：资源包\TM\sl\14\01）

创业指导课上，老师要求每个学生说一句关于创业的经典语录。将马云的经典语录“Together we creat a team culture，rather than complaining about culture.”写到文件中，以“#”结束输入。

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fp;                                   /*定义一个指向 FILE 类型结构体的指针变量*/
    char ch;                                    /*定义变量为字符型*/
    if((fp = fopen("E:\\exp01.txt", "w")) == NULL)   /*以只写方式打开指定文件*/
    {
        printf("不能打开文件\n");
        exit(0);
    }
    printf("请输入名人名言:\n");
    ch = getchar();                             /*getchar 函数带回一个字符赋予 ch*/
    while(ch != '#')                            /*当输入"#"时结束循环*/
    {
        fputc(ch, fp);                          /*将读入的字符写到磁盘文件中*/
        ch = getchar();                         /*getchar 函数继续带回一个字符赋给 ch*/
    }
    fclose(fp);                                 /*关闭文件*/
    return 0;
}
```

当输入如图14.1所示的内容时，E:\exp01.txt文件中的内容如图14.2所示。

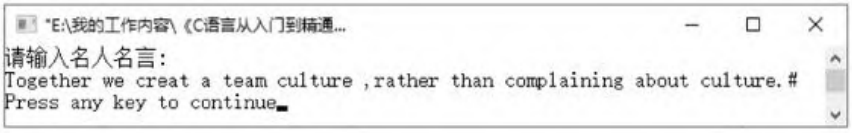


图14.1　运行界面

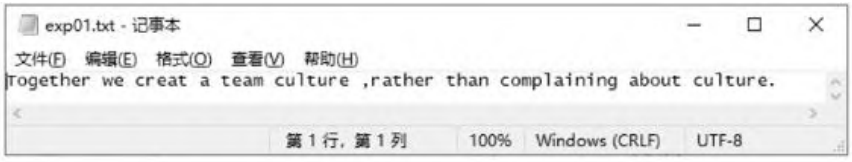


图14.2　文件中的内容

14.3.2　fgetc函数



fgetc函数用于从指定文件（fp指向的文件）中读取一个字符赋予ch。其一般形式如下：

```
ch=fgetc(fp);
```

注意，文件必须以读或读写方式打开。当函数遇到文件结束符时，将返回文件结束标志EOF。

【例14.2】读取《早发白帝城》诗句（实例位置：资源包\TM\sl\14\02）

在E盘创建一个文件poem.txt，内容为李白的《早发白帝城》，将其中的诗句输出在控制台上。

```
#include<stdio.h>
int main()
{
    FILE *fp;                          /*定义一个指向 FILE 类型结构体的指针变量*/
    char ch;                           /*定义变量及数组为字符型*/
    fp = fopen("E:\\poem.txt", "r");   /*以只读方式打开指定文件*/
    ch = fgetc(fp);                    /*fgetc 函数读取一个字符并赋予 ch*/
    while(ch != EOF)                   /*当读取的字符值等于 EOF 时，结束循环。如果不等于 EOF*/
    {
        putchar(ch);                   /*将读取的字符输出到屏幕上*/
        ch = fgetc(fp);                /*fgetc 函数继续读取下一个字符并赋予 ch*/
    }
    printf("\n");
    fclose(fp);                        /*关闭文件*/
    return 0;
}
```

运行程序，显示效果如图14.3所示。

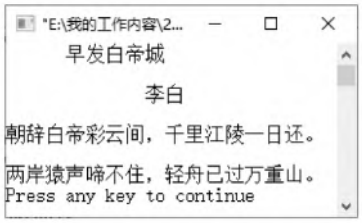


图14.3　读取磁盘文件

误区警示

fgets函数和gets函数的区别如下。

gets函数读完回车换行（'\n'），并不存储它；fgets函数存储换行符并在其后写一个null character（'\0'）。不能简单地把fgets函数理解为gets函数限制字符数目的stdin加强版。在改用fgets函数后，应该注意把字符串末尾之前的'\n'删除。

14.3.3 fputs函数



fputs函数与fputc函数类似，用于向指定的文件写入一个字符串。其一般形式如下：

```
fputs(字符串,文件指针)
```

其中，字符串可以是字符串常量，也可以是字符数组名、指针或变量。

【例14.3】录入歌词（实例位置：资源包\TM\sl\14\03）

某唱片公司为歌手录制MV，需要将歌词录入文件中。代码如下：

```
#include<stdio.h>
#include<process.h>
int main()
{
    FILE *fp;
    char filename[30],str[30];                          /*定义两个字符型数组*/
    printf("请输入文件路径:\n");
    scanf("%s",filename);                               /*输入文件名*/
    if((fp=fopen(filename,"w"))==NULL)                  /*判断文件是否打开失败*/
    {
        printf("不能打开文件!\n 请按任意键结束\n");
        getchar();
        exit(0);
    }
    printf("请输入歌词:\n");                            /*提示输入字符串*/
    getchar();
    gets(str);
    fputs(str,fp);                                      /*将字符串写入 fp 所指向的文件中*/
    fclose(fp);
    return 0;
}
```

程序运行效果如图14.4所示，写入的文件内容如图14.5所示。

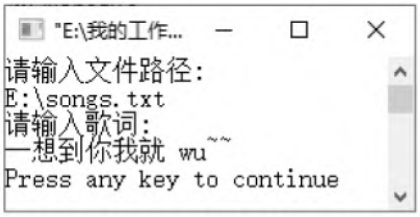


图14.4　程序运行效果



图14.5　写入文件的内容

14.3.4　fgets函数



fgets函数与fgetc函数类似，用于从指定的文件中读取一个字符串到字符数组中。其一般形式如下：

```
fgets(字符数组名,n,文件指针);
```

其中，n表示读取的字符串中字符的个数（包含“\0”）。

【例14.4】读取歌曲《500 Miles》歌词（实例位置：资源包\TM\sl\14\04）

在E盘创建一个文件sing.txt，文件内容为《500 Miles》的一句歌词，将内容输出在控制台上。

```
#include<stdio.h>
#include<process.h>
int main()
{
    FILE *fp;
    char filename[30],str[30];                    /*定义两个字符型数组*/
    printf("请输入文件路径:\n");
    scanf("%s",filename);                         /*输入文件名*/
    if((fp=fopen(filename,"r"))==NULL)            /*判断文件是否打开失败*/
    {
        printf("不能打开文件!\n 请按任意键结束\n");
        getchar();
        exit(0);
    }
    fgets(str,sizeof(str),fp);                    /*读取磁盘文件中的内容*/
    printf("%s",str);
    printf("\n");
    fclose(fp);
    return 0;
}
```

所要读取的磁盘文件中的内容如图14.6所示，程序运行效果如图14.7所示。



图14.6　文件中的内容

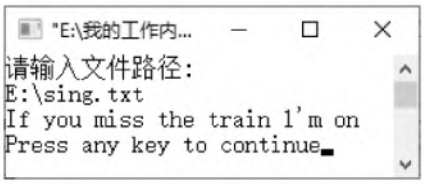


图14.7　程序运行效果

14.3.5　fprintf函数



前面讲过printf和scanf函数，两者都是格式化读写函数。fprintf和fscanf函数与之相似，但读写对象不是终端，而是磁盘文件。

fprintf函数的一般形式如下：

```
ch=fprintf(文件类型指针,格式字符串,输出列表);
```

例如，下面语句的作用是将整型变量i的值以“%d”的格式输出到fp指向的文件中。

```
fprintf(fp,"%d",i);
```

【例14.5】打印巴斯卡三角形（实例位置：资源包\TM\sl\14\05）

输出3行巴斯卡三角形，如下所示：

1

1 1

1 2 1

```
#include<stdio.h>
#include<process.h>
int main()
{
    FILE *fp;                              /*定义一个指向 FILE 类型结构体的指针变量*/
    int i=1;
    int j=2;
    char filename[30];                     /*定义一个字符型数组*/
    printf("请输入保存文件路径:\n");          /*输入文件名*/
    scanf("%s",filename);
    if((fp=fopen(filename,"w"))==NULL)     /*判断文件是否打开失败*/
    {
        printf("不能打开文件!\n 请按任意键结束\n");
        getchar();
        exit(0);
    }
    fprintf(fp,"%4d\n",i);                 /*写入 fp 所指的磁盘文件中*/
    fprintf(fp,"%2d",i);
    fprintf(fp,"%4d\n",i);
    fprintf(fp,"%d",i);
    fprintf(fp,"%3d",j);
    fprintf(fp,"%4d\n",i);
    fclose(fp);
    return 0;
}
```

程序运行效果如图14.8所示。将巴斯卡三角形写入磁盘文件中，最终文件效果如图14.9所示。

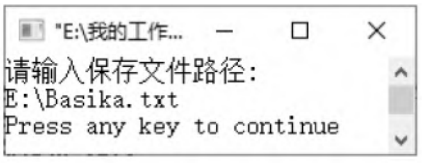


图14.8　程序运行效果



图14.9 生成的文件效果

14.3.6 fscanf函数



fscanf函数的一般形式如下：

```
fscanf(文件类型指针,格式字符串,输入列表);
```

例如，下面语句的作用是读取fp指向文件中的i的值。

```
fscanf(fp,"%d",&i);
```

【例14.6】公布选择题答案（实例位置：资源包\TM\sl\14\06）

期末考试后，为便于审阅试卷，需要编写程序将选择题答案显示出来。例如，答案是ACBDDCBAD CBCAAB。运行程序前需要在E盘设置answer.txt文件，内容是选择题答案。代码如下：

```
#include<stdio.h>
#include<process.h>
int main()
{
    FILE *fp;
    char i,j;
    char filename[30];                          /*定义一个字符型数组*/
    printf("请输入文件路径:\n");
    scanf("%s",filename);                       /*输入文件名*/
    if((fp=fopen(filename,"r"))==NULL)          /*判断文件是否打开失败*/
    {
        printf("不能打开文件!\n 请按任意键继续\n");
        getchar();
        exit(0);
    }
    for(i=0;i<15;i++)
    {
        fscanf(fp,"%c",&j);
        printf("%d answer is:%5c\n",i+1,j);
    }
    fclose(fp);
    fclose(fp);
    return 0;
}
```

所读取的磁盘文件中的内容如图14.10所示。程序运行界面如图14.11所示。



图14.10　文件中的内容

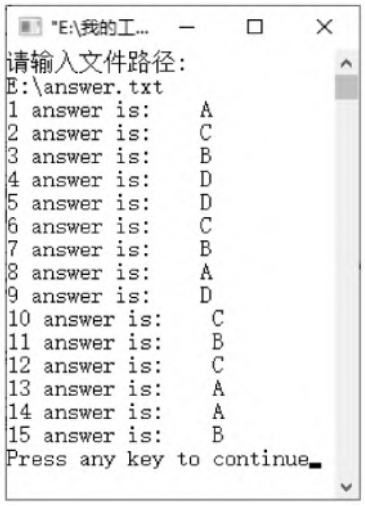


图14.11　程序运行效果

14.3.7　fread和fwrite函数



实际开发中，往往需要对整块数据进行读写，例如，对一个结构体类型变量值进行读写。下面介绍可实现整块读写功能的fread和fwrite函数。

fread函数的功能是：从fp指向的文件中读取count次，每次读size字节，读取的信息保存在buffer地址中。其一般形式如下：

```
fread(buffer,size,count,fp);
```

fwrite函数的功能是：将buffer地址开始的信息输出count次，每次写size字节到fp指向的文件中。其一般形式如下：

```
fwrite(buffer,size,count,fp);
```

- ☑ buffer：一个指针。对fwrite函数来说，是待输出数据的地址（起始地址）；对fread函数来说，是待读取数据存放的地址。
- ☑ size：要读写的字节数。
- ☑ count：要读写多少个size字节的数据项。
- ☑ fp：文件型指针。

例如，下面的语句表示从fp指向的文件中每次读取两个字节并保存在数组a中，连续读取3次。

```
fread(a,2,3,fp);
```

下面的语句表示将数组a中的信息每次输出两个字节到fp指向的文件中，连续输出3次。

```
fwrite(a,2,3,fp);
```

【例14.7】新实习生信息（实例位置：资源包\TM\sl\14\07）

某企业招聘，经过初复试，筛选出3名实习生。输入实习生信息，将信息显示出来。

```
#include<stdio.h>
#include<process.h>
struct address_list                                          /*定义结构体，存储实习生成绩信息*/
{
    char name[10];
    char adr[20];
    char tel[15];
} info[100];
void save(char *name, int n)                                 /*自定义 save 函数*/
{
    FILE *fp;                                                /*定义一个指向 FILE 类型结构体的指针变量*/
    int i;
    if((fp = fopen(name, "wb")) == NULL)                     /*以只写方式打开指定文件*/
    {
        printf("不能打开文件\n");
        exit(0);
    }
    for(i = 0; i < n; i++)
        if(fwrite(&info[i], sizeof(struct address_list), 1, fp) != 1) /*将一组数据输出到 fp 指向的文件中*/
            printf("写入有误\n");                            /*如果写入文件不成功，则提示错误*/
    fclose(fp);                                              /*关闭文件*/
}
void show(char *name, int n)                                 /*自定义 show 函数*/
{
    int i;
    FILE *fp;                                                /*定义一个指向 FILE 类型结构体的指针变量*/
    if((fp = fopen(name, "rb")) == NULL)                     /*以只读方式打开指定文件*/
    {
        printf("不能打开文件\n");
        exit(0);
    }
    for(i = 0; i < n; i++)
    {
        fread(&info[i], sizeof(struct address_list), 1, fp);    /*从 fp 指向的文件中读取数据存入 score 数组*
        printf("%15s%20s%20s\n", info[i].name, info[i].adr,info[i].tel);
    }
    fclose(fp);                                              /*关闭文件*/
}
int main()
{
    int i, n;                                                /*变量类型为基本整型*/
    char filename[50];                                       /*数组为字符型*/
    printf("一共几名?\n");
    scanf("%d", &n);                                         /*输入实习生人数*/
    printf("请输入文件路径:\n");
    scanf("%s", filename);                                   /*输入文件所在路径及名称*/
    printf("请输入姓名，地址，手机号\n");
    for (i = 0; i < n; i++)                                  /*输入实习生信息*/
    {
        printf("NO%d", i + 1);
        scanf("%s%s%s", info[i].name, info[i].adr, info[i].tel);
        save(filename, n);                                   /*调用函数 save*/
    }
    printf("信息如下:\n");
    show(filename, n);                                       /*调用函数 show*/
}
```

程序运行结果如图14.12所示。



图14.12　录入并显示实习生信息

编程训练（答案位置：资源包\TM\sl\14\编程训练\）

训练1：输出疫情数据　在F盘创建一个exp.txt文件，内容为2月某天吉林省肺炎疫情数据，将数据输出在控制台上。

训练2：默写诗歌《咏鹅》　在控制台上默写《咏鹅》，并保存在文件中。

14.4　文件的定位

对文件进行操作时，往往不需要从头开始，而是只操作指定内容。这时，就需要使用文件定位函数来实现对文件的随机读取。下面介绍3个随机读写函数。

14.4.1　fseek函数



fseek函数的作用是移动文件内部的位置指针。其一般形式如下：

```
fseek(文件类型指针,位移量,起始点);
```

其中，“文件类型指针”指向被移动的文件；“位移量”表示移动的字节数，一般为long型数据，以保证文件长度大于64KB时不会出错；“起始点”表示从何处开始计算位移量，一般是文件首、文件当前位置和文件尾，其表示方法如表14.2所示。

表14.2　起始点

起　始　点	表 示 符 号	数 字 表 示
文件首	SEEK—SET	0
文件当前位置	SEEK—CUR	1
文件尾	SEEK—END	2

例如，下面的语句表示将位置指针从当前位置向后退20个字节。

```
fseek(fp,-20L,1);
```

注意，fseek函数一般用于二进制文件。在文本文件中使用时，由于要进行转换，计算的位置会出现错误。

文件的随机读写在移动位置指针之后进行，即可用前面介绍的任一种读写函数进行读写。

【例14.8】快递员送快递（实例位置：资源包\TM\sl\14\08）

快递员通常会将收货人电话的后4位写到快递包裹上，作为取货号使用。编写程序，例如收货人电话是123****8900，则输出取货号8900。运行程序前，需要在E盘设置goods.txt文件，内容是收货人的手机号。代码如下：

```
#include<stdio.h>
#include<process.h>
int main()
{
    FILE *fp;
    char filename[30],str[50];                    /*定义两个字符型数组*/
    printf("请输入文件路径:\n");
    scanf("%s",filename);                         /*输入文件名*/
    if((fp=fopen(filename,"wb"))==NULL)           /*判断文件是否打开失败*/
    {
        printf("不能打开文件!\n 请按任意键继续\n");
        getchar();
        exit(0);
    }
    printf("请输入字符串:\n");
    getchar();
    gets(str);
    fputs(str,fp);
    fclose(fp);
    if((fp=fopen(filename,"rb"))==NULL)           /*判断文件是否打开失败*/
    {
        printf("不能打开文件!\n 请按任意键继续\n");
        getchar();
        exit(0);
    }
    fseek(fp,7L,0);
    fgets(str,sizeof(str),fp);
    printf("\n 取货号为:\n");
    puts(str);
    fclose(fp);
    return 0;
}
```

程序运行结果如图14.13所示。

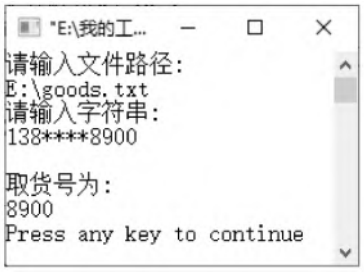


图14.13　输出快递取货号

其中，如下代码的含义是将文件指针指向距文件首5个字节的位置，也就是指向字符串中的第6个字符。

```
fseek(fp,5L,0);
```

14.4.2 rewind函数



rewind函数用于将位置指针重返文件开头，没有返回值。其一般形式如下：

```
int rewind(文件类型指针)
```

【例14.9】给支付宝设置密码（实例位置：资源包\TM\sl\14\09）

给支付宝设置密码时，用户输入一次密码后，会提示再输入一次密码，模拟此场景。运行程序之前，需要在E盘设置password.txt文件，内容是设置的密码。

```
#include<stdio.h>
#include<process.h>
int main()
{
    FILE *fp;
    char ch,filename[50];
    printf("请输入文件路径:\n");
    scanf("%s",filename);                          /*输入文件名*/
    if((fp=fopen(filename,"r"))==NULL)             /*以只读方式打开该文件*/
    {
        printf("不能打开文件\n");
        exit(0);
    }
    printf("\n 输入密码:\n");
    ch = fgetc(fp);
    while (ch != EOF)
    {
        putchar(ch);                               /*输出字符*/
        ch = fgetc(fp);                            /*获取 fp 指向文件中的字符*/
    }
    rewind(fp);                                    /*指针指向文件开头*/
    printf("\n 请再次输入密码:\n");
    ch = fgetc(fp);
    while (ch != EOF)
    {
        putchar(ch);                               /*输出字符*/
        ch = fgetc(fp);
    }
    printf("\n");
    fclose(fp);                                    /*关闭文件*/
    return 0;
}
```

程序运行结果如图14.14所示。来看一下下方的6行语句：

```
ch = fgetc(fp);
while(ch != EOF)
{
    putchar(ch);
    ch = fgetc(fp);
}
```

通过以上语句，输出了第一个956845，输出后文件指针移动到了文件尾部。使用rewind函数再次将文件指针移到文件开始部分，因此当再次使用上面6行语句时，会输出第二个956845。

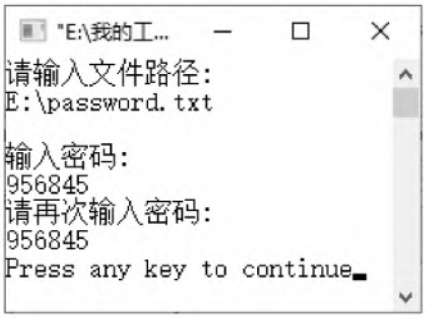


图14.14　为支付宝设置密码

14.4.3　ftell函数



ftell函数可得到流式文件中的当前位置，并使用相对于文件头的位移量表示。其一般形式如下：

```
long ftell(文件类型指针)
```

当ftell函数的返回值为-1L时，表示出错。

【例14.10】设置银行卡密码（实例位置：资源包\TM\sl\14\10）

银行卡密码只能是6位，设置6位，表示设置密码成功，否则失败。运行程序前，先在E盘设置cipher.txt文件，内容是银行卡密码。具体代码如下：

```
#include<stdio.h>
#include<process.h>
int main()
{
    FILE *fp;                                   /*定义一个指向 FILE 类型结构体的指针变量*/
    int n;
    char ch,filename[50];
    printf("请输入文件路径:\n");
    scanf("%s",filename);                       /*输入文件位置*/
    if((fp=fopen(filename,"r"))==NULL)          /*判断是否能打开文件*/
    {
        printf("不能打开此文件\n");
        exit(0);
    }
    printf("密码为:");
    ch = fgetc(fp);                             /*读出一个字符*/
    while (ch != EOF)
    {
        putchar(ch);
        ch = fgetc(fp);
    }
    n=ftell(fp);                                /*输出长度*/
    if(6==n)                                    /*判断长度是否等于 6*/
        printf("\n 设置密码成功\n");
    else
        printf("\n 设置密码失败\n");
    fclose(fp);                                 /*关闭文件*/
    return 0;
}
```

程序运行结果如图14.15所示。

当需要将某个文件中的内容复制到另一个文件中时，可使用fseek函数将文件指针指向文件尾，这样就可以将另一个文件中的内容逐个写到该文件中所有内容的后面，从而实现复制操作。

【例14.11】将两个文件内容合并到一起（实例位置：资源包\TM\sl\14\11）

在E盘创建两个文件saying1、saying2，其中saying1.txt内容是“花儿凋谢不再开，”，saying2.txt内容是“光阴一去不再来。”。编写程序，将文件2的内容复制到文件1中，代码如下：

```
#include<stdio.h>
#include<process.h>
main()
{
    FILE *fp1,*fp2;
    char ch,filename1[30],filename2[30];
    printf("请输入文件 1 的名字：\n");
    scanf("%s",filename1);
    printf("请输入文件 2 的名字：\n");
    scanf("%s",filename2);
    if((fp1=fopen(filename1,"ab+"))==NULL)
    {
        printf("不能打开文件，请按任意键继续\n");
        getchar();
        exit(0);
    }
    if((fp2=fopen(filename2,"rb"))==NULL)
    {
        printf("不能打开文件，请按任意键继续\n");
        getchar();
        exit(0);
    }
    fseek(fp1,0L,2);
    while((ch=fgetc(fp2))!=EOF)
    {
        fputc(ch,fp1);
    }
    fclose(fp1);
    fclose(fp2);
}
```

程序运行结果如图14.16所示。

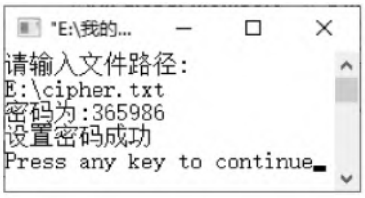


图14.15　设置银行卡密码

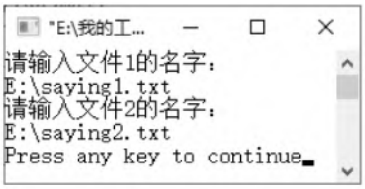


图14.16　将两个文件内容合并到一起

复制前，两个文件中的内容如图14.17和图14.18所示。复制完成后，文件1中的内容如图14.19所示。



图14.17　文件1内容



图14.18　文件2内容



图14.19　执行复制操作后文件1的内容

编程训练（答案位置：资源包\TM\sl\14\编程训练\）

训练3：happy造句　用单词happy写句子，要求字母长度在10个以上。某同学写完将其输入F盘的demo.txt文件中，编写程序，输出字母长度是多少。。

训练4：获取生日信息　编写程序，输入出生年月日，如19900202，显示该员工哪天过生日。

14.5 实践与练习

（答案位置：资源包\TM\sl\14\实践与练习\）

提示：建议在F盘创建练习中所要用的文本文件（.txt）。

综合练习1：读取蚂蚁庄园动态文件 创建如图14.20所示的文件，利用读操作读取文件内容，并输出。程序运行效果如下：

```
请输入文件名：
F:\manor.txt
你使用了一张加速卡，小鸡撸起袖子开始双手吃饲料，进食速度大大加快。
```

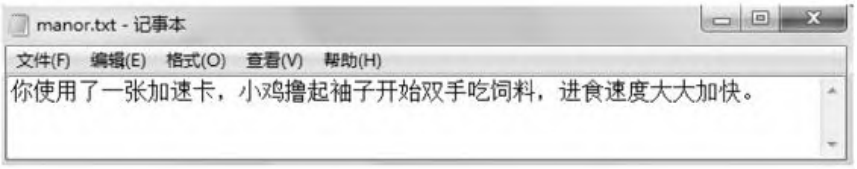


图14.20 蚂蚁庄园动态内容

综合练习2：合并文件 首先创建一个ant.txt，内容如图14.21所示，与题目1中创建的manor.txt这两个.txt文件合并成一个.txt文件，效果如图14.22所示，文件内容如图14.23所示。

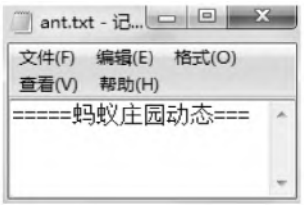


图14.21 ant.txt文件

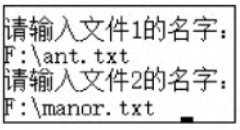


图14.22 合并文件效果

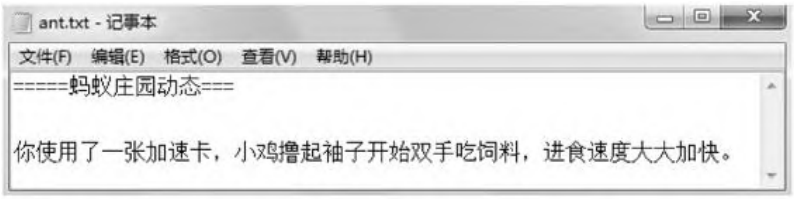


图14.23　合并之后文件内容

综合练习3：模拟淘宝客服的自动回复　为了应对大量的买家，很多电商客服会预先设置许多自动回复内容。创建两个文件（automatic.txt和reply.txt），如图14.24是客服编辑好的automatic.txt的文本内容，其中reply.txt是空的，图14.25是运行效果图，图14.26是复制之后reply.txt文件内容。

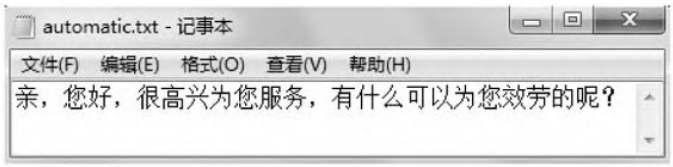


图14.24　automatic.txt文件内容



图14.25　程序运行效果

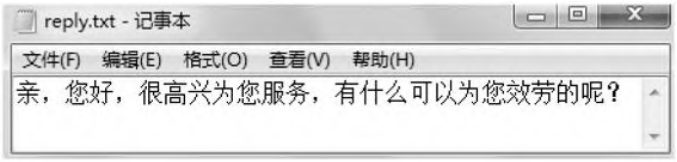


图14.26　复制之后reply.txt文件内容

综合练习4：读取《生僻字》歌词　编写程序，读取歌曲《生僻字》的一句歌词。编写程序前，先创建一个songs.txt文件。运行效果如下：

```
请输入文件路径:
F:\songs.txt

^_^_^_^_^_^_^_^_^_^_^_^
 茕茕孑立 沆瀣一气
^_^_^_^_^_^_^_^_^_^_^_^
```

综合练习5：打印对联　将一副对联存在文件antithetical.txt中，编写程序将它打印出来。运行结果如下：

```
上联：为系统而生，为框架而死，为debug奋斗一辈子！
下联：吃符号的亏，上大小写的当，最后死在需求上！
横批：杯具程序员。
```

综合练习6：输出附近米线店信息　附近有3家米线，用程序将米线店的店名、地址、电话等信息显示出来（提示：需要创建一个空的eat.txt）。运行结果如下：

```
附近一共有多少家米线店?
3
请输入文件路径:
F:\eat.txt
请输入米线店店铺名、地址、电话号:
第 1 家
状元米线
硅谷大街
845695235
第 2 家
三品阁米线
红旗街
85695412
第 3 家
云南过桥米线
重庆路
841256398
        状元米线            硅谷大街        845695235
        三品阁米线          红旗街          856954125
        云南过桥米线        重庆路          841256398
```

综合练习7：查询校园卡密码　很多大学会给新生办理校园卡，以方便学生在校园内消费。设置的默认密码通常是学生身份证号的后6位。创建一个空的idcard.txt，利用fseek函数定位默认密码，并输出。运行结果如下：

```
请输入文件路径:
F:\idcard.txt
请输入身份证号:
220*******11112367
```

```
默认密码为（身份证后6位）:112367
```

综合练习8：打印元音字母　将英语元音字母保存在文件word.txt中，每次使用时需要打开，并用程序来读取此文件。运行结果如下：

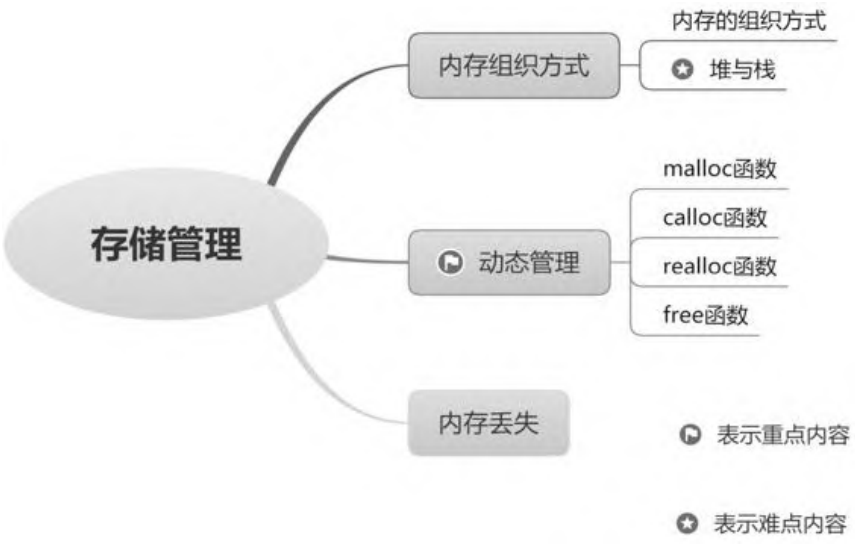
```
请输入文件路径:
F:\word.txt
第1个是:    a
第2个是:    e
第3个是:    i
第4个是:    o
第5个是:    u
```

第15章
存储管理

程序运行时，会将各类数据分类组织存放在内存空间中，并常常需要动态地分配和撤销内存空间。例如，对动态链表中的节点进行插入和删除，就要对内存进行管理。

本章致力于使读者了解内存的组织结构，了解堆和栈的区别，掌握常用的动态管理内存函数，了解内存在什么情况下会丢失。

本章的知识架构及重难点如下：



15.1 内存组织方式

计算机程序的机器语言指令和数据都存储在同一个逻辑内存空间里。那么这些内存是按照怎样的方式组织的呢？下面来进行具体的介绍。

15.1.1 数据的存放方式



程序编写完成后，需要先装载到计算机的内核或者半导体内存中，然后才能运行。计算机程序一般被组织成4个逻辑段：可执行代码、静态数据、动态数据（堆）和栈。

- ☑ 可执行代码和静态数据存储在固定的内存位置。
- ☑ 动态数据需要系统动态分配内存，一般存放在堆区的内存池中。
- ☑ 局部数据对象、函数参数，以及调用函数和被调用函数的联系，存放在栈区的内存池中。

15.1.2 堆与栈



下面来详细介绍堆、栈的定义和用法。

1．堆

内存的全局存储空间中，程序可动态分配和释放的内存块称为自由存储空间，也称为堆。

C程序中，用malloc和free函数来从堆中动态地分配和释放内存。

【例15.1】下载《绝地生存》需要多大空间（**实例位置：资源包\TM\sl\15\01**）

《绝地生存》（俗称“吃鸡”游戏）是一款刺激、好玩的枪战游戏。想要下载这个游戏，需要约20GB空间。编写程序，模拟游戏下载场景，

提示需要占用多少空间。

使用malloc函数分配一个整型变量的内存空间，在使用完该空间后，使用free函数进行释放。

```
#include <stdlib.h>
#include<stdio.h>
int main()
{
    int *pInt;                                  /*定义指针*/
    pInt=(int*)malloc(sizeof(int));             /*分配内存*/
    *pInt=20;                                   /*使用分配的内存*/
    printf("《绝地生存》占%dG 空间\n",*pInt);    /*输出信息*/
    free(pInt);                                 /*释放内存*/
    return 0;
}
```

程序中使用malloc函数分配了一个整型变量的内存空间。运行程序，显示效果如图15.1所示。

2．栈

程序不会像处理堆那样，在栈中显式地分配内存。当程序调用函数和声明局部变量时，系统将自动分配内存。

栈是一个后进先出的“压入弹出式”数据结构。程序运行时，每向栈中压入一个对象，栈指针就会向下移动一个位置。当系统从栈中弹出一个对象时，最晚进栈的对象将被先弹出，然后栈指针向上移动一个位置。如果栈指针位于栈顶，则表示栈是空的；如果栈指针指向最下方数据项的后一个位置，则表示栈为满的。入栈、出栈的操作如图15.2所示。

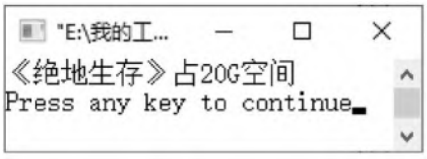


图15.1　在堆中分配内存并释放

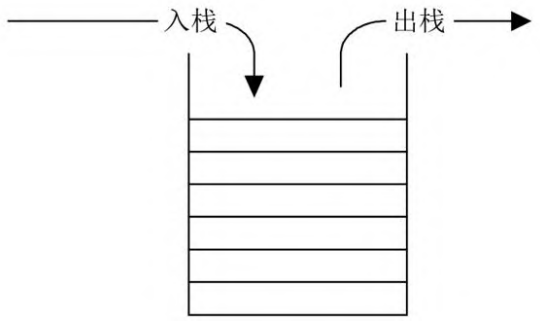


图15.2　栈操作

程序员经常利用栈处理那些适合用“后进先出”逻辑描述的编程问题。栈不需要程序员编写代码去维护，而是运行时由系统自动处理。系统自动维护和后进先出是栈区别于堆的显著标志。

栈是如何工作的呢？通过函数调用来看一下栈的整体操作过程。

当函数A调用函数B时，系统将会把函数A的所有实参和返回地址压入栈中，栈指针将移到合适的位置来容纳这些数据。最后进栈的是函数A的返回地址。

当函数B开始执行后，系统把函数B的自变量压入栈中，并把栈指针再向下移，以保证有足够的空间来存储函数B声明的所有自变量。

当函数A的实参压入栈后，函数B就在栈中以自变量形式建立了形参。函数B内部的其他自变量也是存放在栈中的。由于这些进栈操作，栈指针已经移到了所有局部变量之下。但是函数B记录了刚开始执行时的初始栈指针，以这个指针为参考，用正偏移量或负偏移量来访问栈中的变量。

当函数B准备返回时，系统会弹出栈中的所有自变量，这时栈指针移到了函数B刚开始执行时的位置。接着，函数B返回，系统从栈中弹出返回地址，函数A就可以继续执行了。

当函数A继续执行时，系统还能从栈中弹出调用者的实参，于是栈指针又回到了调用发生前的位置。

【例15.2】函数调用中的栈操作（**实例位置：资源包\TM\sl\15\02**）

本实例中，对上面描述的栈的操作过程进行说明。其中，函数名称根据上面描述所确定。

```
#include<stdio.h>
void DisplayB(char* string)                    /*定义函数 B*/
{
    printf("%s\n",string);
}

void DisplayA(char* string)                    /*定义函数 A*/
{
    char String[20]="MingRi!";
    printf("%s\n",string);
    DisplayB(String);                          /*在 A 中调用函数 B*/
}

int main()
{
    char String[20]="LoveChina!";
    DisplayA(String);                          /*在主函数中调用函数 A，传递一个字符串*/
    return 0;
}
```

程序中定义了两个函数DisplayA和DisplayB，并在DisplayA中调用DisplayB。根据栈的原理移动栈中的指针，进而存储数据。

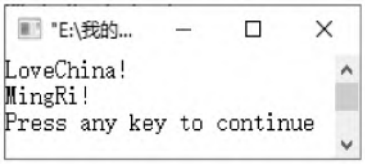


图15.3　栈在函数调用时的操作

运行程序，显示效果如图15.3所示。

编程训练（答案位置：资源包\TM\sl\15\编程训练\）

训练1：动态分配学生数据　定义一个学生结构体，动态分配一块结构体存储区域，然后进行赋值并输出，运行结果如下：

```
Number=102
Name=狗蛋
Sex=M
Score=62.50
```

训练2：用栈及递归计算分段函数　已知如下分段函数，试编写计算$f_n(x)$值的递归算法。

$$f_n(x)=\begin{cases}1 & \text{当 } n=0 \text{ 时}\\ 2x & \text{当 } n=1 \text{ 时}\\ 2x\,f_{n-1}(x)-2(n-1)\,f_{n-2}(x) & \text{当 } n>1 \text{ 时}\end{cases}$$

运行结果如下：

```
请输入n：
4
请输入x：
3
用递归算法得出的函数值是：876.00
用栈方法得出的函数值是：876.00
```

15.2 动态管理

15.2.1 malloc函数



malloc函数的功能是在内存中动态地分配一块size大小的内存空间。其原型如下：

```
void *malloc(unsigned int size);
```

使用malloc函数，要包含stdlib.h头文件。malloc函数会返回一个指针，指向分配的内存空间。如果分配出现错误，则返回NULL。

注意

使用malloc函数分配的内存空间位于堆中，而不是栈中。因此，在使用完这块内存之后，一定要将其释放掉，释放内存空间使用的是free函数。

例如，使用malloc函数分配一个整型内存空间：

```
int *pInt;                              /*定义指针*/
pInt=(int*)malloc(sizeof(int));         /*指针指向分配的 int 型空间*/
```

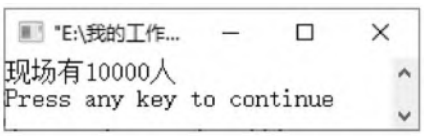
首先定义指针pInt，用来保存分配的内存地址。在使用malloc函数分配内存空间时，需要指定内存空间的大小（size），这里调用sizeof函数就可以得到指定类型的大小。malloc函数成功分配内存空间后会返回一个指针，因为分配的是一个int型空间，所以返回指针时也应该使用对应的int型指针，这样就要进行强制类型转换。最后将函数返回的指针赋值给指针pInt，就可以保存动态分配的整型空间地址了。

【例15.3】使用动态分配空间存储演唱会人数（实例位置：资源包\TM\sl\15\03）

某明星的演唱会，会馆可以容纳一万人。申请内存，存储人数信息，并将人数输出。

```
#include <stdlib.h>
#include<stdio.h>
int main()
{
    int* iIntMalloc;
    iIntMalloc=(int*)malloc(sizeof(int));          /*分配一个整型内存空间*/
    *iIntMalloc=10000;                             /*使用该空间保存人数信息*/
    printf("现场有%d 人\n",*iIntMalloc);           /*输出数据*/
    return 0;
}
```

程序中首先使用malloc函数分配了一个int型内存空间，用指针iIntMalloc指向它，然后使用该空间保存演唱会人数信息，最后显示该数据。



运行程序，显示效果如图15.4所示。

图15.4　使用malloc函数动态分配空间

误区警示

malloc函数不能忽略返回值。C语言规定，如果所申请的内存分配不成功，malloc函数的返回值为null pointer，也就是NULL。在这种情况下，如果继续运行之后的代码，程序就会产生崩溃。所以在申请分配内存后，应及时检查内存分配是否成功。

15.2.2　calloc函数



calloc函数的功能是在内存中动态分配n个长度为size的连续内存空间。其原型如下：

```
void * calloc(unsigned n, unsigned size);
```

使用calloc函数，也要包含头文件stdlib.h。calloc函数会返回一个指针，该指针指向动态分配的连续内存空间的首地址。当空间分配错误时，返回NULL。

例如，使用calloc函数分配一个整型数组内存空间：

```
int* pArray;                                /*定义指针*/
pArray=(int*)calloc(3,sizeof(int));         /*指针指向分配的数组内存*/
```

上面的代码中，pArray为一个整型指针，使用calloc函数分配3个整型内存空间给一个数组，3表示分配数组中元素的个数，int表示元素的类

型。最后将返回的指针赋予pArray指针变量，pArray指向的就是该数组的首地址。

【例15.4】动态分配数组内存（实例位置：资源包\TM\sl\15\04）

在本实例中，动态分配一个指针。使用strcpy函数为字符数组赋值，再进行输出，以验证分配的内存是否正确保存了数据。具体代码如下：

```
#include<stdlib.h>                              /*包含头文件*/
#include<stdio.h>
#include<string.h>
int main()
{
    char* ch;                                   /*定义指针*/
    ch=(char*)calloc(30,sizeof(char));          /*分配一段字符数组内存*/
    strcpy(ch,"清华大学出版社");                 /*复制字符串*/
    printf("%s\n",ch);                          /*输出字符串*/
    free(ch);                                   /*释放空间*/
    return 0;                                   /*程序结束*/
}
```

运行程序，显示效果如图15.5所示。

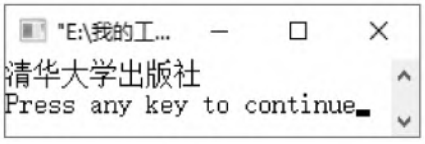


图15.5　使用calloc函数分配数组内存

15.2.3　realloc函数



realloc函数的功能是将ptr指针指向的内存空间大小改为size。其原型如下：

```
void *realloc(void *ptr, size_t size);
```

使用realloc函数前，要先包含头文件stdlib.h。size表示新分配的内存空间的大小，其值可以是任意的，既可以比原来的数值大，也可以比原来的数值小。返回值是一个指向新地址的指针，如果出现错误，则返回NULL。

例如，将分配的实型空间大小改为整型大小，代码如下：

```
fDouble=(double*)malloc(sizeof(double));          /*定义实型指针，分配一个实型空间*/
iInt=realloc(fDouble,sizeof(int));                /*定义整型指针，将实型空间改为整形空间*/
```

其中，fDouble指向分配的实型空间，之后使用realloc函数改变fDouble指向的空间大小，将其大小设置为整型，然后将改变后的内存空间地址返回，赋值给iInt整型指针。

【例15.5】重新分配内存（**实例位置：资源包\TM\sl\15\05**）

本实例中，定义了一个整型指针和实型指针，利用realloc函数重新分配内存，具体代码如下：

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int *iInt;                                    /*定义整型指针*/
    char* cChar;                                  /*定义字符型指针*/
    iInt=(int*)malloc(sizeof(int));               /*使用 malloc 分配整型空间*/
    printf("整型数据内存是：%d\n",sizeof(*iInt));    /*输出空间的大小*/
    cChar =realloc(fDouble,sizeof(char));         /*使用 realloc 改变分配的空间大小*/
    printf("字符数据内存是：%d\n",sizeof(*cChar));   /*输出空间的大小*/
    return 0;                                     /*程序结束*/
}
```

在本实例中，首先使用malloc函数分配了一个整型大小的内存空间，然后通过sizeof函数输出该内存空间的大小，最后使用realloc函数得到新的内存空间大小，并输出新空间的大小。比较两者的数值，可以看出新空间与原来的空间大小不一样。

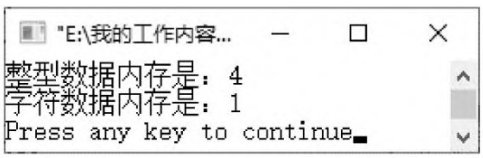


图15.6　使用realloc函数重新分配内存

运行程序，显示效果如图15.6所示。

说明

函数malloc、calloc和realloc都可以动态分配空间，它们之间的区别在于：malloc函数分配一块内存空间，且不进行初始化；calloc函数分配连续多块大小相同的内存空间，且会将其初始化为零；realloc函数主要用于调整内存空间的大小。

15.2.4 free函数



free函数的功能是释放指针ptr指向的内存区域，使该内存区域能被其他变量所使用。其原型如下：

```
void free(void *ptr);
```

其中，ptr是最近一次调用calloc或malloc函数时返回的值。free函数无返回值。

例如，释放一个分配给整型变量的内存空间：

```
free(pInt);
```

代码中，pInt为一个指向整型大小的内存空间，使用free函数将其进行释放。

【例15.6】释放内存空间（实例位置：资源包\TM\sl\15\06）

在本实例中，将分配的内存进行释放，释放前先输出一次内存中保存的数据，释放后再利用指针输出一次。观察两次的结果，可以看出，调用free函数之后内存被释放了。

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int* pInt;                              /*定义整型指针*/
    pInt=(int*)malloc(sizeof(pInt));        /*分配整型空间*/
    *pInt=100;                              /*赋值*/
    printf("分配的值是:%d\n",*pInt);        /*将值进行输出*/
    free(pInt);                             /*释放该内存空间*/
    printf("释放完内存的值是:%d\n",*pInt);  /*将值进行输出*/
    return 0;
}
```

在程序中，定义指针pInt指向动态分配的内存空间，保存数据，之后利用指针输出该空间存储的内容。调用free函数将该空间释放，当再次输出时因为保存数据的空间已经被释放，所以数据肯定就不存在了。



图15.7　使用free函数释放内存空间

运行程序，显示效果如图15.7所示。

编程训练（答案位置：资源包\TM\sl\15\编程训练\）

训练3：为数组分配内存　为一个包含3个元素的数组动态分配内存，为元素赋值并将其输出。运行结果如下：

```
数组值为：10
数组值为：20
数组值为：30
```

训练4：商品信息动态分配　动态分配一块内存区域，存放某商品信息。运行结果如下：

```
编号=1001
```

```
名称=苹果
数量=100
价格=2.10
```

15.3 内存泄露与内存丢失



1．内存泄露

使用malloc等函数分配过内存后，还需要使用free函数及时释放内存。如果不进行释放，就会造成内存泄漏，甚至会导致系统崩溃。

free函数可以实时地执行回收内存操作。如果程序很简单，程序结束之前不会使用过多的内存，不会降低系统的性能，可以不用写free函数。程序结束后，操作系统会自动完成释放的功能。

但是在开发大型程序时，如果不及时通过free函数释放内存，后果是很严重的。例如，程序中可能要重复一万次分配10MB的内存，如果每次分配内存后都使用free函数及时释放用完的内存空间，那么整个程序只需要使用10MB内存就可以运行。如果不使用free函数释放用完的内存空间，整个程序就要使用100GB的内存！这其中包括绝大部分的虚拟内存，由于虚拟内存的操作需要读写磁盘，所以会极大地影响到系统的性能，甚至导致系统崩溃。

因此，实际开发中，使用malloc函数分配内存后，应及时地写出一个free函数释放内存。这是一个良好的编程习惯，不但在处理大型程序时非常有必要，也在一定程度上体现了程序的优美和健壮性。

2．内存丢失

下面来看一个内存丢失的例子。例如：

```
pOld=(int*)malloc(sizeof(int));
pNew=(int*)malloc(sizeof(int));
```

这两段代码创建了两块内存，并且将内存地址分别赋给了指针pOld和pNew，此时指针pOld和pNew分别指向两块内存。如果接下来进行这样的操作：

```
pOld=pNew;
```

则pOld指针就指向了pNew指向的内存地址。这时再进行释放内存操作：

```
free(pOld);
```

由于pOld所指向的内存空间是原来pNew指向的，于是这块空间被释放了。但是pOld原来指向的那块内存空间还没有被释放（因为没有指针指向这块内存），这块内存就造成了丢失。

15.4 实践与练习

（答案位置：资源包\TM\sl\15\实践与练习\）

综合练习1：下载明日学院App需要多大内存　明日学院是明日科技为手机端客户搭建的App应用程序。模拟下载场景，编写程序显示其需要占用多少内存空间。（提示：明日学院App占31.4MB内存，效果如下：

```
→*→*→*→*→*→*→*→*→*→*→*→*
  明日学院占 31.4MB 空间
→*→*→*→*→*→*→*→*→*→*→*→*
```

综合练习2：申请内存，输出10个0　利用calloc函数申请内存，输出10个0，看看能得出什么结论。效果如下：

```
----------
|   0  |
|   0  |
|   0  |
|   0  |
|   0  |
|   0  |
|   0  |
|   0  |
|   0  |
|   0  |
----------
```

综合练习3：申请内存，存放观众席人数　某年的足球世界杯正在进行，观众可谓是人山人海，但是观众席只能容纳91000人。编写程序，申请内存，存储观众席人数信息，然后将人数输出。效果如下：

```
≈≈≈≈≈≈≈≈≈≈≈≈≈≈
     可容纳 91000 人
≈≈≈≈≈≈≈≈≈≈≈≈≈≈
```

综合练习4：接受用户信息　写一个函数，该函数可以接受用户输入的字符并存储在内存中（由于不确定用户会输入几个字符，所以这些内存不可以用数组来表示）。当用户输入字符'q'时，输出用户输入的所有字符，并退出程序。效果如下：

```
请用户输入：Fine
q
 第 0 个字母是 F
 第 1 个字母是 i
 第 2 个字母是 n
 第 3 个字母是 e
```

综合练习5：库房还有多少货　某服装店进了10240件衣服，为了将这批衣服顺利地入库，老板将库房收拾了，以存放这批衣服。使用malloc函数申请内存空间，将这批衣服件数输出显示。效果如下：

```
%-%-%-%-%-%-%-%-
  衣服有 10240 件
%-%-%-%-%-%-%-%-
```

综合练习6：申请内存，写入Mingri　动态分配一个数组，使用strcpy函数为字符数组赋值，再进行输出，验证分配的内存已正确保存数

据。效果如下：

```
$-$-$-$-$-$-$-$-$-$
    Mingrisoft
$-$-$-$-$-$-$-$-$-$
```

综合练习7：合理分配空间　定义char型数据，分别用malloc、realloc函数为其分配空间并输出空间值，效果如下：

```
№ № № № № № № № № №
  malloc 分配内存：7f2db0
  realloc 分配内存：7f0e28
№ № № № № № № № № №
```

综合练习8：为二维数组释放内存　编写程序，先为二维数组动态分配内存，再释放对应内存空间。效果如下：

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

第16章
网络套接字编程

网络程序有多种开发方式，Windows Socket就是其中比较简单的一种实现方法。

通过本章，读者可详细了解有关计算机网络的基础知识，包括IP地址、OSI七层参考模型、地址解析、域名系统、TCP/IP协议和端口；熟悉套接字的有关内容，并能使用套接字编写网络聊天程序。

本章的知识架构及重难点如下：



16.1 计算机网络基础

计算机网络技术是计算机技术和通信技术相结合的产物，代表着计算机的一个重要发展方向。了解计算机的网络结构，有助于用户开发网络应

用程序。本节将介绍有关计算机网络的基础知识和基本概念。

16.1.1 IP地址



为了使网络上的计算机能够彼此识别对方，每台计算机都被赋予了一个独一无二的IP地址。IP地址是由IP协议规定的32位的二进制数表示，最新的IPv6协议已将IP地址升级为128位，这使得IP地址范围更加广泛，未来能很好地缓解IP地址紧缺的压力。目前，IPv6协议距离实际应用还有一段距离，多数操作系统和应用软件仍然以32位的IP地址为基准。

32位的IP地址主要分为前缀和后缀两部分。前缀表示计算机所属的物理网络，后缀用于确定该网络上的唯一一台计算机。在互联网上，每一个物理网络都有唯一的网络号，根据网络号的不同，可以将IP地址分为5类，即A类、B类、C类、D类和E类。其中，A类、B类和C类属于基本类，D类用于多播发送，E类属于保留类。表16.1描述了各类IP地址的范围。

表16.1 各类IP地址的范围

类 型	范 围	类 型	范 围
A 类	0.0.0.0～127.255.255.255	D 类	224.0.0.0～239.255.255.255
B 类	128.0.0.0～191.255.255.255	E 类	240.0.0.0～247.255.255.255
C 类	192.0.0.0～223.255.255.255		

需要注意，有几个IP地址是特殊的，它们有单独的用途。

☑ 网络地址：主机地址为0的表示网络地址，如128.111.0.0。

☑ 广播地址：在网络号后跟所有位全是1的IP地址，表示广播地址。

☑ 回送地址：127.0.0.1表示回送地址，用于测试。

16.1.2 OSI七层参考模型



开放系统互联（Open System Interconnection，OSI）是国际标准化组织（ISO）为了实现计算机网络的标准化而颁布的参考模型。OSI参考模型将网络中的数据传输划分为7层，每一层使用下层的服务，并向上层提供服务。表16.2描述了OSI参考模型的结构。

表16.2 OSI参考模型

层　次	名　称	功 能 描 述
第 7 层	应用层（Application）	负责网络中应用程序与网络操作系统之间的联系，例如，建立和结束使用者之间的连接，管理建立相互连接使用的应用资源
第 6 层	表示层（Presentation）	用于确定数据交换的格式，解决应用程序之间在数据格式上的差异，并负责设备之间需要的字符集和数据转换
第 5 层	会话层（Session）	用户应用程序与网络层的接口，用于建立与其他设备的连接（即会话），并对会话进行有效的管理
第 4 层	传输层（Transport）	提供会话层和网络层之间的传输服务，该服务从会话层获得数据，必要时对数据进行分割，然后将数据正确无误地传送到网络层
第 3 层	网络层（Network）	将传输的数据封包，然后通过路由选择、分段组合等控制，将信息从源设备传送到目标设备
第 2 层	数据链路层（Data Link）	修正传输过程中的错误信号，能够提供可靠的通过物理介质传输数据的方法
第 1 层	物理层（Physical）	利用传输介质为数据链路层提供物理连接，本层规范了网络硬件的特性、规格和传输速度

OSI参考模型的建立，不仅创建了通信设备之间的物理通道，还规划了各层之间的功能，为标准化组合和生产厂家制定协议提供了基本依据，有助于用户了解复杂的协议，如TCP/IP、X.25协议等。用户可以将这些协议与OSI参考模型对比，从而了解协议的工作原理。

16.1.3 地址解析

所谓地址解析，是指将计算机的协议地址解析为物理地址，即MAC（Medium Access Control）地址，又称为媒体访问控制地址。通常，在网络上由地址解析协议（ARP）来实现地址解析。下面以本地网络上的两台计算机通信为例，介绍ARP协议解析地址的过程。

假设主机A和主机B处于同一个物理网络上，主机A的IP地址为192.168.1.21，主机B的IP地址为192.168.1.23，当主机A与主机B通信时，主机B的IP地址192.168.1.23将按如下步骤解析为物理地址。

（1）主机A从本地ARP缓存中查找IP地址192.168.1.23对应的物理地址。用户可以在命令行窗口中输入“arp -a”命令查看本地ARP缓存，如图16.1所示。



图16.1　本地ARP缓存

（2）如果主机A在ARP缓存中没有发现192.168.1.23映射的物理地址，将发送ARP请求帧到本地网络上的所有主机。在ARP请求帧中包含了主机A的物理地址和IP地址。

（3）本地网络上的其他主机接收到ARP请求帧后，检查是否与自己的IP地址匹配，如果不匹配，则丢弃ARP请求帧。如果主机B发现与自己的IP地址匹配，则将主机A的物理地址和IP地址添加到自己的ARP缓存中，然后主机B将自己的物理地址和IP地址发送给主机A，当主机A接收到主机B发来的信息后，将用这些信息更新ARP缓存。

（4）主机B的物理地址确定后，主机A就可以与主机B通信了。

16.1.4 域名解析



虽然使用IP地址可以标识网络中的任意一台计算机，但由于IP地址容易混淆，并且不容易记忆，人们更倾向于使用主机名来标识IP地址。在Internet上存在许多计算机，为了防止主机名相同，Internet管理机构采取在主机名后加后缀名的方法标识一台主机，该后缀名被称为域名。例如，www.mingrisoft.com，主机名为www，域名为mingrisoft.com。这里的域名为二级域名，其中com为一级域名，表示商业组织，mingrisoft为本地域名。为了能够利用域名进行不同主机间的通信，需要将域名解析为IP地址，称之为域名解析。域名解析是通过域名服务器来完成的。

假如主机A的本地域名服务器是dns.local.com，根域名服务器是dns.mr.com；所要访问的主机B的域名为www.mingribook.com，域名服务器为dns.mrbook.com。

（1）当主机A通过域名www.mingribook.com访问主机B时，将发送解析域名www.mingribook.com的报文，本地域名服务器收到请求后，首先查询本地缓存，如果没有该记录，则向根域名服务器dns.mr.com发出请求，要求解析域名www.mingribook.com。

（2）根域名服务器dns.mr.com收到请求后查询本地记录，如果发现 mingribook.com NS dns.mrbook.com 信息，将给出dns.mrbook.com的IP地址，并将结果返回给主机A的本地域名服务器dns.local.com.

（3）当本地域名服务器dns.local.com收到信息后，会向主机B的域名服务器dns.mrbook.com发送解析域名www.mingribook.com的报文。

（4）域名服务器dns.mrbook.com收到请求后，开始查询本地的记录，发现www.mingribook.com A 211.120.X.X类似的信息，将结果返回给主机A的本地域名服务器dns.local.com，其中211.120.X.X表示域名www.mingribook.com的IP地址。

16.1.5 TCP/IP协议



TCP/IP（Transmission Control Protocal/Internet Protocal，传输控制协议／网际协议）是互联网上最流行的协议，它能够实现互联网上不同类型操作系统的计算机相互通信。对于网络开发人员，必须了解TCP/IP协议的结构。TCP/IP协议将网络分为4层，分别对应于OSI参考模型的7层结构。表16.3列出了TCP/IP协议与OSI参考模型的对应关系。

表16.3 TCP/IP协议结构层次

TCP/IP 协议	OSI 参考模型
应用层（包括 Telnet、FTP、SNTP 协议）	会话层、表示层和应用层
传输层（包括 TCP、UDP 协议）	传输层
网络层（包括 ICMP、IP、ARP 等协议）	网络层
数据链路层	物理层和数据链路层

由表16.3可以发现，TCP/IP不是单个协议，而是一个协议簇，包含多种协议，其中最主要的协议是网际协议（IP）和传输控制协议（TCP）。下面给出TCP/IP中一些主要的协议结构。

1．TCP协议

传输控制协议（TCP）是一种提供可靠数据传输的通用协议，它是TCP/IP体系结构中传输层上的协议。在发送数据时，应用层的数据传输到传输层，加上TCP的首部，数据就构成了报文。报文是网际层IP的数据，如果再加上IP首部，就构成了IP数据报。TCP协议的C语言数据描述如下：

```
typedef struct HeadTCP
{
    WORD    SourcePort;        /*16 位源端口号*/
    WORD    DePort;            /*16 位目的端口号*/
    DWORD   SequenceNo;        /*32 位序号*/
    DWORD   ConfirmNo;         /*32 位确认序号*/
    BYTE    HeadLen;           /*与 Flag 为一个组成部分，首部长 4 位，保留 6 位，6 位标识，共 16 位*
    BYTE    Flag;
    WORD    WndSize;           /*16 位窗口大小*/
    WORD    CheckSum;          /*16 位校验和*/
    WORD    UrgPtr;            /*16 位紧急指针*/
} HEADTCP;
```

2．IP协议

IP协议又称为网际协议，它工作在网络层，主要提供无链接数据报传输。IP协议不保证数据报的发送，但可以最大限度地发送数据。IP协议的C语言数据描述如下：

```
typedef struct HeadIP
{
    unsigned char headerlen:4;     /*首部长度，占 4 位*/
    unsigned char version:4;       /*版本，占 4 位 */
    unsigned char servertype;      /*服务类型，占 8 位，即 1 个字节*/
    unsigned short totallen;       /*总长度，占 16 位*/
    unsigned short id;             /*与 idoff 构成标识，共占 16 位，前 3 位是标识，后 13 位是片偏移*/
    unsigned short idoff;
    unsigned char ttl;             /*生存时间，占 8 位*/
    unsigned char proto;           /*协议，占 8 位*/
    unsigned short checksum;       /*首部检验和，占 16 位*/
    unsigned int sourceIP;         /*源 IP 地址，占 32 位*/
    unsigned int destIP;           /*目的 IP 地址，占 32 位*/
}HEADIP;
```

3．ICMP协议

ICMP协议又称为网际控制报文协议。它负责网络上设备状态的发送和报文检查，可以将某个设备的故障信息发送到其他设备上。ICMP协议的C语言数据描述如下：

```
typedef struct HeadICMP
{
    BYTE Type;              /*8 位类型*/
    BYTE Code;              /*8 位代码*/
    WORD ChkSum;            /*16 位校验和*/
} HEADICMP;
```

4．UDP协议

用户数据报协议（UDP）是一个面向无连接的协议，采用该协议，两个应用程序不需要先建立连接。它为应用程序提供一次性的数据传输服务。UDP协议不提供差错恢复，不能提供数据重传，因此该协议传输数据安全性略差。UDP协议的C语言数据描述如下：

```
typedef struct HeadUDP
{
    WORD SourcePort;        /*16 位源端口号*/
    WORD DePort;            /*16 位目的端口*/
    WORD Len;               /*16 为 UDP 长度*/
    WORD ChkSum;            /*16 位 UDP 校验和*/
} HEADUDP;
```

16.1.6　端口



虽然计算机可通过IP地址来标识自己，但两台计算机具体通信时，还会出现一个问题。假设主机A中的应用程序A1想与主机B中的应用程序B1通信，如何知道是主机A中的A1应用程序而不是其他应用程序与主机B中的应用程序通信呢？当主机B接收到数据时，它又该如何知道数据是发往应用程序B1（此时主机B中可能同时运行着多个应用程序）的呢？

为了解决上述问题，TCP/IP协议提出了“端口”的概念，用于标识具体通信的应用程序。当应用程序（严格说应该是进程）与某个端口绑定后，系统会将收到的给该端口的数据送往该应用程序。端口通常用一个16位的无符号整数值来表示，范围为0~65535，低于256的端口系统保留端口，用于系统进程的通信，其他端口则是自由端口，可以由进程自由使用。

16.1.7 套接字的引入



美国伯克利大学在UNIX上推出了一种应用程序访问通信协议的操作系统调用套接字（socket）。socket的出现，使得程序员可以很方便地访问TCP/IP，进而开发各种网络应用程序。后来，套接字被引入Windows操作系统，成为网络应用程序开发的有效工具。

套接字存在于通信区域中，通信区域也称为地址族，主要用于将通过套接字通信的进程的公有特性综合在一起。套接字通常只与同一区域的套接字交换数据。Windows Sockets只支持一个通信区域——AF_INET网际域，使用网际协议族通信的进程使用该域。

16.1.8 网络字节顺序



不同的计算机存放多字节值的顺序不同，有的机器在起始地址存放低位字节，有的机器在起始地址存放高位字节。例如，基于Intel CPU的PC

机采用低位先存的方式。由于不同的计算机存放数据字节的顺序不同，因此发送数据后再接收该数据时，有可能无法查看所接收到的数据。为了保证数据的正确性，在网络协议中需要指定网络字节顺序，TCP/IP协议使用16位整数和32位整数的高位先存格式。

16.2 套接字基础

套接字是网络通信的基本构件，最初是加利福尼亚大学伯克利分校为UNIX开发的网络通信编程接口。为了在Windows操作系统上使用套接字，20世纪90年代初，微软和第三方厂商共同制定了一套标准，即Windows Socket规范，简称WinSock。本节将介绍有关WinSock的相关知识。

16.2.1 套接字概述



所谓套接字，实际上是一个指向传输提供者的句柄。在WinSock中，就是通过操作该句柄来实现网络通信和管理的。根据性质和作用的不同，套接字可以分为原始套接字、流式套接字和数据包套接字3种。

- ☑ 原始套接字：是在WinSock 2规范中提出的，它能够使程序开发人员对底层的网络传输机制进行控制，在原始套接字下接收的数据中包含IP头。
- ☑ 流式套接字：提供双向、有序、可靠的数据传输服务。该类型套接字在通信前需要双方建立连接，大家熟悉的TCP协议采用的就是流式套接字。

☑ 数据包套接字：提供双向的数据流，但不能保证数据传输的可靠性、有序性和无重复性。UDP协议采用的就是数据包套接字。

16.2.2 TCP套接字的socket编程



TCP是面向连接的可靠的传输协议。利用TCP协议进行通信时，首先要建立通信双方的连接。一旦连接建立完成，就可以进行通信。TCP提供了数据确认和数据重传的机制，保证了发送的数据一定能到达通信的对方。

基于TCP，面向连接的socket编程的服务器端程序流程如下。

（1）创建套接字socket。

（2）将创建的套接字绑定（bind）到本地的地址和端口上。

（3）设置套接字的状态为监听状态（listen），准备接受客户端的连接请求。

（4）接受请求（accept），同时返回得到一个用于连接的新套接字。

（5）使用这个新套接字进行通信（通信函数使用send/recv）。

（6）通信完毕，释放套接字资源（closesocket）。

基于TCP，面向连接的socket编程的客户端程序流程如下。

（1）创建套接字socket。

（2）向服务器发出连接请求（connect）。

（3）连接后，与服务器进行通信操作（send/recv）。

（4）释放套接字资源（closesocket）。

在服务器端，当调用accept函数时（关于套接字函数后文将进行介绍），程序就会进行等待，直到有客户端调用connect函数发送连接请求，然后服务器接受该请求，这样服务器端与客户端就建立了连接，可以开始通信了。

注意

在服务器端要建立套接字绑定到指定的主机IP和端口上等待客户的请求，但是对于客户端来说，当发起连接请求并被接受后，在服务器端就保存了该客户端的IP地址和端口号的信息。对于服务器端来说，一旦建立连接，实际上它已经保存了客户端的IP地址和端口号的信息，因此可以利用返回的套接字进行与客户端的通信。

16.2.3 UDP套接字的socket编程



UDP是无连接的不可靠的传输协议。采用UDP进行通信时，不需要建立连接，可以直接向一个IP地址发送数据，但是不能保证对方能收到。

对于基于UDP，面向无连接的套接字编程来说，服务器端和客户端的概念不是特别的严格。可以把服务器称为接收端，客户端就是发送数据的发送端。

基于UDP，面向无连接的socket编程的接收端程序流程如下。

（1）创建套接字socket。

（2）将套接字绑定（bind）到一个本地地址和端口上。

（3）等待接收数据（recvfrom）。

（4）释放套接字资源（closesocket）。

基于UDP，面向无连接的socket编程的发送端程序流程如下。

（1）创建套接字socket。

（2）向服务器发送数据（sendto）。

（3）释放套接字资源（closesocket）。

注意

> 基于UDP的套接字编程中，仍然需要使用bind进行绑定。虽然面向无连接的socket编程无须建立连接，但为了完成通信，首先应启动接收端，等待接收发送端发来的数据，这样接收端就必须告诉它自己的地址和端口。因此，必须调用bind函数将套接字绑定到一个本地地址和端口上。

基于UDP的套接字编程时，利用的是sendto和recvfrom两个函数实现数据的发送和接收；基于TCP的套接字编程时，发送数据使用的是send函数，接收数据使用的是recv函数。

16.3 套接字函数

前面介绍了使用套接字编写程序的流程，本节介绍使用套接字编程时用到的函数。

16.3.1 套接字常用函数介绍



1．WSAStartup函数

WSAStartup函数的功能是初始化套接字库。其原型如下：

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

注意

WSAStartup函数用于初始化Ws2_32.dll动态链接库。在使用其他套接字函数之前，必须先初始化Ws2_32.dll动态链接库。

- ☑ wVersionRequested：表示调用者使用的WinSock版本，高字节记录修订版本，低字节记录主版本。例如，如果WinSock的版本为2.1，则高字节记录1，低字节记录2。
- ☑ lpWSAData：WSADATA结构指针，详细记录了Windows套接字的相关信息。其定义如下：

```
typedef struct WSAData {
    WORD            wVersion;
    WORD            wHighVersion;
    char            szDescription[WSADESCRIPTION_LEN+1];
    char            szSystemStatus[WSASYS_STATUS_LEN+1];
    unsigned short  iMaxSockets;
    unsigned short  iMaxUdpDg;
    char FAR *      lpVendorInfo;
} WSADATA, FAR * LPWSADATA;
```

  - ➢ wVersion：表示调用者使用的WS2_32.dll动态库的版本号。

- wHighVersion：表示WS2_32.dll支持的最高版本，通常与wVersion相同。
- szDescription：表示套接字的描述信息，通常没有实际意义。
- szSystemStatus：表示系统的配置或状态信息，通常没有实际意义。
- iMaxSockets：表示最多可以打开多少个套接字。在WinSock 2及以后的版本中，该成员将被忽略。
- iMaxUdpDg：表示数据报的最大长度。在WinSock 2及以后的版本中，该成员将被忽略。
- lpVendorInfo：表示套接字的厂商信息。在WinSock 2及以后的版本中，该成员将被忽略。

例如，使用WSAStartup初始化套接字，版本号为2.2：

```
WORD wVersionRequested;                          /*WORD（字），类型为 unsigned short*/
WSADATA wsaData;                                 /*库版本信息结构*/
/*定义版本类型，将两个字节组合成一个字，前面是低字节，后面是高字节*/
wVersionRequested = MAKEWORD(2, 2);              /*表示版本号*/
/*加载套接字库，初始化 Ws2_32.dll 动态链接库*/
WSAStartup(wVersionRequested, &wsaData);
```

从上面的代码中可以看出，MAKEWORD宏的作用是：根据给定的两个无符号字节，创建一个16位的无符号整型，将创建的值赋予wVersionRequested变量，表示套接字的版本号。

2．socket函数

socket函数的功能是创建一个套接字。其原型如下：

```
SOCKET socket(int af,int type, int protocol);
```

☑ af：表示一个地址家族，通常为AF_INET。

- ☑ type：表示套接字类型。如果为SOCK_STREAM，表示创建面向连接的流式套接字；如果为SOCK_DGRAM，表示创建面向无连接的数据报套接字；如果为SOCK_RAW，表示创建原始套接字。对于这些值，用户可以在winsock2.h头文件中找到。
- ☑ protocol：表示套接字采用的协议，如果用户不指定，可以设置为0。
- ☑ 返回值：创建的套接字句柄。

例如，使用socket函数创建一个套接字socket_server：

```
/*创建服务器套接字*/
/*AF_INET表示指定地址族，SOCK_STREAM表示流式套接字TCP，特定的地址家族相关的协议*/
socket_server=socket(AF_INET,SOCK_STREAM,0);
```

在上面代码中，如果socket函数调用成功，就会返回一个新的SOCKET数据类型的套接字描述符。使用定义好的套接字socket_server进行保存。

3．bind函数

bind函数的功能是将套接字绑定到指定的端口和地址上。其原型如下：

```
int bind(SOCKET s,const struct sockaddr FAR*  name,int
namelen);
```

- ☑ s：套接字标识。
- ☑ name：sockaddr结构指针，包含了要结合的地址和端口号。
- ☑ namelen：表示name缓冲区的长度。
- ☑ 返回值：如果函数执行成功，则返回值为0，否则为SOCKET_ERROR。

在创建了套接字后，应该将该套接字绑定到本地的某个地址和端口上，这时就需要使用bind函数了。例如，使用bind函数绑定一个套接字：

```
SOCKADDR_IN Server_add;                    /*服务器地址信息结构*/
Server_add.sin_family=AF_INET;             /*地址族，必须是 AF_INET，注意只有它不是网络字节顺序*/
Server_add.sin_addr.S_un.S_addr=htonl(INADDR_ANY);          /*主机地址*/
Server_add.sin_port=htons(5000);                            /*端口号*/
bind(socket_server,(SOCKADDR*)&Server_add,sizeof(SOCKADDR))  /*使用 bind 函数进行绑定*/
```

4．listen函数

listen函数的功能是将套接字设置为监听模式。对于流式套接字，必须处于监听模式才能够接收客户端套接字的连接。该函数的原型如下：

```
int listen(SOCKET s, int backlog);
```

- ☑ s：套接字标识。
- ☑ backlog：表示等待连接的最大队列长度。例如，如果backlog被设置为2，此时有3个客户端同时发出连接请求，那么前两个客户端连接会放置在等待队列中，第3个客户端会得到错误信息。

例如，使用listen函数设置套接字为监听状态：

```
listen(socket_server,5);
```

上述代码中，设置套接字为监听状态，为连接做准备，最大等待的数目为5。

5．accept函数

accept函数的功能是接受客户端的连接。在流式套接字中，只有当套接字处于监听状态时，才能接受客户端的连接。该函数的原型如下：

```
SOCKET accept(SOCKET s, struct sockaddr FAR* addr, int FAR*
addrlen);
```

- ☑ s：套接字标识，它应处于监听状态。
- ☑ addr：sockaddr_in结构指针，包含一组客户端的端口号、IP地址等信息。
- ☑ addrlen：用于接收参数addr的长度。
- ☑ 返回值：一个新的套接字，对应于已经接受的客户端连接。对于该客户端的所有后续操作，都应使用这个新的套接字。

例如，使用accept函数接受客户端的连接请求，代码如下：

```
/*接受客户端的发送请求，等待客户端发送connect请求*/
socket_receive=accept(socket_server,
(SOCKADDR*)&Client_add,&Length);
```

其中，socket_receive用于保存接受请求后返回的新的套接字，socket_server为绑定在地址和端口上的套接字，而Client_add是有关客户端的IP地址和端口的信息结构，最后的Length是Client_add的大小（可以使用sizeof函数取得，然后用Length变量保存）。

6．closesocket函数

closesocket函数的功能是关闭套接字。其原型如下：

```
int closesocket(SOCKET s);
```

其中，s是套接字标识。如果参数s设置了SO_DONTLINGER选项，则调用该函数后会立即返回。此时如果有数据尚未传送完毕，则会继续传递数据，然后再关闭套接字。

例如，使用closesocket函数关闭套接字，释放客户端的套接字资源，代码如下：

```
closesocket(socket_receive);                    /*释放客户端的套接字资源*/
```

在代码中，socket_receive是一个套接字，当不使用时就可以利用closesocket函数将其资源释放。

7．connect函数

connect函数的功能是发送一个连接请求。其原型如下：

```
int connect(SOCKET s,const struct sockaddr FAR*  name,int
namelen);
```

- ☑ s：套接字标识。
- ☑ name：表示套接字s要连接的主机地址和端口号。
- ☑ namelen：表示name缓冲区的长度。
- ☑ 返回值：如果函数执行成功，则返回值为0，否则为SOCKET_ERROR。用户可以通过WSAGETLASTERROR得到其错误描述。

例如，使用connect函数与一个套接字建立连接，代码如下：

```
connect(socket_send,
(SOCKADDR*)&Server_add,sizeof(SOCKADDR));
```

在上面代码中，socket_send表示要与服务器建立连接的套接字，而Server_add是要连接的服务器地址信息。

8．htons函数

htons函数的功能是将一个16位的无符号短整型数据从主机排列方式转换为网络排列方式。其原型如下：

```
u_short htons(u_short hostshort);
```

☑ hostshort：主机排列方式的16位无符号短整型数据。

☑ 返回值：网络排列方式的16位无符号短整型数据。

例如，使用htons函数对一个无符号短整型数据进行转换，代码如下：

```
Server_add.sin_port=htons(5000);
```

在上面代码中，Sever_add是有关主机地址和端口的结构，其中sin_port表示的是端口号。因为端口号要使用网络排列方式，所以使用htons函数进行转换，设定新的端口号。

9．htonl函数

htonl函数的功能是将一个无符号长整型数据从主机排列方式转换为网络排列方式。其原型如下：

```
u_long htonl(u_long hostlong);
```

☑ hostlong：主机排列方式的32位无符号长整型数据。

☑ 返回值：网络排列方式的32位无符号长整型数据。

其使用方式与htons函数相似，不过是将一个32位数值转换为TCP/IP网络字节顺序。

10．inet_addr函数

inet_addr函数的功能是将一个由点分十进制表示的IP地址字符串转换为32位的无符号长整型数据。其原型如下：

```
unsigned long inet_addr(const char FAR * cp);
```

☑ cp：表示IP地址的字符串。

☑ 返回值：32位无符号长整型数据。

例如，使用inet_addr函数将一个以点分十进制格式表示的IP地址192.168.1.43转换为32位的无符号长整型数据，代码如下：

```
Server_add.sin_addr.S_un.S_addr = inet_addr("192.168.1.43");
```

11．recv函数

recv函数的功能是从面向连接的套接字中接收数据。其原型如下：

```
int recv(SOCKET s,char FAR* buf,int len,int flags);
```

☑ s：套接字标识。

☑ buf：表示接收数据的缓冲区。

☑ len：表示buf的长度。

☑ flags：表示函数的调用方式。如果为MSG_PEEK，表示查看传来的数据，在序列前端的数据会被复制一份到返回缓冲区中，但是这个数据不会从序列中移走；如果为MSG_OOB，表示处理Out-Of-Band数据，也就是外带数据。

例如，使用recv函数接收数据，代码如下：

```
recv(socket_send,Receivebuf,100,0);
```

其中，socket_send是用于连接的套接字，Receivebuf是用来接收保存数据的空间，100是该空间的大小。

12．send函数

send函数用于在面向连接方式的套接字间发送数据。其原型如下：

```
int send(SOCKET s,const char FAR * buf, int len,int flags);
```

☑ s：套接字标识。

☑ buf：表示存放要发送数据的缓冲区。

☑ len：表示缓冲区长度。

☑ flags：表示函数的调用方式。

例如，使用send函数发送数据，代码如下：

```
send(socket_receive,Sendbuf,100,0);
```

在上面代码中，socket_receive用于连接的套接字，而Sendbuf保存要发送的数据，100为该数据的大小。

13．recvfrom函数

recvfrom函数用于接收一个数据报信息并保存源地址。其原型如下：

```
int recvfrom(SOCKET s, char FAR* buf, int len, int flags,
struct sockaddr FAR* from, int FAR* fromlen);
```

☑ s：表示准备接收数据的套接字。

☑ buf：指向缓冲区的指针，用来接收数据。

☑ len：表示缓冲区的长度。

☑ flags：表示函数的调用方式。

☑ from：一个指向地址结构的指针，用来接收发送数据方的地址信息。

☑ fromlen：表示缓冲区的长度。

14．sendto函数

sendto函数用于向一个特定的目的方发送数据。其原型如下：

```
int sendto(SOCKET s,const char FAR * buf,int len,int
flags,const struct sockaddr FAR * to,int tolen);
```

- ☑ s：套接字标识符（可能已经建立连接）。
- ☑ buf：指向缓冲区的指针，该缓冲区包含将要发送的数据。
- ☑ len：表示缓冲区的长度。
- ☑ flags：表示函数的调用方式。
- ☑ to：指定目标套接字的地址。
- ☑ tolen：表示缓冲区的长度。

15．WSACleanup函数

WSACleanup函数用于释放为Ws2_32.dll动态链接库初始化时分配的资源。其原型如下：

```
int WSACleanup(void);
```

例如，使用该函数可关闭动态链接库，代码如下：

```
WSACleanup();                    /*关闭动态链接库*/
```

16.3.2 基于TCP的网络聊天程序



本节将编写一个基于TCP网络通信的聊天程序，希望通过本案例，读者可对前面学习的内容有一个更深的理解。

【例16.1】网络聊天服务器端的程序（实例位置：资源包\TM\sl\16\01）

本实例将实现一个基于TCP的网络聊天程序，根据有关TCP套接字socket编程中服务器的设计过程，编写下面的代码。

```
#include<stdio.h>
#include<winsock.h>                                    /*包含 winsock 头文件*/

int main()
{
    /*------------------------------------------------------------*/
    /*--------------------定义变量--------------------------------*/
    /*------------------------------------------------------------*/
    char Sendbuf[100];                                 /*发送数据的缓冲区*/
    char Receivebuf[100];                              /*接收数据的缓冲区*/
    int SendLen;                                       /*发送数据的长度*/
    int ReceiveLen;                                    /*接收数据的长度*/
    int Length;                                        /*SOCKADDR 的大小*/

    SOCKET socket_server;                              /*定义服务器套接字*/
    SOCKET socket_receive;                             /*定义连接套接字*/

    SOCKADDR_IN Server_add;                            /*服务器地址信息结构*/
    SOCKADDR_IN Client_add;                            /*客户端地址信息结构*/

    WORD wVersionRequested;                            /*字（word）：unsigned short*/
    WSADATA wsaData;                                   /*库版本信息结构*/
    int error;                                         /*表示错误*/

    /*------------------------------------------------------------*/
    /*--------------------初始化套接字库--------------------------*/
    /*------------------------------------------------------------*/
    /*定义版本类型，将两个字节组合成一个字，前面是低字节，后面是高字节*/
    wVersionRequested = MAKEWORD(2, 2);
    /*加载套接字库，初始化 Ws2_32.dll 动态链接库*/
```

```
error = WSAStartup(wVersionRequested, &wsaData);
if(error!=0)
{
    printf("加载套接字失败！ ");
    return 0;                                   /*程序结束*/
}
/*判断请求加载的版本号是否符合要求*/
if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
{
    WSACleanup();                               /*不符合，关闭套接字库*/
    return 0;                                   /*程序结束*/
}

/*-----------------------------------------------------------*/
/*-------------------设置连接地址----------------------------*/
/*-----------------------------------------------------------*/
Server_add.sin_family=AF_INET;          /*地址族，必须是 AF_INET。注意，只有它不是网络字节顺序*
Server_add.sin_addr.S_un.S_addr=htonl(INADDR_ANY);/*主机地址*/
Server_add.sin_port=htons(5000);                /*端口号*/

/*-------------------创建套接字------------------------------*/
/*AF_INET 表示指定地址族，SOCK_STREAM 表示流式套接字 TCP，特定的地址族相关协议*/
socket_server=socket(AF_INET,SOCK_STREAM,0);

/*-----------------------------------------------------------*/
/*--------绑定套接字到本地的某个地址和端口上---------*/
/*-----------------------------------------------------------*/
/*socket_server 为套接字，(SOCKADDR*)&Server_add 为服务器地址*/
if(bind(socket_server,(SOCKADDR*)&Server_add,sizeof(SOCKADDR))==SOCKET_ERROR)
{
    printf("绑定失败\n");
}

/*-----------------------------------------------------------*/
/*--------------------设置套接字为监听状态-----------------*/
/*-----------------------------------------------------------*/
/*监听状态，为连接做准备，最大等待数目为 5*/
if(listen(socket_server,5)<0)
{
    printf("监听失败\n");
}

/*-----------------------------------------------------------*/
/*----------------------接受连接-----------------------------*/
/*-----------------------------------------------------------*/
Length=sizeof(SOCKADDR);
/*接受客户端的发送请求，等待客户端发送 connect 请求*/
socket_receive=accept(socket_server,(SOCKADDR*)&Client_add,&Length);
if(socket_receive==SOCKET_ERROR)
```

```
    {
        printf("接受连接失败");
    }

    /*------------------------------------------------------------*/
    /*-----------------进行聊天-----------------------------------*/
    /*------------------------------------------------------------*/
    while(1)                                                    /*无限循环*/
    {
        /*-----------------接收数据-----------------------------------*/
        ReceiveLen =recv(socket_receive,Receivebuf,100,0);
        if(ReceiveLen<0)
        {
            printf("接收失败\n");
            printf("程序退出\n");
            break;
        }
        else
        {
            printf("client say: %s\n",Receivebuf);
        }

        /*-----------------发送数据-----------------------------------*/
        printf("please enter message:");
        scanf("%s",Sendbuf);
        SendLen=send(socket_receive,Sendbuf,100,0);
        if(SendLen<0)
        {
            printf("发送失败\n");
        }
    }

    /*------------------------------------------------------------*/
    /*-----------------释放套接字，关闭动态库---------------------*/
    /*------------------------------------------------------------*/
    closesocket(socket_receive);                                /*释放客户端的套接字资源*/
    closesocket(socket_server);                                 /*释放服务器的套接字资源*/
    WSACleanup();                                               /*关闭动态链接库*/
    return 0;
}
```

运行程序之前，要先添加相应的库文件ws2_32.lib，如图16.2所示。

以上就是网络聊天程序服务器端的代码。整个程序流程按照以下顺序编写。

（1）创建套接字。

（2）绑定套接字到本地的地址和端口上。

（3）设置套接字为监听状态。

（4）接受请求连接的请求。

（5）进行通信。

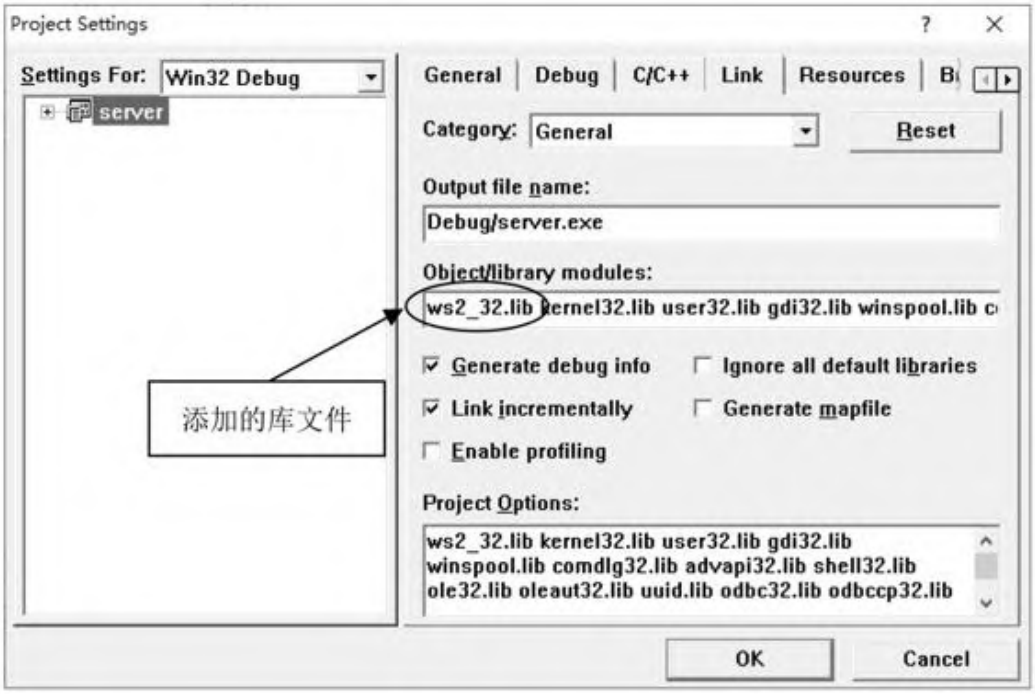


图16.2　添加的库文件

（6）通信完毕，释放套接字资源。

【例16.2】网络聊天客户端的程序（实例位置：资源包\TM\sl\16\02）

根据有关TCP套接字socket编程中的客户端设计过程，编写下面的代码：

```
#include<stdio.h>
#include<winsock.h>                                    /*包含 winsock 头文件*/

int main()
{
    /*------------------------------------------------------------*/
    /*----------------------定义变量----------------------------*/
    /*------------------------------------------------------------*/
    char Sendbuf[100];                                 /*发送数据的缓冲区*/
    char Receivebuf[100];                              /*接收数据的缓冲区*/
    int SendLen;                                       /*发送数据的长度*/
    int ReceiveLen;                                    /*接收数据的长度*/

    SOCKET socket_send;                                /*定义套接字*/
    SOCKADDR_IN Server_add;                            /*服务器地址信息结构*/

    WORD wVersionRequested;                            /*字（word）：unsigned short*/
    WSADATA wsaData;                                   /*库版本信息结构*/
    int error;                                         /*表示错误*/

    /*------------------------------------------------------------*/
    /*----------------------初始化套接字库----------------------*/
    /*------------------------------------------------------------*/
    /*定义版本类型。将两个字节组合成一个字，前面是低字节，后面是高字节*/
    wVersionRequested = MAKEWORD(2, 2);
```

```
/*加载套接字库，初始化 Ws2_32.dll 动态链接库*/
error = WSAStartup(wVersionRequested, &wsaData);
if(error!=0)
{
    printf("加载套接字失败！");
    return 0;                                          /*程序结束*/
}
/*判断请求加载的版本号是否符合要求*/
if(LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2)
{
    WSACleanup();                                      /*不符合，关闭套接字库*/
    return 0;                                          /*程序结束*/
}

/*--------------------------------------------------------------*/
/*----------------------设置服务器地址----------------------*/
/*--------------------------------------------------------------*/
Server_add.sin_family=AF_INET;          /*地址族，必须是 AF_INET。注意，只有它不是网络字节顺序*
/*服务器地址，将一个点分十进制表示为 IP 地址，inet_ntoa 是将地址转换成字符串*/
Server_add.sin_addr.S_un.S_addr = inet_addr("192.168.1.43");
Server_add.sin_port=htons(5000);                       /*端口号*/

/*--------------------------------------------------------------*/
/*----------------------进行服务器连接----------------------*/
/*--------------------------------------------------------------*/
/*客户端创建套接字，但是不需要绑定，只需要和服务器建立起连接即可*/
/*socket_sendr 表示的是套接字，Server_add 是服务器的地址结构*/
socket_send=socket(AF_INET,SOCK_STREAM,0);

/*--------------------------------------------------------------*/
/*----------------------创建用于连接的套接字----------------*/
/*--------------------------------------------------------------*/
/*AF_INET 表示指定地址族，SOCK_STREAM 表示流式套接字 TCP，特定的地址族相关协议*/
if(connect(socket_send,(SOCKADDR*)&Server_add,sizeof(SOCKADDR)) == SOCKET_ERROR)
{
    printf("连接失败!\n");
}

/*--------------------------------------------------------------*/
/*----------------------进行聊天----------------------------*/
/*--------------------------------------------------------------*/
while(1)                                               /*无限循环*/
{
    /*---------------发送数据过程---------------*/
    printf("please enter message:");
    scanf("%s",Sendbuf);
    SendLen = send(socket_send,Sendbuf,100,0);         /*发送数据*/
    if(SendLen < 0)
    {
        printf("发送失败!\n");
```

```
        }

        /*-------------接收数据过程----------------*/
        ReceiveLen =recv(socket_send,Receivebuf,100,0);        /*接收数据*/
        if(ReceiveLen<0)
        {
            printf("接收失败\n");
            printf("程序退出\n");
            break;                                             /*跳出循环*/
        }
        else
        {
            printf("Server say: %s\n",Receivebuf);
        }
    }

    /*-----------------------------------------------------------*/
    /*----------------------释放套接字，关闭动态库----------*/
    /*-----------------------------------------------------------*/
    closesocket(socket_send);                                  /*释放套接字资源*/
    WSACleanup();                                              /*关闭动态链接库*/
    return 0;
}
```

以上就是网络聊天程序客户端的代码。整个程序流程按照以下顺序编写。

（1）创建套接字。

（2）发出连接请求。

（3）请求连接后进行通信操作。

（4）释放套接字资源。

先运行例16.1，然后运行例16.2。首先在客户端输入数据，按Enter键后，即可以在服务器端看到输入的信息。客户端输入完后，服务器端就可以对其进行回复。在服务器端输入数据并按Enter键，可发送消息到客户端。

客户端程序运行效果如图16.3所示。服务器端程序运行效果如图16.4所示。



图16.3　客户端



图16.4　服务器端

注意

要实现网络通信，一定先运行服务器端，再运行客户端。注意，在客户端输入IP地址时，计划与哪台计算机通信，就输入哪台计算机的IP地址。

编程训练（答案位置：资源包\TM\sl\16\编程训练\）

训练1：反馈连接信息　设计程序，要求当客户端连接到服务器一端时，服务器会显示连接的提示信息，并反馈信息给客户。

训练2：基于UCP聊天程序　修改例16.1和例16.2，将其改为基于UCP的网络聊天程序。

16.4　实践与练习

（答案位置：资源包\TM\sl\16\实践与练习\）

综合练习1：获取主机名　使用网络socket函数，获取本地主机的名称，运行结果如下：

```
主机名是：DESKTOP-QM9GUAL
```

综合练习2：附近的人　微信、QQ都可以搜索附近的人，并可以主动向其打招呼。将附近的人看成客户端，将自己看成服务器端，模拟向附近的人打招呼。运行效果如下：

Client:

```
你说：你好
来自附近的人：你好
你说：我能加你吗
来自附近的人：不可以
```

Sever：

```
来自附近的人：你好
你说：你好
来自附近的人：我能加你吗
你说：不可以
```

综合练习3：获取主机IP地址　使用socket函数，获取主机的IP地址，运行结果如下：

```
本机的 IP 地址是 192.168.1.22
```

综合练习4：网络传情书　使用网络向心目中喜欢的女（男）生传送一封情书，使用服务器端和客户端实现。在服务器端先创建情书love.txt文件，如图16.5所示，然后先运行服务器端，再运行客户端。运行的服务器端效果如图16.6所示，运行客户端之后在客户端文件中会出现download.txt文件，如图16.7所示。

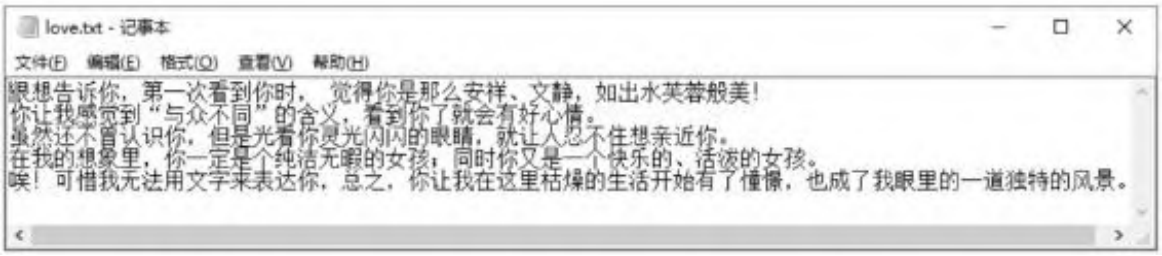


图16.5　love.txt文件

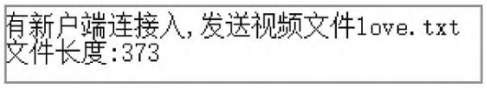


图16.6　服务器端

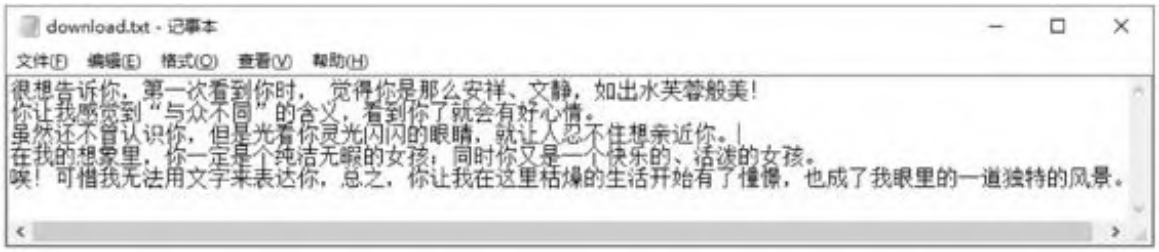


图16.7　传送的download.txt文件

综合练习5：推荐铃声音乐　编写程序，通过网络传送音乐文件，例如传送一个名为bj.mp3的音频。先运行服务器端，再运行客户端。运行之后的服务器端如图16.8所示，此时音频也会出现在客户端，如图16.9所示。

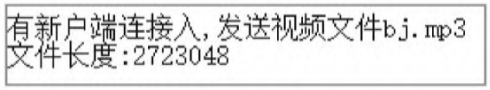


图16.8　服务器端



图16.9　客户端文件夹

综合练习6：网络传输Word文件　编写程序，利用网络传送Word文档。在服务器端文件夹内创建要传送的Word文档name.docx，如图16.10所示。运行程序后，在客户端文件夹内将出现nameDown.docx文件，如图16.11所示。



图16.10　服务器端文件夹



图16.11　客户端文件夹

综合练习7：实现通信功能　实现服务器与客户端的通信功能，客户端输入“hello”，服务器端返回“登录成功”。运行结果如下：

Client:

```
please enter message:hello
Server say: 登录成功
please enter message:
```

Sever:

```
client say: hello
please enter message: 登录成功
```

综合练习8：发送Email文件　一些重要文件，往往会选择Email的方式发送。编写程序，使用网络传输简历模板。服务器端文件夹的简历模板

内容如图16.12所示，运行程序后，客户端文件的简历模板（下载）内容如图16.13所示。



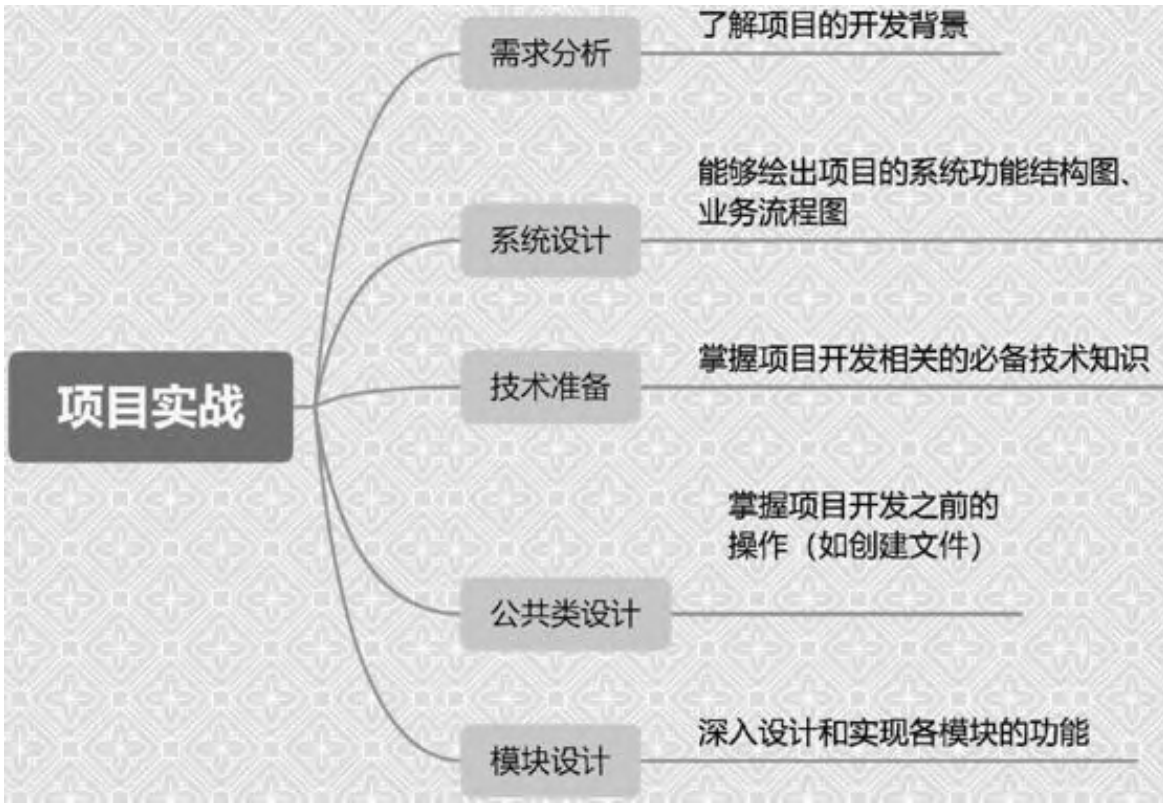
图16.12 服务器端文件的简历模板



图16.13 客户端文件的简历模板（下载）内容

第4篇
项目实战

本篇运用软件工程的设计思想，开发了两个游戏——单词背记闯关游戏和防空大战游戏。书中按照“需求分析➡系统设计➡技术准备➡公共类设计➡实现项目模块功能”的步骤，带领读者一步一步地亲身体验项目开发的全过程。



第17章
单词背记闯关游戏

通过前面的学习，读者已对C语言的知识体系有了一定的了解，本章将引领大家开发自己的第一个项目——单词背记闯关游戏。运行游戏，欢迎界面中，玩家可切换选项，开始游戏、查看积分规则或退出游戏；游戏过程中，可随机出现中文或英文的单词以及4个答案选项，玩家给出答案后系统将进行计分，并刷新历史最高分；游戏结束界面会显示一段简单评价，以及玩家与记载的历史最高分的差值。

通过本项目，读者可快速掌握文件的常见操作，熟悉windows.h函数库中的光标函数，并深刻体验开发一个项目带来的趣味性和成就感。

本章的知识架构及重难点如下：



17.1 需求分析



本章将使用C语言，开发一个单词背记闯关游戏，需要设计欢迎界面、游戏界面，实现计分功能，能够存储最高分，并能在游戏结束时显示字符画以及玩家当前分与历史最高分的差值。

17.2 系统设计

17.2.1 系统目标

单词背记闯关游戏需要实现以下目标。

- ☑ 欢迎界面，通过不同选项可进入不同界面。
- ☑ 游戏过程中的中英文互译。
- ☑ 判断玩家答案是否正确，显示积分情况。
- ☑ 存储最高分，并在游戏结束界面显示玩家当前分以及与历史最高分的相差情况。

17.2.2 构建开发环境

- ☑ 系统开发平台：Visual C++ 6.0。
- ☑ 系统开发语言：C语言。
- ☑ 运行平台：Windows 7（SP1）/ Windows 8/Windows 8.1/Windows 10。

17.2.3 系统功能结构

单词背记闯关游戏是一个简单的应用程序，其需要实现的系统功能如下。

☑ 欢迎界面：显示游戏名字，提醒用户进行下一步选择。选择数字1，进入游戏界面；选择数字2，进入积分规则说明界面；选择数字3，退出游戏。

☑ 游戏界面：系统随机弹出中文或中文，用户选择对应的翻译。每次选择后，会有提示是否继续游戏，选择1表示游戏继续，选择0表示结束游戏。

☑ 游戏积分功能：用户给出答案后，判断用户的答案正确与否，正确加10分，错误扣10分。

☑ 存储游戏最高分功能：游戏中会自动创建save.txt文件，以保存玩家的历史最高分。若当前分数大于历史最高分，自动更新该分值，否则在游戏结束界面显示当前分与历史最高分的差值。

☑ 游戏结束界面：结束界面中将出现一个字符画，同时输出此次得分情况，以及与历史最高分的分数差。字符画下面，可选择是否重新开始游戏。

单词背记闯关游戏的系统功能结构如图17.1所示。

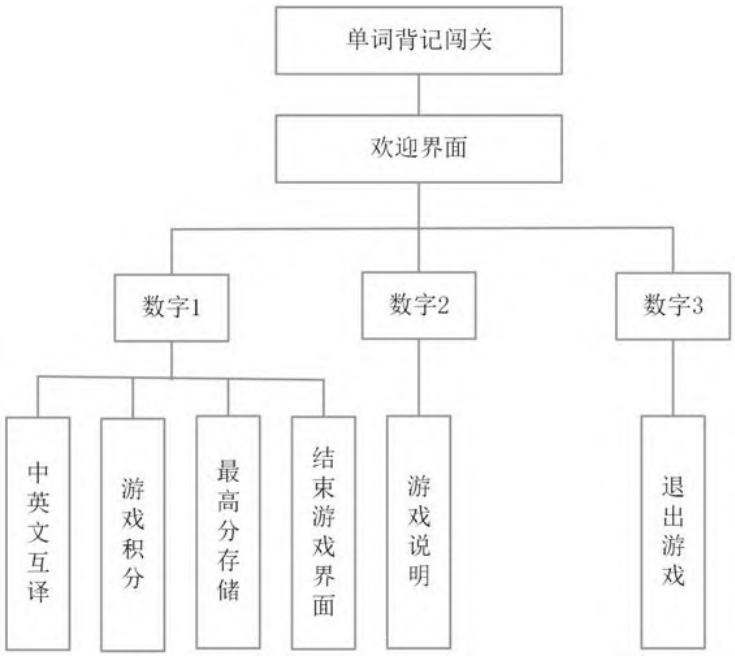


图17.1　系统功能结构

17.2.4 业务流程图

单词背记闯关游戏的业务流程如图17.2所示。

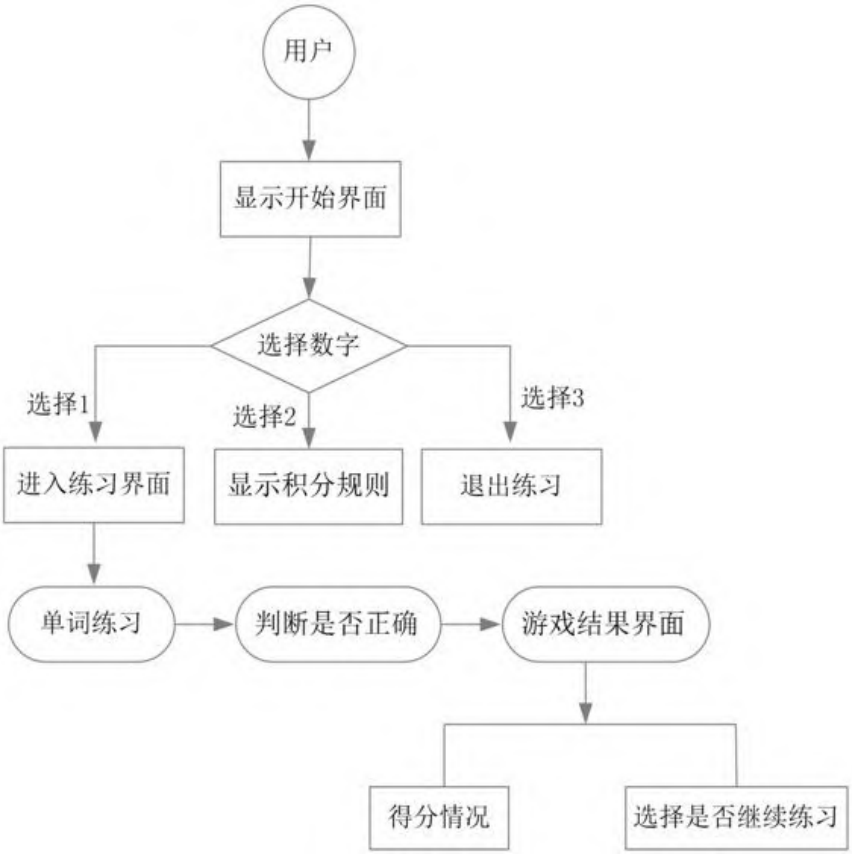


图17.2 业务流程图

17.2.5 系统预览

单词背记闯关游戏由多个模块组成，限于篇幅，书中仅给出了核心代码，完整的代码请参照资源包中的源程序。

游戏开始界面如图17.3所示，游戏过程主界面如图17.4所示。

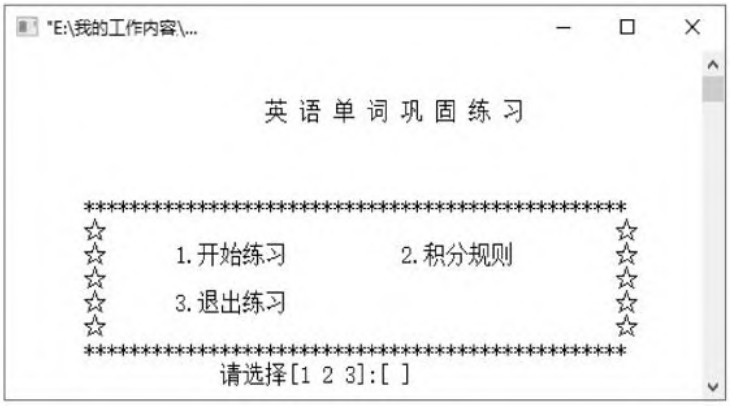


图17.3　开始界面

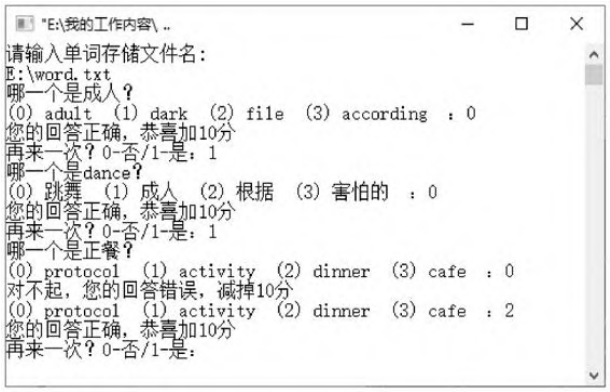


图17.4　单词练习主界面

积分规则界面如图17.5所示，游戏结束界面如图17.6所示。

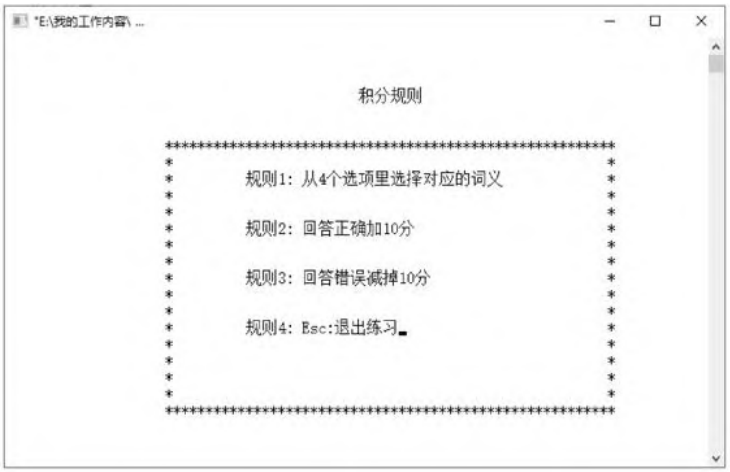


图17.5　积分规则界面

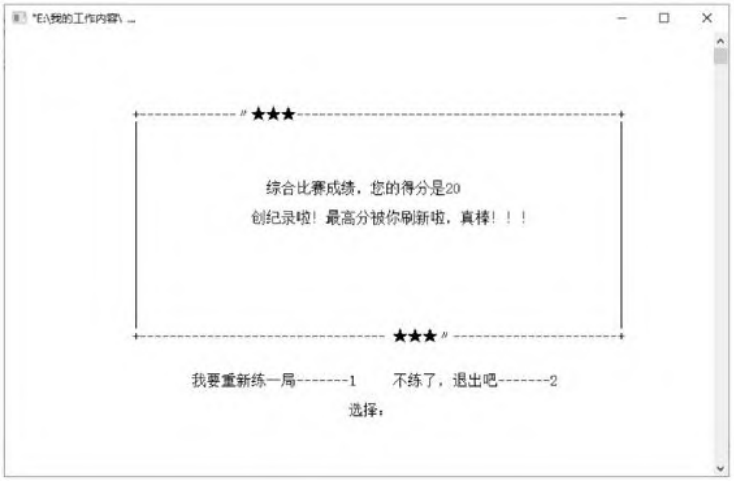


图17.6　显示结果界面

17.3　技术准备



本程序使用了文件操作函数，下面就来详细介绍一下对应的文件操作函数。

17.3.1　文件的基本操作

1．文件指针

文件指针是一个指向文件信息的指针，这些信息包括文件名、状态和当前位置，保存在一个结构体变量中。使用文件时需要在内存中为其分配空间，用来存放文件的基本信息。该结构体类型由系统定义，C语言规定该类型为FILE型，其声明如下：

```
typedef struct
{
    short level;
    unsigned flags;
    char fd;
    unsigned char hold;
    short bsize;
    unsigned char *buffer;
    unsigned ar *curp;
    unsigned istemp;
    short token;
}FILE;
```

上述代码使用typedef定义了一个FILE结构体类型，编写程序时可直接使用上面定义的FILE类型来定义变量。注意，定义变量时不必将结构体内容全部给出，只需写成如下形式：

```
FILE *fp;
```

说明

fp是一个指向FILE类型的指针变量。

2．打开文件

fopen函数用来打开一个文件，其本质就是创建了一个流。fopen函数的原型在stdio.h中，其调用的一般形式为：

```
FILE *fp;
fp=fopen(文件名,使用文件方式);
```

其中，“文件名”是将要被打开的文件名称，“使用文件方式”用于指定对打开文件进行的读写操作方式，如表17.1所示。

表17.1　使用文件方式

使用文件方式	含　义
r（只读）	打开一个文本文件，只允许读数据
w（只写）	打开或创建一个文本文件，只允许写数据
a（追加）	打开一个文本文件，并在文件末尾写数据
rb（只读）	打开一个二进制文件，只允许读数据
wb（只写）	打开或创建一个二进制文件，只允许写数据
ab（追加）	打开一个二进制文件，并在文件末尾写数据
r+（读写）	打开一个文本文件，允许读和写
w+（读写）	打开或创建一个文本文件，允许读和写
a+（读写）	打开一个文本文件，允许读，或在文件末尾追加数据
rb+（读写）	打开一个二进制文件，允许读和写
wb+（读写）	打开或创建一个二进制文件，允许读和写
ab+（读写）	打开一个二进制文件，允许读，或在文件末尾追加数据

如果要以只读方式打开文件名为123的文本文档文件，应写成如下形式：

```
FILE *fp;
fp=("123.txt","r");
```

如果使用fopen函数打开文件成功，则返回一个有确定指向的FILE类型指针；若打开文件失败，则返回NULL。打开失败的原因通常为以下几个方面。

☑ 指定的盘符或路径不存在。

☑ 文件名中含有无效字符。

☑ 以r模式打开一个不存在的文件。

3．关闭文件

文件在使用完毕后，应使用fclose函数将其及时关闭。就像水龙头，当用完水时，必须要关上水龙头。fclose函数和fopen函数一样，原型也在stdio.h中，调用的一般形式如下：

```
fclose(文件指针);
```

例如：

```
fclose(fp);
```

fclose函数也带回一个值，当正常完成关闭文件操作时，fclose函数的返回值为0，否则返回EOF。

说明

在程序结束之前应关闭所有文件，以防止因为没有关闭文件而造成的数据流失。

17.3.2 文件的读写操作

1．写字符函数fputc、fputs

fputc函数用于把一个字符写到磁盘文件（fp所指向的是文件）中，其一般形式如下：

```
ch=fputc(ch,fp);
```

其中，ch是待写入的字符，可以是一个字符常量，也可以是一个字符变量。fp是文件指针变量。如果字符写入成功，则返回值就是写入的字符；如果写入失败，则返回EOF。

fputs函数用于向文件中写入多个字符。其一般形式如下：

```
fputs(字符串,文件指针);
```

其中，字符串可以是字符串常量，也可以是字符数组名、字符指针或字符变量。

2．格式输出函数fprintf

fprintf函数的作用是将数据格式化保存到文件中，可根据需要改变输出流。其一般形式如下：

```
ch=fprintf(文件类型指针,格式字符串,输出列表);
```

3．读取字符函数fgetc、fgets

fgetc函数用于从指定的文件（fp指向的文件）中读取一个字符赋给ch。其一般形式如下：

```
ch=fgetc(fp);
```

使用fgetc函数，可逐个字符地读取某个文件。需要注意的是，文件必须以读或读写的方式打开。遇到文件结束符时，将返回文件结束标志EOF。

fgets函数用于从指定的文件中读取一个字符串到字符数组中。其一般形式如下：

```
fgets(字符数组名,n,文件指针);
```

其中，n表示待读取的字符串中的字符个数（包含“\0”）。

4．格式输入函数fscanf

fscanf函数的作用是按照格式输入数据。其一般形式如下：

```
fscanf(文件类型指针,格式字符串,输入列表);
```

例如，下面代码的含义是读取fp所指向的文件中i的值。

```
fscanf(fp,"%d",&i);
```

17.4 公共类设计



编写程序之前，首先应在磁盘中创建一个存储单词的文件（如word.txt），内容为英语单词以及对应的汉语，如图17.7所示。



图17.7 存储单词的文件

17.5 预处理模块设计



17.5.1 模块概述

预处理模块设计包括文件引用、定义全局变量、声明函数以及定义字符常量等。下面分别介绍其代码实现。

17.5.2 代码实现

1．文件引用

首先使用#include命令引入部分库文件，对程序的一些基本函数进行支持。代码如下：

```
#include<stdio.h>                    //包含标准输入/输出库
#include<time.h>                     //包含日期和时间库，用于获得随机数
#include<windows.h>                  //包含 windows 库，用于控制 dos 界面
#include<stdlib.h>                   //包含标准库，里面定义了一些宏和通用工具函数
#include<string.h>
#include<conio.h>                    //包含控制台输入/输出库，用于接收按键盘信息和产生对应操作
```

2．定义全局变量

把程序中公用的变量放在程序的最前面，即为全局变量。项目需要定义的全局变量如下：

```
int QNO;                             //单词数量
char **cptr;                         //指向中文单词的指针数组
char **eptr;                         //指向英语单词的指针数组
int score = 0;                       //分数
int HighScore = 0;                   //最高分数
```

3．声明函数

要实现项目的全部功能，需要定义很多函数。在定义函数之前需要先进行函数声明，代码如下：

```
void gotoxy(int x, int y);                  //设置光标函数
void start();                               //开始界面函数
void run();                                 //判断选择题结果函数
void File_in();                             //存储最高分进文件
void File_out();                            //读取文件最高分
void endgame();                             //结束练习函数
void choose();                              //分支选择函数
void Lost();                                //结果界面
void rule();                                //积分规则函数
int read();                                 //读取单词文件函数
int make(int c[], int n);                   //生成选项并返回正确的下标
void print(const int c[], int sw);          //显示选型
```

4．定义字符常量

项目需要定义的字符常量代码如下：

```
#define CNO 4                               //选项数量
#define swap(type, x, y)    do { type t = x; x = y; y = t; } while (0)
```

17.6　欢迎界面设计



17.6.1　模块概述

欢迎界面（也就是开始游戏界面）要实现3个切换选项，分别是开始练习、积分规则和退出练习。为了界面美观，这里采用了多个颜色，而且还定义光标来控制每个字符的排列位置。欢迎界面的运行效果如图17.8所示。

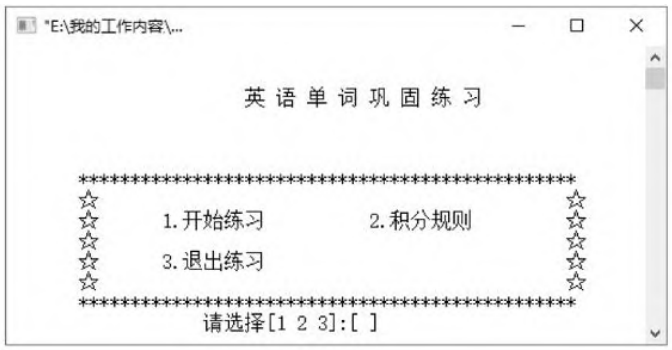


图17.8　欢迎界面

17.6.2　代码实现

（1）定义gotoxy函数，控制光标的位置，具体代码如下：

```
/*设置光标的位置*/
void gotoxy(int x, int y)
{
    COORD c;
    c.X = x;
    c.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), c);
}
```

（2）欢迎界面主要由菜单选项组成，这里定义start函数实现欢迎界面，具体代码如下：

```
/*欢迎界面（开始游戏界面）设计*/
void start()
{
    int n;
    int i, j = 1;
    gotoxy(23, 2);
    printf("英 语 单 词 巩 固 练 习");
    for (i = 6; i <= 12; i++)                    //外层循环，用“*”输出上下边框
    {
        for (j = 7; j <= 54; j++)                //内层循环，用“☆”输出左右边框
        {
            gotoxy(j, i);
            if (i == 6 || i == 12)
            {
                printf("*");
            }
            else if (j == 7 || j == 54)
            {
                printf("☆");
            }
        }
    }
    gotoxy(15, 8);                               //gotoxy 函数用于控制光标位置
    printf("1.开始练习");
    gotoxy(35, 8);
    printf("2.积分规则");
    gotoxy(15, 10);
    printf("3.退出练习");
    gotoxy(19, 13);
    printf("请选择[1 2 3]:[ ]\b\b");                //\b 的功能是退格，使得光标处于[]中间
    scanf("%d", &n);                             //用户输入自己的选择
    switch (n)
    {
        case 1:
            system("cls");
            break;
        case 2:
            break;
        case 3:
            exit(0);                             //退出游戏
            break;
        default:                                 //用户输入非 1～3 范围的数字
            gotoxy(40, 28);
            printf("请输入 1~3 的数!");
            getch();                             //输入任意键
            system("cls");                       //清屏
            start();
    }
}
```

17.7　积分规则界面设计



17.7.1 模块概述

在欢迎界面中输入数字2，即可进入积分规则界面，如图17.9所示。

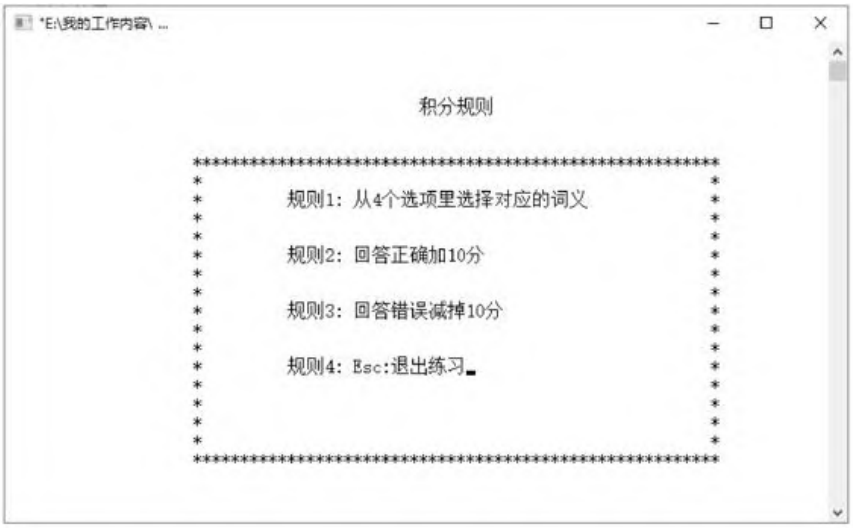


图17.9 积分规则界面

17.7.2 代码实现

（1）定义rule函数，实现积分规则显示，具体代码如下：

```
/*积分规则界面设计*/
void rule()
{
    int i, j = 1;
    system("cls");
    gotoxy(44, 3);
    printf("积分规则");
    for (i = 6; i <= 22; i++)                    //外层循环，用“*”输出上下边框
    {
        for (j = 20; j <= 75; j++)               //内层循环，用“*”输出左右边框
        {
            gotoxy(j, i);
            if (i == 6 || i == 22) printf("*");
            else if (j == 20 || j == 75) printf("*");
        }
    }
    gotoxy(30, 8);
    printf("规则 1: 从 4 个选项里选择对应的词义");
    gotoxy(30, 11);
    printf("规则 2: 回答正确加 10 分");
    gotoxy(30, 14);
    printf("规则 3: 回答错误减掉 10 分");
    gotoxy(30, 17);
    printf("规则 4: Esc:退出练习");
    getch();                                     //按任意键返回主界面
    system("cls");
    start();
}
```

（2）修改欢迎界面start函数中的switch选项代码，具体如下：

```
switch (n)
{
    case 1:
        system("cls");
        break;
    case 2:
        rule();                                  //新添加的代码，调用积分规则函数
        break;
    case 3:
        exit(0);                                 //退出游戏
        break;
    default:                                     //用户输入非 1~3 范围的数字
        gotoxy(40, 28);
    printf("请输入 1~3 的数!");
    getch();                                     //输入任意键
    system("cls");                               //清屏
    start();
}
```

17.8　显示游戏最高分设计

17.8.1 模块概述

本模块需要实现单词背记闯关游戏的历史最高分功能。首先需要创建一个最高分文件save.txt，如图17.10所示。单词背记练习结束时，可读取其中的最高分，也可写入新的最高分。



图17.10 最高分文件

17.8.2 代码实现

（1）定义read函数，读取单词存储文件，具体代码如下：

```
/*读取单词文件*/
int read()
{
    int i;
    FILE *fp;
    char filename[30];
    printf("请输入单词存储文件名:\n");
    scanf("%s", filename);
    if ((fp = fopen(filename, "r")) == NULL)
        return 1;
    fscanf(fp, "%d", &QNO);                    //读取单词数量
    if ((cptr = calloc(QNO, sizeof(char *))) == NULL) return 1;
    if ((eptr = calloc(QNO, sizeof(char *))) == NULL) return 1;
    for (i = 0; i < QNO; i++) {
        char etemp[1024];
        char ctemp[1024];
        fscanf(fp, "%s%s", etemp, ctemp);
        if ((eptr[i] = malloc(strlen(etemp) + 1)) == NULL) return 1;
        if ((cptr[i] = malloc(strlen(ctemp) + 1)) == NULL) return 1;
        strcpy(eptr[i], etemp);
        strcpy(cptr[i], ctemp);
    }
    fclose(fp);
    return 0;
}
```

游戏设定答对一个题目加10分，答错一个题目减10分，使用save.txt文件存储游戏的最高分，并显示当前分数与最高分差多少分。

（2）定义File_in函数，把新的历史最高分存储到文件中，具体代码如下：

```
/*储存新的最高分到文件中*/
void File_in()
{
    FILE *fp;
    fp = fopen("save.txt", "w+");          //以读写方式建立一个名为 save.txt 的文件
    fprintf(fp, "%d", score);              //把分数写进文件中
    fclose(fp);                            //关闭文件
}
```

（3）定义File_out函数，读取文件中存储的历史最高分，具体代码如下：

```
/*从文件中读取最高分*/
void File_out()
{
    FILE *fp;
    fp = fopen("save.txt", "a+");          //打开文件 save.txt
    fscanf(fp, "%d", &HighScore);          //读取文件中的最高分
    fclose(fp);                            //关闭文件
}
```

17.9 系统逻辑设计



17.9.1 模块概述

用户在欢迎界面中输入数字1时，将进入如图17.11所示的单词背记闯关游戏主界面。



图17.11　单词背记闯关游戏主界面

17.9.2　代码实现

（1）定义make函数，用来随机生成4个答案选项，具体代码如下：

```
/*生成答案选项并返回正确选项的下标*/
int make(int c[], int n)
{
    int i, j, x;
    c[0] = n;                                   //在开头元素中存入正确答案
    for (i = 1; i < CNO; i++) {
        do {                                    //生成不重复的随机数
            x = rand() % QNO;
            for (j = 0; j < i; j++)
                if (c[j] == x)                  //如果生成了相同的随机数
                    break;
        } while (i != j);
        c[i] = x;
    }
    j = rand() % CNO;
    if (j != 0)
        swap(int c[0], c[j]);                   //移动正确答案
    return j;
}
```

（2）定义print函数，用来显示4个答案选项，具体代码如下：

```
/*显示 4 个答案选项*/
void print(const int c[], int sw)
{
	int i;
	for (i = 0; i < CNO; i++)
		printf("(%d) %s   ", i, sw ? cptr[c[i]] : eptr[c[i]]);
	printf("：");
}
```

（3）定义run函数，运行程序，实现整个单词背记练习过程的逻辑与功能设计。具体代码如下：

```
/*判断用户输入的答案是否正确，并统计得分情况*/
int run()
{
    int i;
    int nq, pq;                                  //题目编号和上一次的题目编号
    int na;                                      //正确答案的编号
    int sw;                                      //题目语言（0：中文/1：英语）
    int retry;                                   //是否重新挑战
    int cand[CNO];                               //选项的编号
    if (read() == 1)                             //调用 read 函数，读取单词文件
    {
        printf("\a 单词文件读取失败。\n");
        return 1;
    }
    srand(time(NULL));                           //设定随机数的种子

    pq = QNO;                                    //上一次的题目编号（不存在的编号）

    do {                                         //do...while 循环，进行单词背记练习
        int no;
        do {                                     //决定用于出题的单词的编号
            nq = rand() % QNO;
        } while (nq == pq);                      //不连续出同一个单词

        na = make(cand, nq);                     //调用 make 函数，随机生成 4 个答案选项
        sw = rand() % 2;
        printf("哪一个是%s？ \n", sw ? eptr[nq] : cptr[nq]);     //输出问题

        do {
            print(cand, sw);                     //调用 print 函数，显示 4 个答案选项
            scanf("%d", &no);                    //用户输入答案
            if (no != na)
            {
                puts("对不起，您的回答错误，减掉 10 分");
                score -= 10;                     //回答错误，减 10 分
            }
        } while (no != na);
        puts("您的回答正确，恭喜加 10 分");
        score += 10;                             //回答正确，加 10 分
        pq = nq;
        printf("再来一次？ 0-否/1-是：");
        scanf("%d", &retry);                     //用户选择是否再次挑战
    } while (retry == 1);
    for (i = 0; i < QNO; i++) {
        free(eptr[i]);
        free(cptr[i]);
    }
    free(cptr);
    free(eptr);
    File_out();
    endgame();
    exit(0);
    return 0;
}
```

（4）修改欢迎界面start函数中switch选项的代码，具体如下：

```
switch (n)
    {
    case 1:
        system("cls");
        run();                          //新添加的代码，调用单词背记练习函数
        break;
    case 2:
        rule();                         //调用积分规则函数
        break;
    case 3:
        exit(0);                        //退出游戏
        break;
    default:                            //输入非 1～3 范围的数字
        gotoxy(40, 28);
        printf("请输入 1~3 的数!");
        getch();                        //输入任意键
        system("cls");                  //清屏
        start();
    }
}
```

17.10　结果显示界面设计



17.10.1　模块概述

单词背记练习结束后，将会显示练习的得分情况，并给出信息提示。如果分数大于历史最高分，则将最高分存储到文件中；如果分数小于历史最高分，则会显示与历史最高分的差值。最后还会给出一个分支选择，提醒用户是否再练一局。结果显示界面如图17.12所示。

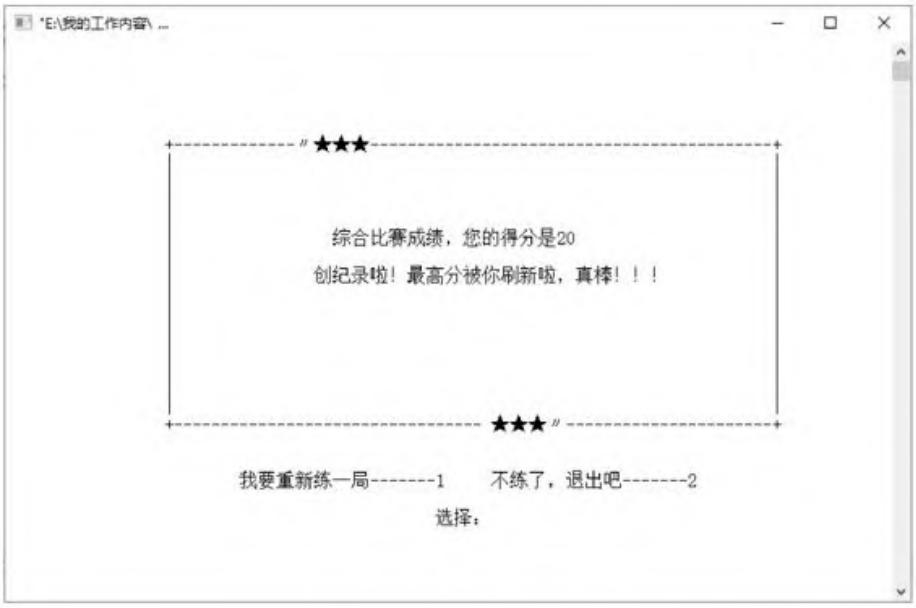


图17.12　结果显示界面

17.10.2　代码实现

（1）定义endgame函数，输出游戏结束时单词背记闯关的得分情况，具体代码如下：

```
/*输出练习结束时的提示信息*/
void endgame()
{
    system("cls");
    Lost();                                    //输出字符图案
    gotoxy(35, 10);
    printf("综合比赛成绩，您的得分是%d", score);
    if (score >= HighScore)                    //如果当前得分高于文件中存储的历史最高分
    {
        gotoxy(33, 12);
        printf("创纪录啦！最高分被你刷新啦，真棒！！！\n");
        File_in();                             //把最高分写进文件
    }
    else
    {                                          //否则，输出与历史最高分的分值差
        gotoxy(33, 12);
        printf("继续努力吧~ 你离最高分还差：%d\n", HighScore - score);
    }
    choose();
}
```

（2）定义choose函数，实现是否再次练习分支选项。用户输入数字1，将重新开始练习；输入数字2，将退出练习。具体代码如下：

```
/*实现边框下面的分支选项*/
void choose()
{
    int n;
    gotoxy(25, 23);
    printf("我要重新练一局-------1");
    gotoxy(52, 23);
    printf("不练了，退出吧-------2");
    gotoxy(46, 25);
    printf("选择：");
    scanf("%d", &n);                    //用户输入一个选择
    switch (n)
    {
        case 1:                         //用户输入数字 1
            system("cls");              //清屏
            score = 0;                  //分数归零
            start();                    //开始新一局游戏
            break;
        case 2:                         //用户输入数字 2
            exit(0);                    //退出游戏
            break;
        default:                        //用户输入其他字符
            gotoxy(35, 27);
            printf("※※您的输入有误，请重新输入※※");
            system("pause >nul");
            endgame();
            choose();                   //重复提醒用户进行选择
            break;
    }
}
```

（3）定义Lost函数，输出界面的字符图案，具体代码如下：

```
/*输出字符图案*/
void Lost()
{
	int i;
	system("cls");
	gotoxy(17, 5);
	printf("+------------------------");
	gotoxy(31, 5);
	printf("〃★★★");
	gotoxy(40, 5);
	printf("-------------------");
	gotoxy(55, 5);
	printf("----------");
	gotoxy(65, 5);
	printf("-----------------+");
	for (i = 6; i <= 19; i++)                    //输出两侧的竖边框
	{
		gotoxy(17, i);
		printf("|");
		gotoxy(82, i);
		printf("|");
	}
	gotoxy(17, 20);
	printf("+---------------------------------");

	gotoxy(52, 20);
	printf("★★★〃");
	gotoxy(60, 20);
	printf("---------------------+");
}
```

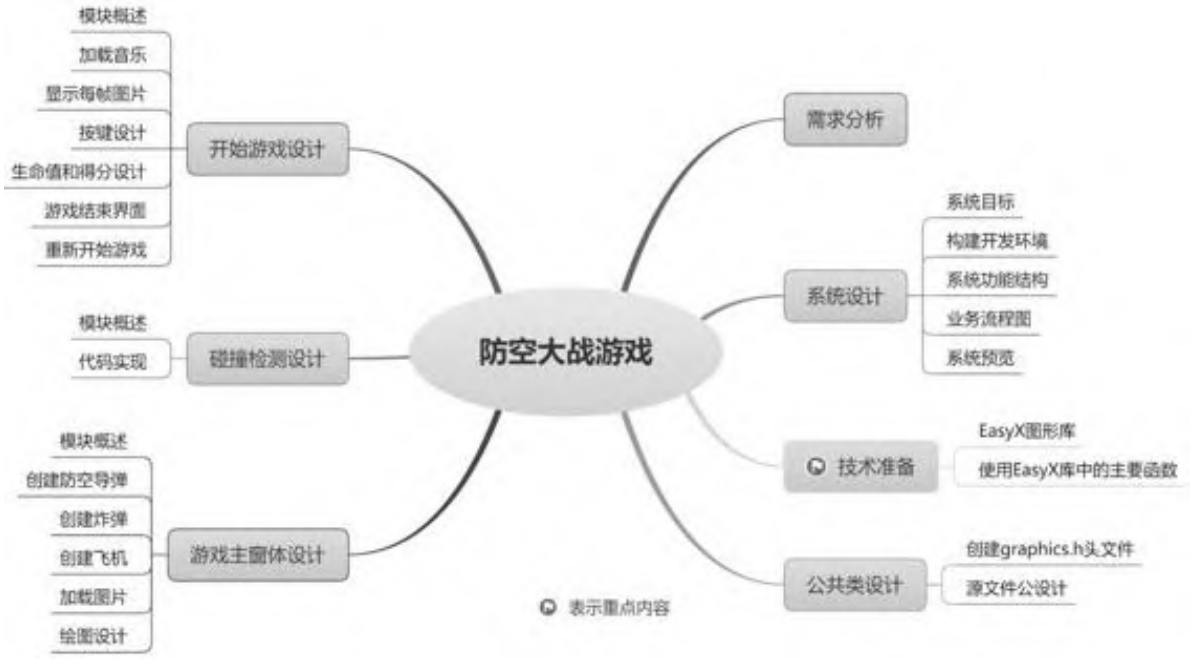
（4）主函数main（）的具体代码如下：

```
/*主函数*/
int main()
{
	start();                                     //欢迎界面
	run();                                       //练习过程
	return 0;
}
```

第18章
防空大战游戏

防空大战游戏使用Visual C++ 6.0开发环境和EasyX图形库插件完成。玩家操纵防空车，射击飞过的3种飞机。防空车每击中一个，就会累计1分；相反，飞机每击中防空车一下，防空车的生命值就会减少1分。当防空车的生命值为0时，游戏终止，系统自动弹出对话框显示玩家得分情况。为了增加游戏的沉浸性，还添加了背景音乐和防空车击中飞机时的特殊音效。

本章的知识架构及重难点如下：



18.1 需求分析



本章将利用C语言实现一个射击类游戏——防空大战游戏。要求使用键盘左右键控制防空车的位置，用空格键控制防空导弹的发射，飞机和炸弹的速度使用程序控制，出现的飞机类型是随机的。

18.2 系统设计

18.2.1 系统目标

防空大战游戏需要实现以下主要功能。

- ☑ 控制防空车的位置。
- ☑ 发射防空导弹。
- ☑ 显示玩家的生命值和得分情况。

18.2.2 构建开发环境

- ☑ 系统开发平台：Visual C++ 6.0。
- ☑ 图形库插件：EasyX。
- ☑ 系统开发语言：C语言。
- ☑ 运行平台：Windows 7（SP1）/ Windows 8/Windows 8.1/Windows 10。

18.2.3 系统功能结构

防空大战游戏的系统功能结构如图18.1所示。

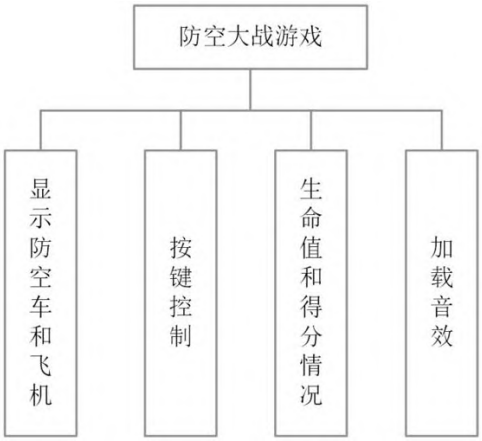


图18.1　系统功能结构

18.2.4　业务流程图

防空大战游戏的业务流程如图18.2所示。

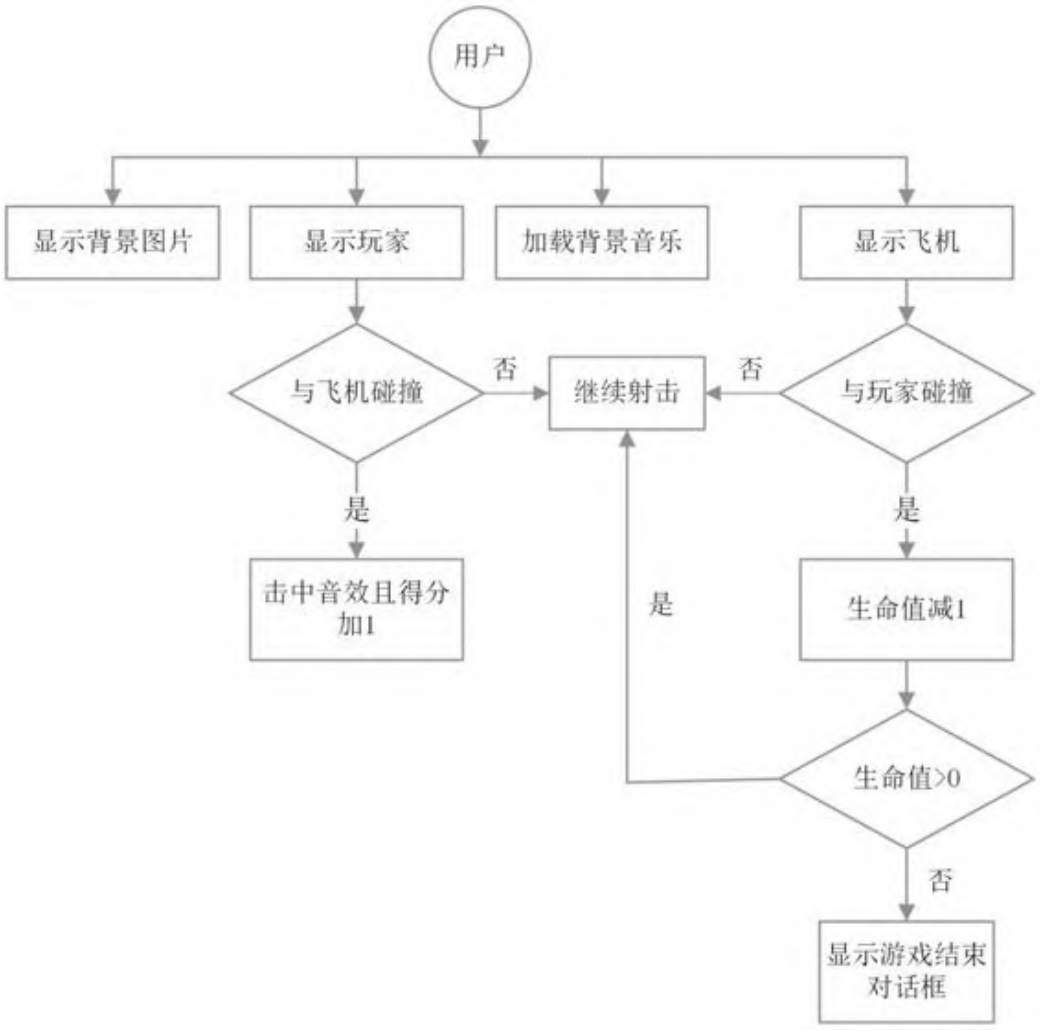


图18.2　业务流程图

18.2.5　系统预览

防空大战游戏的主界面如图18.3所示。游戏结束时会弹出对话框，如图18.4所示。



图18.3　游戏主界面

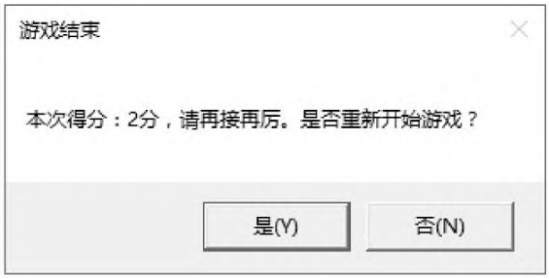


图18.4　游戏结束对话框

18.3　技术准备



18.3.1　EasyX图形库的下载与安装

EasyX是一款简单、易用的图形库，其最新版本可以从EasyX的官网（http://www.easyx.cn）下载。EasyX图形库可以应用于Visual C++6.0或者Visual Studio的不同版本中。

EasyX可以帮助C语言初学者快速上手图形和游戏编程。比如，可以用Visual Studio+EasyX绘制一架飞机或跑步的人物，也可以用它编写俄罗斯方块、贪吃蛇、飞机大战等小游戏。

注意

> 除EasyX外，比较常用的图形库还有OpenGL和QT。OpenGL是目前最常用的图形库，主流语言是C++、Java、JavaScript、C#和Objective-C。但OpenGL和QT两种库的绘图非常复杂，图形编程对数学的要求很高。

要想在Visual C++ 6.0上用EasyX绘图，首先需要下载并配置EasyX图形库。

（1）在浏览器中输入EasyX官网地址https://www.easyx.cn/，单击右侧上方的“下载EasyX”按钮，下载EasyX安装包，如图18.5所示。下载完毕的EasyX安装包将在左侧栏中显示。



图18.5　下载EasyX安装包

（2）双击安装包，打开软件安装向导，单击“下一步”按钮，安装向导会自动搜索本地安装的Visual C++/Visual Studio版本，如图18.6所示。单击“安装”按钮，耐心等待，相关头文件会自动导入对应头文件目录中，然后单击“关闭”按钮，即可完成EasyX的安装。

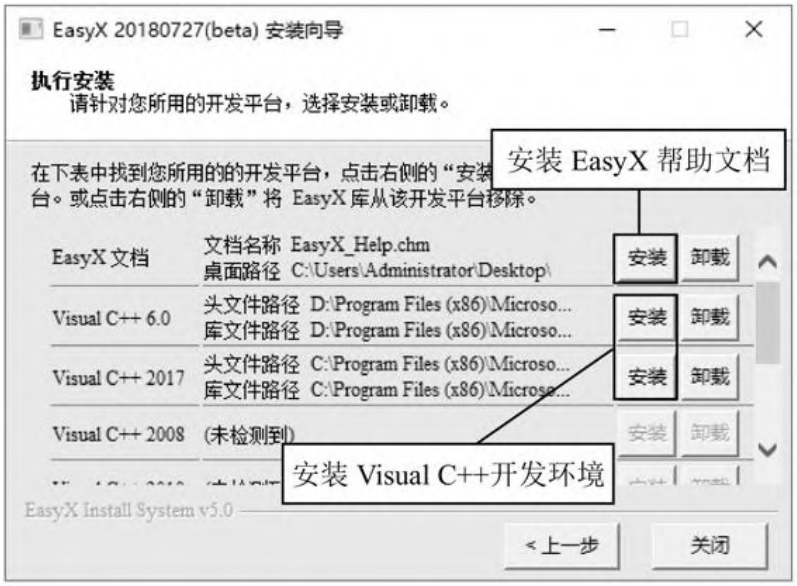


图18.6　安装EasyX

说明

EasyX的帮助文档对EasyX图形库的学习很有帮助，通过帮助文档可以查询一些常用的绘图函数，如图18.7所示。

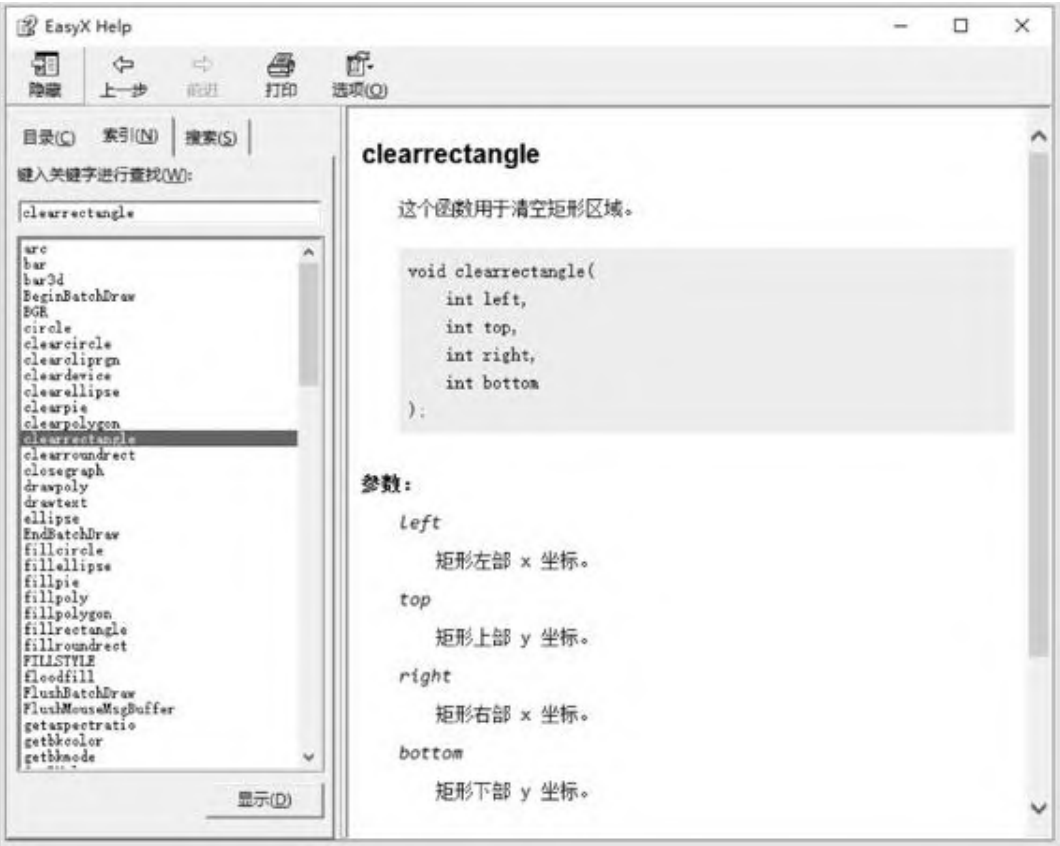


图18.7　EasyX帮助文档

18.3.2　EasyX库的主要函数

EasyX图形库也是一个函数库，其中包含了许多绘图函数（定义在easyx.h头文件中）。下面介绍本游戏开发将用到的一些函数。

1．颜色设置函数

EasyX中设置绘制颜色的函数有以下几个。

☑ setlinecolor（c）：设置线条颜色。

☑ setfillcolor（c）：设置填充颜色。

☑ setbkcolor（c）：设置背景颜色。

☑ setcolor（c）：设置前景颜色。

设置颜色时，颜色值可以使用常量来表示，如表18.1所示。

表18.1　颜色的常量值

颜色常数	数　值	含　义	颜色常数	数　值	含　义
BLACK	0	黑色	DARKGRAY	8	深灰
BLUE	1	蓝色	LIGHTBLUE	9	深蓝
GREEN	2	绿色	LIGHTGREEN	10	淡绿
CYAN	3	青色	LIGHTCYAN	11	淡青
RED	4	红色	LIGHTRED	12	淡红
MAGENTA	5	洋红	LIGHTMAGENTA	13	淡洋红
BROWN	6	棕色	YELLOW	14	黄色
LIGHTGRAY	7	淡灰	WHITE	15	白色

除此以外，也可以通过RGB三原色的值进行更多颜色设定，形式为RGB（r，g，b）。其中，r、g、b分别表示红色、绿色和蓝色，范围都是0~155。例如，RGB（0，0，0）表示黑色，RGB（255，255，255）表示白色，RGB（255，0，0）表示红色。

2．模式的初始化函数

不同的显示适配器有着不同的图形分辨率。既使是同一显示适配器，在不同的模式下分辨率也不相同。因此，在作图之前，必须根据显示适配器的种类将其设置为某种图形模式。未设置图形模式之前，计算机系统会默认屏幕为文本模式，此时所有图形函数均不能工作。

为屏幕设置图形模式可使用函数initgraph，它的一般形式如下：

```
initgraph(int *gdriver, int *gmode, char *path);
```

☑ gdriver：表示图形驱动器，它是一个整型值，常用的有EGA、VGA、PC3270等。如果开发者不知道计算机的显示适配器种类，可使用gdriver=DETECT语句，后再跟initgraph函数，就能自动检测显示器硬件并初始化图形界面。

☑ gmode：用来设置图形显示模式。不同的图形驱动程序，有不同的图形显示模式；同一种图形驱动程序下，也有几种图形显示模式。

☑ path：图形驱动程序所在的目录路径。如果驱动程序在用户当前目录下，则该参数可以为空。

退出图形状态的函数为closegraph，其一般形式如下：

```
closegraph(void);
```

用该函数后可退出图形状态，进入文本状态，并释放用于保存图形驱动程序和字体的系统内存。

3．jpg图片加载函数

加载jpg格式图片使用loadimage函数。loadimage函数有两种加载图片的方式，一种是从文件中读取图像，一般格式如下：

```
Loadimage(IMAGE* pDstImg, LPCTSTR pImgFile);
```

☑ IMAGE* pDstImg：保存图像的IMAGE对象指针。如果为NULL，表示图片将被读取至绘图窗口。

☑ LPCTSTR pImgFile：图片文件名。

例如，要加载C盘中的图片，主要代码如下：

```
initgraph(640, 480);
IMAGE img;                              //定义 IMAGE 对象
loadimage(&img, "C:\\test.jpg");        //读取图片到 img 对象中
```

另一种是从资源文件中获取图像，一般格式如下：

```
Loadimage(IMAGE* pDstImg, LPCTSTR pResType, LPCTSTR pResName
);
```

☑ IMAGE* pDstImg：保存图像的IMAGE对象指针。如果为NULL，表示图片将被读取至绘图窗口。

☑ LPCTSTR pResType：资源类型。

☑ LPCTSTR pResName：资源名称。

例如，要加载资源图像，主要代码如下：

```
initgraph(640, 480);
IMAGE img;
loadimage(&img, _T("IMAGE"), _T("Player"));
```

18.4 公共类设计



18.4.1 创建graphics.h头文件

在项目中创建一个头文件graphics.h，创建步骤和1.6.1节C++源文件的创建步骤类似，只不过这里创建的是.h头文件，如图18.8所示。

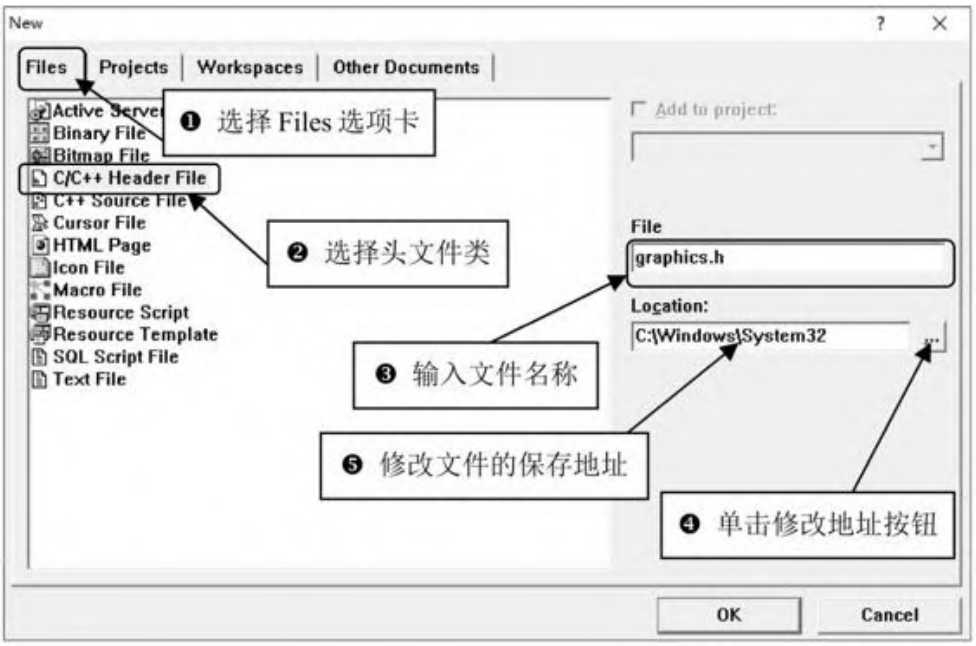


图18.8　创建头文件

创建完成之后，在头文件graphics.h中添加如下代码。

（1）初始化图像窗口，代码如下：

```
#include <easyx.h>
// BGI 格式的初始化图形设备，默认 640 x 480
HWND initgraph(int* gdriver, int* gmode, char* path);

void bar(int left, int top, int right, int bottom);              //绘制无边框填充矩形
void bar3d(int left, int top, int right, int bottom, int depth, bool topflag);      //绘制有边框三维填充矩形

void drawpoly(int numpoints, const int *polypoints);  //绘制多边形
void fillpoly(int numpoints, const int *polypoints);      //绘制填充的多边形

int getmaxx();                                                   //获取最大的宽度值
int getmaxy();                                                   //获取最大的高度值

COLORREF getcolor();                                             //获取当前绘图前景色
void setcolor(COLORREF color);                                   //设置当前绘图前景色
void setwritemode(int mode);                                     //设置前景的二元光栅操作模式
```

（2）使用接口函数，代码如下：

```
#if _MSC_VER > 1200
#define _EASYX_DEPRECATE(_NewFunc) __declspec(deprecated("This function is deprecated. Instead, us
this new function: " #_NewFunc ". See https://docs.easyx.cn/#" #_NewFunc " for details."))
#define _EASYX_DEPRECATE_OVERLOAD(_Func) __declspec(deprecated("This overload is deprecate
See https://docs.easyx.cn/#" #_Func " for details."))
#else
#define _EASYX_DEPRECATE(_NewFunc)
#define _EASYX_DEPRECATE_OVERLOAD(_Func)
#endif
```

（3）设置字体样式，代码如下：

```
_EASYX_DEPRECATE(settextstyle) void setfont(int nHeight, int nWidth, LPCTSTR lpszFace);
_EASYX_DEPRECATE(settextstyle) void setfont(int nHeight, int nWidth, LPCTSTR lpszFace, int nEscapemer
int nOrientation, int nWeight, bool bItalic, bool bUnderline, bool bStrikeOut);
_EASYX_DEPRECATE(settextstyle) void setfont(int nHeight, int nWidth, LPCTSTR lpszFace, int nEscapemer
int nOrientation, int nWeight, bool bItalic, bool bUnderline, bool bStrikeOut, BYTE fbCharSet, BYT
fbOutPrecision, BYTE fbClipPrecision, BYTE fbQuality, BYTE fbPitchAndFamily);
_EASYX_DEPRECATE(settextstyle) void setfont(const LOGFONT *font);       //设置当前字体样式
_EASYX_DEPRECATE(gettextstyle) void getfont(LOGFONT *font);             //获取当前字体样式

//以下为填充样式，更多填充样式可参考帮助文件
#define    NULL_FILL          BS_NULL
#define    EMPTY_FILL         BS_NULL
#define    SOLID_FILL         BS_SOLID
//普通一组
// /// 填充（对应 BGI 的 LTSLASH_FILL）
#define    BDIAGONAL_FILL     BS_HATCHED, HS_BDIAGONAL
#define    CROSS_FILL         BS_HATCHED, HS_CROSS                  // +++ 填充
// xxx 填充（heavy cross hatch fill）（对应 BGI 的 XHTACH_FILL）
#define    DIAGCROSS_FILL     BS_HATCHED, HS_DIAGCROSS
// xxx 填充（对应 BGI 的 WIDE_DOT_FILL）
#define    DOT_FILL           (BYTE*)"\x80\x00\x08\x00\x80\x00\x08\x00"
#define    FDIAGONAL_FILL     BS_HATCHED, HS_FDIAGONAL              // \\\ 填充
#define    HORIZONTAL_FILL    BS_HATCHED, HS_HORIZONTAL             // === 填充
#define    VERTICAL_FILL      BS_HATCHED, HS_VERTICAL               // ||| 填充
//密集一组
#define    BDIAGONAL2_FILL    (BYTE*)"\x44\x88\x11\x22\x44\x88\x11\x22"
//（对应 BGI 的 fill HATCH_FILL）
#define    CROSS2_FILL        (BYTE*)"\xff\x11\x11\x11\xff\x11\x11\x11"
#define    DIAGCROSS2_FILL    (BYTE*)"\x55\x88\x55\x22\x55\x88\x55\x22"
//（对应 BGI 的 CLOSE_DOT_FILL）
#define    DOT2_FILL          (BYTE*)"\x88\x00\x22\x00\x88\x00\x22\x00"
#define    FDIAGONAL2_FILL    (BYTE*)"\x22\x11\x88\x44\x22\x11\x88\x44"
#define    HORIZONTAL2_FILL(BYTE*)"\x00\x00\xff\x00\x00\x00\xff\x00"
#define    VERTICAL2_FILL     (BYTE*)"\x11\x11\x11\x11\x11\x11\x11\x11"
//粗线一组
#define    BDIAGONAL3_FILL    (BYTE*)"\xe0\xc1\x83\x07\x0e\x1c\x38\x70"    //对应 BGI 的 SLASH_FILL
#define    CROSS3_FILL        (BYTE*)"\x30\x30\x30\x30\x30\x30\xff\xff"
#define    DIAGCROSS3_FILL    (BYTE*)"\xc7\x83\xc7\xee\x7c\x38\x7c\xee"
#define    DOT3_FILL          (BYTE*)"\xc0\xc0\x0c\x0c\xc0\xc0\x0c\x0c"
#define    FDIAGONAL3_FILL    (BYTE*)"\x07\x83\xc1\xe0\x70\x38\x1c\x0e"
//（对应 BGI 的 LINE_FILL）
#define    HORIZONTAL3_FILL(BYTE*)"\xff\xff\x00\x00\xff\xff\x00\x00"
#define    VERTICAL3_FILL     (BYTE*)"\x33\x33\x33\x33\x33\x33\x33\x33"
//其他
//（对应 BGI 的 INTERLEAVE_FILL）
#define    INTERLEAVE_FILL    (BYTE*)"\xcc\x33\xcc\x33\xcc\x33\xcc\x33"
```

18.4.2 源文件公设计

1．添加函数库

在完整的源文件中不仅需要引入前面自定义的函数库，还需要引入一些其他函数库，代码如下：

```
#include <graphics.h>                                    //引入函数库
#include <vector>
#include <time.h>
#include <math.h>
#include <mmsystem.h>
#pragma comment(lib,"winmm.lib")
#pragma comment (lib,"MSIMG32.lib")
using namespace std;
using std::vector;                                       //引用容器
```

2．函数声明

防空大战游戏使用的函数需要事先声明，具体代码如下：

```
void setpicture(int x, int y, IMAGE img);                                    //设计透明贴图函数
void sleep(int ms);                                                          //精确延时
bool collision(int x1, int y1, int w1, int h1, int x2, int y2, int w2, int h2);  //矩形碰撞检测函数
void createmissibles();                                                      //创建防空导弹
void createplane();                                                          //创建飞机
void loadpicture();                                                          //加载图片资源
void drawing();                                                              //每帧绘图
int startGame();                                                             //开始游戏函数
```

3．全局变量定义

全局变量可以被文件中的所有函数共用。它的有效范围从定义变量的位置开始，到本源文件结束。定义全局变量的具体代码如下：

```
int playerx = 0;                                         //玩家的 X 坐标
int playery = 333;                                       //玩家的 Y 坐标
int firereload = 5;                                      //装弹剩余时间
int strength = 50;                                       //玩家的生命值
int score = 0;                                           //玩家的得分
```

4．图片对象声明以及宏定义

要想加载图片，需要先定义图像对象。定义图像对象的代码如下：

```
IMAGE buffer(500, 400);                    //IMAGE 图片尺寸
IMAGE background;                          //背景图片
IMAGE player;                              //玩家图片
IMAGE missible;                            //防空导弹图片
IMAGE planes[3];                           //3 种飞机图片
IMAGE planebomb;                           //炸弹图片
```

同时，游戏需要使用键盘中的按键控制玩家的位置以及防空导弹的发射，因此需要以下的宏定义：

```
#define KEY_DOWN(vk_c) (GetAsyncKeyState(vk_c)&0x8000?1:0)
//设置按键情况
```

18.5 游戏主窗体设计



18.5.1 模块概述

游戏的主窗体中需要显示背景图片、防空车图片、3种飞机图片、炸弹图片以及防空炸弹图片，如图18.9所示。

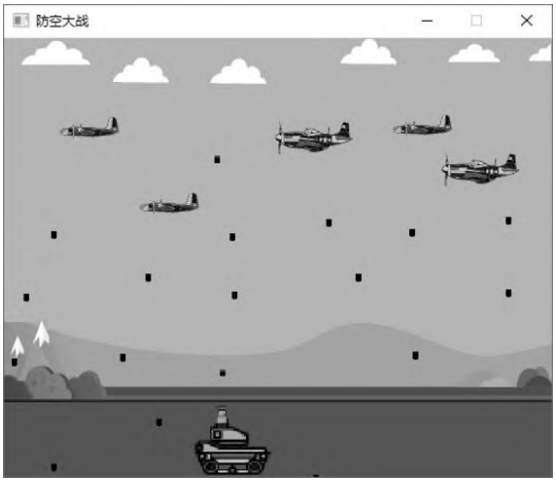


图18.9 游戏主窗体

18.5.2 创建防空导弹

准备资源图片防空导弹，然后在Visual C++ 6.0创建的文件中导入资源文件，步骤如下：

（1）在环境菜单中选择Insert→Resource命令，如图18.10所示，将打开如图18.11所示的添加资源对话框。

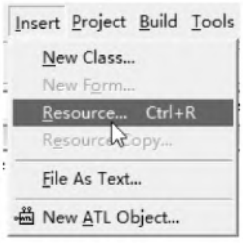


图18.10 添加资源

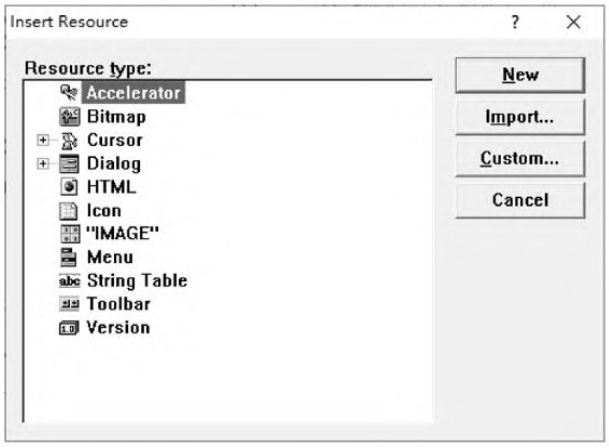


图18.11 添加资源对话框

（2）选择import导入命令，在打开的对话框中选择防空导弹图片，如图18.12所示。

（3）双击图片之后，会出现如图18.13所示的对话框，在“资源类型”文本框中输入“IMAGE”，单击“确定”按钮，即可在项目中看到新增加的图片资源。

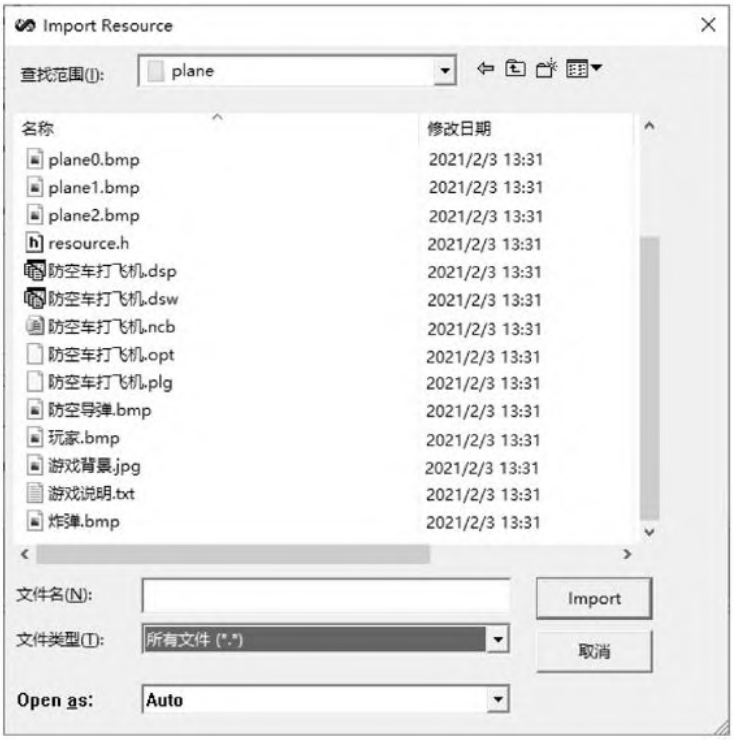


图18.12　导入防空导弹图片资源



图18.13　导入防空导弹图片

（4）右击该资源，在弹出的快捷菜单中选择“属性”命令，修改ID为MISS。

（5）添加完图片资源之后，在源文件中编写创建防空导弹的代码，具体如下：

```
struct PLAYERMISSIBLE                    //玩家防空导弹坐标
{
    int x;
    int y;
};
vect or<PLAYERMISSIBLE> miss;            //声明防空导弹容器
void createmissibles()                   //创建一颗防空导弹
{
    PLAYERMISSIBLE playermiss;
    playermiss.x = playerx + 12;
    playermiss.y = playery - 4;
    miss.push_back(playermiss);
}
```

18.5.3 创建炸弹

添加炸弹资源的步骤与添加防空导弹资源的步骤基本相同，唯一不同的是右击资源并在弹出的快捷菜单中选择“属性”命令后，需要设置ID为BOMB。

添加完资源后，在源文件中编写创建炸弹的代码，具体如下：

```
struct BOMB                              //炸弹
{
    int x;
    int y;
};
vector<BOMB> bomb;                       //声明炸弹容器
void createbomb(PLANE p)                 //创建一颗炸弹
{
    BOMB b;
    b.x = p.x + threeplane[p.type].width / 2 - 4;
    b.y = p.y + threeplane[p.type].height+2;
    bomb.push_back(b);
}
```

18.5.4 创建飞机

添加飞机资源的步骤不再赘述，注意最后要修改ID为PLANE0、PLANE1和PLANE2。

添加完资源后，在源文件中编写创建飞机的代码，具体如下：

```
struct THREEPLANE                                   //飞机类型
{
    int speed;                                      //飞机速度
    int width;                                      //飞机宽度
    int height;                                     //飞机高度
} threeplane[3] = { {15,85,25},{12,79,22},{8,100,45} };

struct PLANE                                        //飞机
{
    int x;
    int y;
    int type;
    int reload;                                     //还有多少帧重新扔炸弹
};

vector<PLANE> plane;                                //声明飞机容器
void createplane()                                  //创建一架飞机
{
    PLANE newplane;
    newplane.x = 500;
    newplane.y = rand() % 100 +40;
    newplane.type = rand() % 3;
    newplane.reload = 10;                           //炸弹发射间隔时间为 10 帧
    plane.push_back(newplane);
}
```

18.5.5 加载图片

按照同样的步骤添加背景和玩家图片资源，添加完之后分别修改ID为BG和PLAYER。

添加完资源后，在源文件中编写加载图片代码，具体如下：

```
//设计透明图片函数
void setpicture(int x, int y, IMAGE img)
{
	HDC dstDC = GetImageHDC(&buffer);		//获取图片缓冲区
	HDC srcDC = GetImageHDC(&img);		//获取图片资源
	//目标设备环境函数，对指定的源设备环境中的矩形区域像素的颜色数据进行位块转换
	TransparentBlt(dstDC, x, y, img.getwidth(), img.getheight(), srcDC, 0, 0, img.getwidth(), img.getheight
RGB(40, 112, 162));
}

//加载图片资源
void loadpicture()
{
	loadimage(&background, "IMAGE", "BG");
	loadimage(&player, "IMAGE", "PLAYER");
	loadimage(&planebomb, "IMAGE", "BOMB");
	loadimage(&missible, "IMAGE", "MISS");
	char filename[10];
	for (int i = 0; i < 3; i++)
	{
		sprintf_s(filename, "PLANE%d", i);
		loadimage(&planes[i], "IMAGE", filename);
	}
}
```

18.5.6 绘图设计

接下来需要在控制台上显示前面添加的各类资源，具体绘图代码如下：

```
//绘图设计
void drawing()
{
    char info[25];
    SetWorkingImage(&buffer);                    //先在缓冲区绘图
    putimage(0, 0, &background);
    setpicture(playerx, playery, player);        //显示防空车
    vector<PLAYERMISSIBLE>::iterator it1;
    vector<PLANE>::iterator it2;
    vector<BOMB>::iterator it3;
    for (it1 = miss.begin(); it1 != miss.end(); it1++)
    {
        setpicture(it1->x, it1->y, missible);    //显示防空导弹
    }
    for (it2 = plane.begin(); it2 != plane.end(); it2++)
    {
        setpicture(it2->x, it2->y, planes[it2->type]);   //显示飞机
    }
    for (it3 = bomb.begin(); it3 != bomb.end(); it3++)
    {
        setpicture(it3->x, it3->y, planebomb);   //显示炸弹
    }
SetWindowText(GetHWnd(), "防空大战");            //设置窗口标题文字，GetHWnd()获取窗口句柄
putimage(0, 0, &buffer);                         //一次性绘制缓冲区内容，以消除闪烁
}
```

18.6 碰撞检测设计



18.6.1 模块概述

玩家是否得分，要看防空车发射的子弹是否与飞机发生碰撞。一旦发生碰撞，玩家就会得1分。同时，玩家的生命值也需要通过碰撞来计算是否减分。一旦炸弹与防空车发生碰撞，生命值就会减去1分，直到生命值为0时游戏结束。由此看来，碰撞检测是计分的重要依据。

18.6.2 代码实现

本程序的碰撞检测具体代码如下：

```
//矩形碰撞检测函数
bool collision(int x1, int y1, int w1, int h1, int x2, int y2, int w2, int h2)
{
    if ((abs((x1 + w1 / 2) - (x2 + w2 / 2)) < (w1 + w2) / 2) && abs((y1 + h1 / 2) - (y2 + h2 / 2)) < (h1 + h2) / 2)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

18.7　开始游戏设计



18.7.1　模块概述

在开始游戏模块包含加载音乐、3种飞机随机出现、按键设计、生命值和得分统计、游戏结束界面以及重新开始游戏等功能部分。完整的游戏界面如图18.14所示。



图18.14　开始游戏主界面

18.7.2　加载音乐

为了增加游戏的趣味性，在游戏过程中添加了背景音乐和特殊音效。游戏的背景音乐，从进入游戏便开始播放。当玩家击中飞机时，会播放击中音效；击中飞机消失后，击中音效停止播放。要实现音乐加载功能，需要使用mciSendString函数。

mciSendString函数用来播放mp3文件，使用时需要在程序前加上宏定义“#pragma comment（lib，"winmm.lib"）”。

mciSendString函数的一般形式如下：

```
MCIERROR mciSendString(
    LPCTSTR lpszCommand,          //MCI 命令字符串
    LPTSTR lpszReturnString,      //存放反馈信息的缓冲区
    UINT cchReturn,               //缓冲区的长度
    HANDLE hwndCallback           //回调窗口的句柄，一般为 NULL
);
```

平时只需要用到第一个参数，另外3个参数设置为NULL、0、NULL即可。

第一个参数是多媒体命令字符串，不区分大小写。以操作back.mp3音乐文件为例，第一个参数可以为以下4种。

- ☑ open back.mp3：打开音乐文件。可以使用alias指定别名，之后的代码中就可以通过别名访问该音乐了。当然，并不是必须要指定别名，每次通过文件名访问也是可以的。open后面的back.mp3用绝对路径或相对路径都可以，最好是将mp3文件放到源文件的文件夹内，这样可以省略路径。
- ☑ play back.mp3：开始播放音乐。如果需要音乐重复播放，可以在音乐文件名后面加上repeat。
- ☑ stop back.mp3：停止播放音乐。
- ☑ close back.mp3：关闭音乐文件。

注意

如果文件名中有空格，需要用双引号将文件名括起来，并且需要注意文件路径中“\”和双引号的转义，例如：

```
mciSendString("open \"E:\\My Music\\觅香 栗先达.mp3\" alias
mymusic", NULL, 0, NULL);
```

如果需要同时播放多个音乐，需要为不同的音乐指定不同的别名，然后分别进行操作。

说明

PlaySound函数也可以用来播放声音，由于其不支持mp3/wma格式音乐，这里就不多做介绍了。

下面介绍背景音乐和击中音效的代码实现。

1．播放背景音乐

在本项目中，默认播放背景音乐。在开始游戏函数中添加播放音乐代码，具体如下：

```
mciSendString("play bj.mp3 repeat", NULL, 0, NULL); //播放背景音乐
```

2．播放击中音效

在本项目中，当防空导弹与飞机发生碰撞时，就会播放击中音效。飞机消失之后，就会停止音效播放。要实现这个功能，需要在开始游戏函数的遍历防空导弹图像的代码段中添加如下代码：

```
//判断玩家发生的防空导弹是否击中飞机，如果击中
if (collision(it1->x, it1->y, 5, 7, it2->x, it2->y, threeplane[it2->type].width, threeplane[it2->type].height))
{
    mciSendString("play ss.mp3", NULL, 0, NULL);      //播放击中飞机音效
    it1 = miss.erase(it1);
    it2 = plane.erase(it2);
    yon = true;
    score += 1;
}
else                                                  //如果没击中
{
    mciSendString("close ss.mp3", NULL, 0, NULL);     //关闭击中飞机音效
    ++it1;
}
```

说明

挑选音效时，需要选择时长1~2s的，不然会有拖沓的感觉，影响游戏代入感。

18.7.3　飞机随机出现

游戏中用到了3种飞机类型，这3种飞机类型随机出现，炸弹也是随机出现。也就是说，游戏需要显示玩家、炸弹、防空导弹以及3种飞机的每帧图片，实现这一功能的具体代码如下：

```
int startGame()
{
	mciSendString("play bj.mp3 repeat", NULL, 0, NULL);	//播放背景音乐
	srand((unsigned)time(NULL));	//设置随机数种子
	loadpicture();	//加载图片
	initgraph(500, 400);	//初始化背景
	vector<PLAYERMISSIBLE>::iterator it1;	//创建防空导弹迭代器
	vector<PLANE>::iterator it2;	//创建飞机迭代器
	vector<BOMB>::iterator it3;	//创建炸弹迭代器
	gamestart:	//开始游戏标识符
	bool yon = false;
	while (true)
	{
		if (strength < 1)	//生命值小于 1，退出程序
		{
			break;
		}
		if (plane.size() < 3)
		{
			createplane();	//创建飞机
		}
		for (it1 = miss.begin(); it1 != miss.end();)	//发射防空导弹
		{
			it1->y -= 15;
			if (it1->y < 0)
			{
				it1 = miss.erase(it1);
			}
			else
			{
				++it1;
			}
		}
		for (it2 = plane.begin(); it2 != plane.end();)	//显示飞机
		{
			it2->x -= threeplane[it2->type].speed;
			if (it2->x + threeplane[it2->type].width < 0)
			{
				it2 = plane.erase(it2);
			}
			else
			{
				if (it2->reload == 0)
				{
					it2->reload = 10;
					createbomb(*it2);
				}
				else
				{
					it2->reload -= 1;
```

```
                    }
                    ++it2;
                }
            }
            for (it2 = plane.begin(); it2 != plane.end();)                          //遍历飞机每帧图像
            {
                for (it1 = miss.begin(); it1 != miss.end();)                        //遍历防空导弹的每帧图像
                {
                    //判断玩家发出的防空导弹是否击中飞机，如果击中
                    if (collision(it1->x, it1->y, 5, 7, it2->x, it2->y, threeplane[it2->type].width, threeplane[it2->type].height
                    {
                        mciSendString("play ss.mp3", NULL, 0, NULL);            //播放击中飞机音效
                        it1 = miss.erase(it1);
                        it2 = plane.erase(it2);
                        yon = true;
                        score += 1;
                    }
                    else                                                        //如果没击中
                    {
                        mciSendString("close ss.mp3", NULL, 0, NULL);           //关闭击中飞机音效
                        ++it1;
                    }
                }
                if (yon == false)
                {
                    it2++;
                }
                else
                {
                    yon = false;
                }

            }

            for (it3 = bomb.begin(); it3 != bomb.end();)                        //遍历随机显示炸弹的每帧图像
            {
                it3->y += 5;
                if (it3->y > 400)
                {
                    it3 = bomb.erase(it3);
                }
                else
                {
                    ++it3;
                }
            }
            for (it3 = bomb.begin(); it3 != bomb.end();)                        //遍历炸弹的每帧图像
            {
                if (collision(it3->x, it3->y, 5, 7, playerx, playery, 58, 49))  //判断是否发生炸弹碰撞
                {
                    it3 = bomb.erase(it3);
                    strength -= 1;
                }
                else
                {
                    ++it3;
                }
            }
        }
        return 0;
}
```

18.7.4　按键设计

本游戏需要通过键盘上的左右按键控制防空车的左右移动，通过空格键控制防空导弹的发射。实现此功能需要在startGame函数中添加以下代码：

说明

在18.7.3节的startGame函数的倒数第二个“}”前添加实现按键功能的代码。

```
if (KEY_DOWN(VK_LEFT) && playerx >= 10)          //判断按键向左
{
    playerx -= 10;                               //防空车坐标 X 减 10，表示左移
}
if (KEY_DOWN(VK_RIGHT) && playerx <= 384)        //判断按键向右
{
    playerx += 10;                               //防空车坐标 X 加 10，表示右移
}
if (KEY_DOWN(VK_SPACE) && firereload == 0)       //判断玩家按空格键
{
    createmissibles();                           //创建防空导弹
    firereload = 5;
}
if (firereload >= 1)                             //如果有时间
{
    firereload -= 1;                             //继续执行减 1
}
drawing();                                       //绘图
sleep(33);
```

18.7.5　生命值和得分统计

玩家的防空导弹与飞机发生碰撞，就会得1分。炸弹与玩家发生碰撞，玩家生命值就会减1（生命值的初始值是50）。在游戏主窗口中，显示的生命值和得分情况如图18.15所示。



图18.15　分数显示

由图18.15可以看出，生命值和得分情况的背景是黄色的，字体是红色、黑体，字体大小是20磅。要实现生命值和得分显示功能，需要在绘图设计函数drawing中添加如下代码：

```
setbkcolor(YELLOW);                                          //字体背景设置透明
settextstyle(20, 0, _T("黑体"));                             //字体样式，高度 20，黑体
settextcolor(RGB(255, 51, 68));
sprintf_s(info, "生命值：%d 得分：%d", strength, score);
outtextxy(150, 0, info);                                     //显示游戏数据
SetWorkingImage();                                           //设置绘图目标为绘图窗口
```

说明

在绘图设计函数drawing中的putimage代码前添加如上生命值和得分设计代码。

18.7.6 游戏结束界面

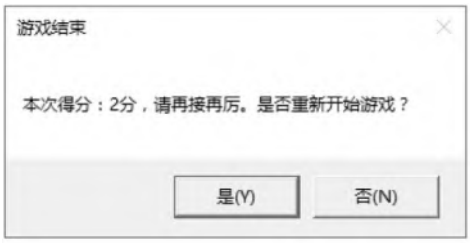


图18.16 游戏结束对话框

当玩家的生命值为0时，就会结束游戏，同时弹出结束游戏对话框，如图18.16所示。

要实现此功能，需要在startGame函数的按键设计代码之后添加如下代码：

```
char result[100];                                                //显示结果
sprintf_s(result, "本次得分：%d 分，请再接再厉。是否重新开始游戏？", score);    //输出提示窗口
```

18.7.7 重新开始游戏

在游戏结束对话框中有两个按钮，一个是“是”，一个是“否”。当玩家单击“是”按钮时，就会重新开始游戏；当玩家单击“否”按钮时，就会退出游戏。要实现此功能，需要在startGame的游戏结束代码后添加如下代码：

```
//判断是否为重新开始游戏
if (MessageBox(GetHWnd(), result, "游戏结束", MB_YESNO) == IDYES)
{
    //清屏 重新开始游戏
    playerx = 0;
    playery = 333;
    strength = 50;
    score = 0;
    firereload = 5;
    miss.clear();
    plane.clear();
    bomb.clear();
    goto gamestart;                                    //回到 gamestart 重新开始游戏
}
```

18.8 为游戏应用添加图标



18.8.1 添加图标概述

本项目为应用程序添加了图标，该图标可以显示在游戏界面的标题栏，也可以显示在系统的任务栏中，如图18.17所示。

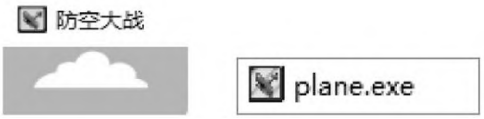
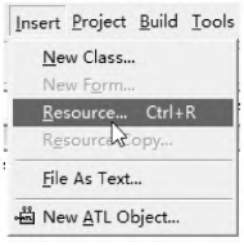


图18.17 游戏图标

18.8.2 实现步骤

通过Visual C++ 6.0可以为应用程序添加图标，步骤如下。



（1）在项目名称上右击，在弹出的快捷菜单中选择Insert→Resource命令，打开添加资源对话框，如图18.18所示。

（2）因为图标资源的文件类型是Icon，所以在添加资源界面中，选择Icon类型，然后单击Import按钮导入，如图18.19所示。

图18.18　选择命令

（3）在导入资源对话框中，找到飞机的Icon文件，单击Import按钮，如图18.20所示。

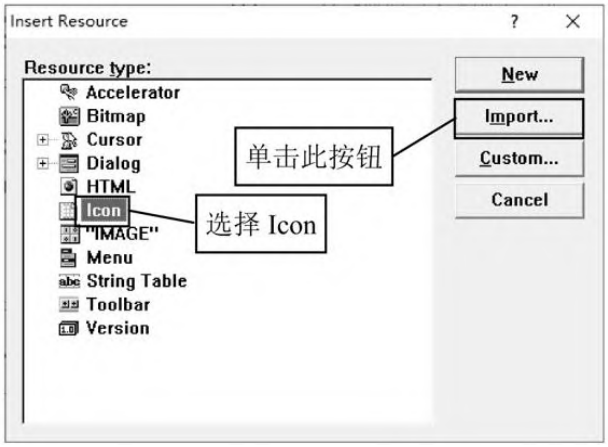


图18.19　添加资源对话框



图18.20　打开资源

（4）如果想对图标进行进一步的设计，可以在如图18.21所示的窗口中对图标进行设计。

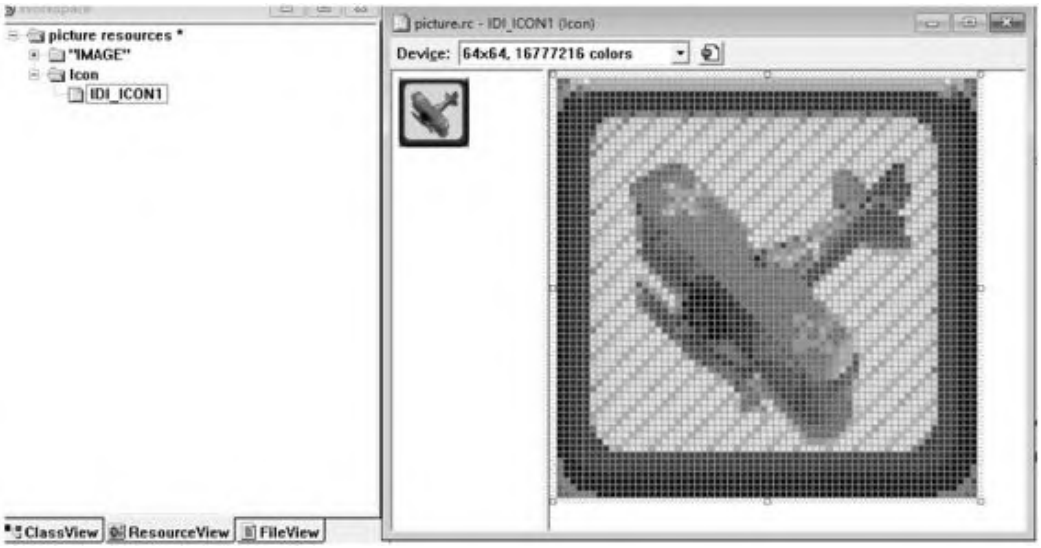


图18.21　设计图标

（5）运行项目，就会发现图标已经添加成功，如图18.22所示。



图18.22　添加图标游戏界面

附录　ASCII码对照表

ASCII 值	缩写/字符	解　　释
0	NUL（null）	空字符（\0）
1	SOH（star to fhanding）	标题开始
2	STX（star to ftext）	正文开始
3	ETX（end of text）	正文结束
4	EOT（end of transmission）	传输结束
5	ENQ（enquiry）	请求
6	ACK（acknowledge）	收到通知
7	BEL（bell）	响铃（\a）
8	BS（backspace）	退格（\b）
9	HT（horizontal tab）	水平制表符（\t）
10	LF（NL）（linefeed,newline）	换行键（\n）
11	VT（verticaltab）	垂直制表符
12	FF（NP）（formfeed,newpage）	换页键（\f）
13	CR（carriagereturn）	回车键（\r）
14	SO（shift out）	不用切换
15	SI（shift in）	启用切换
16	DLE（data link escape）	数据链路转义
17	DC1（device control1）	设备控制 1
18	DC2（device control2）	设备控制 2
19	DC3（device control3）	设备控制 3
20	DC4（device control4）	设备控制 4
21	NAK（negative acknowledge）	拒绝接收
22	SYN（synchronousidle）	同步空闲
23	ETB（end of trans.block）	传输块结束
24	CAN（cancel）	取消
25	EM（end of medium）	介质中断
26	SUB（substitute）	替补
27	ESC（escape）	溢出
28	FS（file separator）	文件分割符
29	GS（group separator）	分组符
30	RS（record separator）	记录分离符

续表

ASCII 值	缩写/字符	解　　释
31	US（unit separator）	单元分隔符
32	SP（space）	
33	!	
34	"	
35	#	
36	$	
37	%	
38	&	
39	'	
40	(	
41	)	
42	*	
43	+	
44	,	
45	-	
46	.	
47	/	
48	0	
49	1	
50	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	
59	;	
60	<	
61	=	
62	>	
63	?	
64	@	
65	A	
66	B	
67	C	
68	D	
69	E	

续表

ASCII 值	缩写/字符	解　　释
70	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
80	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
90	Z	
91	[	
92	\	
93	]	
94	^	
95	_	
96	`	
97	a	
98	b	
99	c	
100	d	
101	e	
102	f	
103	g	
104	h	
105	i	
106	j	
107	k	
108	l	
109	m	
110	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
120	x	
121	y	
122	z	
123	{	
124	|	
125	}	
126	~	
127	DEL（Delete）	