国内首本系统论述ACM/ICPC竞赛的C语言程序设计教程！

ACM/ICPC资深教练授课经验总结，融合大量优秀竞赛作品实战技能！

清华 开发者书库

C Language Programming
Practical Guide for ACM/ICPC Contest

C语言程序设计

零基础ACM/ICPC竞赛实战指南

王建芳◎编著
Wang Jianfang

清华大学出版社

作者简介

王建芳

博士，副教授，硕士研究生导师，现任河南理工大学ACM/ICPC总教练；主要从事人工智能、数据挖掘和智能计算算法等方向的研究工作，具有丰富的系统研究开发经验和扎实的理论基础知识。长期指导学生参加各种程序算法设计类竞赛，并多次获得省级及以上奖励；曾多次获得相关赛事的“优秀指导教师”称号。

清华开发者书库

C 语言程序设计——零基础 ACM/ICPC 竞赛实战指南

王建芳　编著
Wang Jianfang

清华大学出版社
北 京

内 容 简 介

本书是专为C语言爱好者及ACM/ICPC参赛者编写的入门级教程，针对C语言学习过程中普遍存在的重理论轻实践、重语法轻编程的现象，通过贯穿全书的大量实例来介绍C语言编程的方法和技巧。全书分为三个部分：第一部分介绍C语言的基础性语法，包括标准程序框架、数据类型和控制结构；第二部分介绍了常见的OJ(Online Judge)平台、使用方法及OJ系统的基本输入与输出的常见类型；第三部分通过实例介绍了数组、函数和结构体编程过程中常用的知识点。

本书可以作为“C语言程序设计”课程的基础教材，也可作为参加ACM/ICPC竞赛的指导用书，并可作为各高校和相关培训机构的教学参考书。

图书在版编目(CIP)数据

C语言程序设计：零基础ACM/ICPC竞赛实战指南/王建芳编著.--北京：清华大学出版社，2015
(清华开发者书库)
ISBN 978-7-302-40116-2

Ⅰ. ①C…　Ⅱ. ①王…　Ⅲ. ①C语言－程序设计　Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2015)第089501号

责任编辑：盛东亮
封面设计：李召霞
责任校对：白　蕾
责任印制：宋　林

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
课件下载：http://www.tup.com.cn,010-62795954
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：186mm×240mm　　**印　张**：13.75　　**字　　数**：308千字
版　　次：2015年6月第1版　　**印　　次**：2015年6月第1次印刷
印　　数：1～2500
定　　价：35.00元

产品编号：063522-01

前言
PREFACE

C语言功能丰富、表达能力强、使用灵活方便，20世纪90年代以来，C语言迅速在全世界普及推广。C语言具有高级语言的优点又有低级语言的特性，既适合编写操作系统软件，又能方便地开发领域应用软件。目前，C语言程序设计已经成为最为广泛的一门程序设计课程。依据基于世界范围内的资深软件工程师和第三方供应商提供作为指数的TIOBE开发语言排行榜（每月发布一次），C语言排名一直名列前茅。C语言是实践性很强的一门课程，必须不断地练习编程。在信息技术飞速发展的今天，如何将理论与实践有机结合，有效地推进素质教育和高水平人才培养，是新时期IT人才面临的新课题。

ACM国际大学生程序设计竞赛（ACM International Collegiate Programming Contest，ACM/ICPC）是由美国计算机协会（Association for Computing Machinery，ACM）主办，其目的是使大学生充分展示分析问题和运用计算机解决问题的能力。ACM/ICPC作为一项世界性的竞赛活动，自1970年开始至今，是世界范围内历史最悠久、规模最大的程序设计竞赛。正好迎合了当今社会对创新性IT人才的需求，竞赛较全面地考验学生对知识的综合运用能力、创造性地分析解决问题能力，所以在IT界具有超凡的影响力。该项赛事极大地提高了参赛同学的学习热情、实践动手能力、团队合作能力和创造创新能力。ACM/ICPC在线评判（Online Judge，OJ）系统是该项比赛的评判事务处理平台，提供了一个基于B/S结构的多用户在线系统，允许用户在线提交自己的解题代码，系统自动编译运行给出评判结果，并根据用户解题数和用时综合排出名次。

针对C语言学习过程中普遍存在的重理论轻实践、重语法讲解轻编程思想的现象，本书将“A+B”的示例程序贯穿全书，将ACM竞赛平台OJ系统用于C语言的教学讲解和自学过程中。全书分为三个部分：第一部分为C语言的基础性语法，包括标准程序框架，数据类型和控制结构；第二部分针对常见的OJ平台、使用方法及OJ系统的基本输入与输出等常见类型进行讲解；第三部分为数组、函数和结构体。

本书主要特点：①所讲解的程序框架是ACM/ICPC通用的标准框架；②采用实例讲解方法引出理论；③每个例程均已通过测试，确保能够正确编译并运行；④详细讲解如何使用OJ系统进行编程实践。

本书适用人群：①大学本科一年级没有学过C语言的学生；②已学过或正在学C语言，但对已学内容不得要领的学生；③有强烈参加ACM/ICPC竞赛愿望的学生；④大学本科四年级考取研究生，复试阶段需要上机复试C语言的学生；⑤有强烈提高自己编程能力

欲望，但苦于找不到合适训练方法和习题的读者。

本书程序运行的操作系统为 Windows 7，计算机硬件配置为 Intel(R) Core(TM) i5 CPU M480 @2.67GHz，系统类型为 64 位操作系统。

本书作者是长期从事 ACM/ICPC 竞赛指导和 C 语言教学的一线教师，同时也一直致力于 C 语言教学改革，近三年来所带学生在计算机软件类学科竞赛中获得省级以上奖励三百余人次。

在本书的编写过程中，部分题目参考或改编自杭州电子科技大学和北京大学的在线评判(OJ)系统，在此表示感谢！

由于作者能力和水平所限，加之时间仓促，书中不足之处恳请广大专家、读者批评指正！在 C 语言程序设计竞赛相关书籍中，希望本书抛出的“砖”能够引出更多的“玉”！作者邮箱 hpuacm@126.com。

编著者

2015 年 2 月

目 录

CONTENTS

第 1 章 死记硬背

作为全书的开篇，本章讲述以下内容，为即将开始的 C 语言程序设计做必要的准备工作。

（1）为什么要学习 C 语言。

（2）程序设计与盖房子的类比。

（3）完成并记住第一个完整的程序框架。

1.1 引子

为什么每个程序开发者都应该学习 C 语言？

每个程序开发者在编程生涯中都应该学习 C 语言，因为它有太多难以忽视的好处。除了会给你提供更多的工作机会，C 语言还会教给你更多关于计算机的相关知识。C 语言提供的裨益，简单列举如下：

（1）相比较其他的编程语言（像 C++，Java），C 语言是低级语言。总体来说，低级的编程语言可以让你更好地了解计算机内部构造和原理。

（2）C 语言能够让你深入系统底层。Windows、UNIX、Linux、Mac、Android、OS/2 等操作系统，没有一个例外，都是用 C 语言编写的。如果不懂 C 语言，怎么可能深入到这些操作系统中去呢？更不要说去写它们的内核程序了。

（3）C 语言程序比其他语言写的程序，实现相同的功能，用的代码行数更少，而它带来的运行效率却更高。

（4）如果你学习过 C 语言，就能学习现在任何高级编程语言。因为所有高级语言都是以 C 语言为基础（像 Java、C++、C＃等）。所以，如果你想在程序设计方面有所建树，就必须去学它。

（5）很多新型语言都是衍生自 C 语言，例如 C++，Java，C＃及开发基于 Android 的应用程序。掌握了 C 语言，可以说就掌握了很多门语言，经过简单的学习，就可以用新型的语言进行开发，这再一次验证了 C 语言是程序设计的重要基础。

（6）任何里面有微处理器的设备都支持 C 语言。从微波炉到智能手机，都是由 C 语言技术来推动的。

（7）即使当今比较知名的公司招聘程序员，在部分考试中都要考查C语言的基本知识和技能。想加入IT行业，就一定要掌握好C语言，因为任何算法都可以用C语言实现。

（8）目前，计算机科学与技术专业的大四本科生所参加的研究生入学考试，在复试的机试环节中大部分高校也是通过OJ系统来提交代码，而这些代码都能够用C语言来实现。

C语言在计算机专业中所处地位，如果以一棵树来比喻的话，C语言可以说处于树根的位置，如图1-1所示。

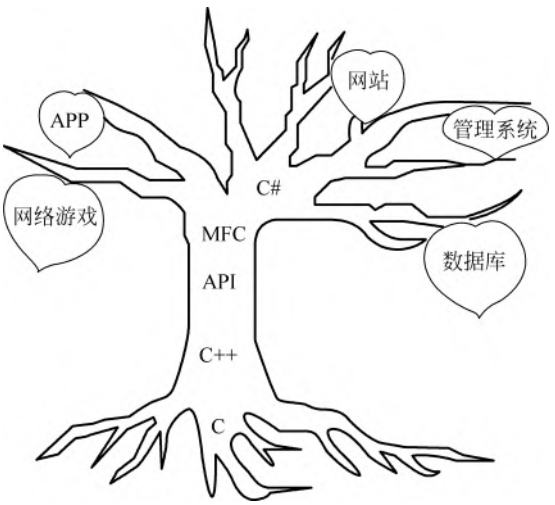


图1-1　语言之树

在正式学习C语言程序设计之前，将程序设计与盖房子做一类比，看看二者有哪些相似之处。

1．类比：盖房子与程序设计

如果要盖一栋房子，需要做哪些事情呢？

盖房子的基本过程可分解为以下步骤：

（1）在哪里盖？

（2）怎么盖？

（3）盖什么？

（4）遇到问题怎么办？

（5）盖房子过程中需要的辅助材料有哪些？

接着一一回答上面的问题：

（1）在哪里盖？

就是要圈地的问题。

（2）怎么盖？

盖房子需要一些基本的规则和步骤，比如先打地基，地基打成什么样子，然后砌墙，墙的厚度、高度为多少等。

(3) 盖什么?

盖什么功能的房子,是人居住的还是厂房?

(4) 遇到问题怎么办?

如果盖房子过程中出现了问题,是推倒重来,还是进行加固性的纠正?

(5) 盖房子过程中需要的辅助材料有哪些?

比如说需要打支子,但房子盖好后这些架子将会拆掉的。

2. 将 C 语言编程与盖房子类比

(1) 在哪里盖房子 VS 在哪里编写代码:开发环境(Dev C++、Visual C++ 6.0 等)。

(2) 怎么盖 VS 怎么编写程序:基本语法规则。

(3) 盖什么 VS 编写什么:编写什么功能的程序。

(4) 遇到问题怎么办:调试、纠错。

(5) 辅助:变量的定义、注释及语句的提示,等等。

3. 用任何语言编程都要考虑的问题

(1) 在哪里编写代码?

(2) 怎么编写?

(3) 编写什么?

(4) 遇到问题怎么办?

(5) 哪些地方需要辅助性的语句?

注:本书 C 语言开发环境统一使用 Dev-C++ 4.9.9.2。

小知识

Dev-C++是一个 C&C++开发工具,使用 Delphi/Kylix 开发,是一款自由软件,遵守 GPL 协议。它使用 MinGW/GCC/Cygwin 编译器,遵循 C/C++标准。

DEV C++已被全国青少年信息学奥林匹克联赛设为 C++语言指定编译器,同时也是 ACM-ICPC 竞赛官方指定 C 语言的编译器之一。

Dev-C++官网:http://sourceforge.net/projects/dev-C++/files/Binaries/

特别提示　Dev- C++安装说明见附录 A;Dev- C++使用说明见附录 B。

1.2　死记硬背

1.2.1　编程基本步骤

C 语言编程的基本步骤分为如下七个步骤:

(1) 打开 Dev-C++开发环境;

(2) 新建文件;

(3) 保存并命名文件;

(4) 编写代码;

（5）编译；

（6）调试；

（7）运行并测试程序。

小技巧

快速建立自己的C语言程序文件，即以c为后缀的文件（**.c）。

注意：快速建立自己的.c文件，此方法适应于计算机只安装一个能够打开C语言的开发环境，即只安装一个Dev-C++软件的计算机。

（1）在D盘先建立一个文件夹（例如：命名为“我的C语言代码”）；

（2）打开文件夹，在空白处右击；

（3）“新建”→“文本文件”；

（4）重命名“新建文本文档.txt”更改为“自己命名的文件名.c”（一定要将原来的扩展名“.txt”更改为“.c”）；

（5）直接双击刚才建立的.c文件，即可编写代码。

如果上述第(4)步没有显示“新建文本文档.txt”的后缀.txt，可以按以下方法操作：

以Windows 7为例：打开“我的电脑”，选择“工具”菜单（如果没有“工具”菜单，按Alt键），如图1-2所示。

图1-2 “工具”菜单

然后在“文件夹选项”（见图1-3）中的“查看”选项卡中去掉“隐藏书籍文件类型的扩展名”的勾选，如图1-4所示。

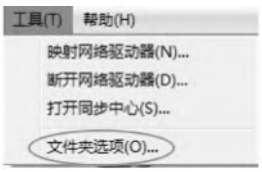


图1-3 文件夹选项

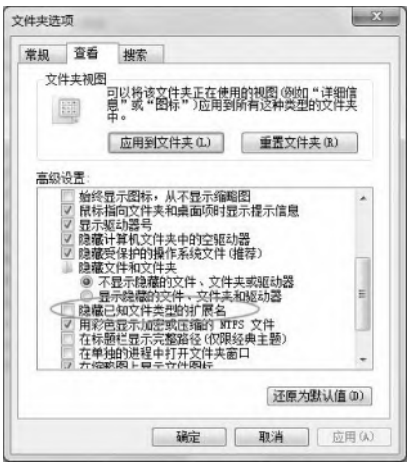


图1-4 “查看”选项卡

特别提示　新建文件，命名文件名时注意事项：不可默认，不可用数字开头，最好不要用汉字命名。

1.2.2　记死

【实例分析 1-1】　实现 A＋B，详细代码如程序 1-1 所示。将程序 1-1 在编程环境 Dev C++中编译并运行，看看结果是什么？

程序 1-1　A＋B 问题

```
#include<stdio.h>
int main()
{
printf("%d\n",1+1);        //输出语句
return 0;
}
```

看看程序 1-1 的运行结果如何？

下面对此程序作以简单解释：

```
#include<stdio.h>          //基本输入与输出头文件,如果程序中出现 printf()函数,必须有此语句
int main                   //主函数,任何程序必须包含主函数
{                          //括号表示函数体的范围,里面有两行命令
    printf("%d\n",1+1);//输出"2",%d 表示输出类型,\n 表示换行
return();                  //表示返回值
}
```

注释说明

单行注释：//注释内容

多行注释：/*注释内容*/

如果编译没有问题，运行后屏幕一闪而过，则需要在程序 1-2 的代码中加上阴影部分的两行代码，具体详细代码如程序 1-2 所示。

程序 1-2　A＋B 问题的改进

```
#include<stdio.h>
#include<stdlib.h>     //增加 system 的头文件
int main()
{
printf("%d\n",1+1);     //输出语句
system("pause");          //增加屏幕暂停语句
return 0;
}
```

注意：在某些C语言的编程环境中，不需要增加system("pause")；这条语句，比如Visual C++ 6.0。

常见错误举例

初次编写程序1-2常见的错误有：

(1) 括号没配对。头文件一定是尖括号< >，其他的括号均为圆括号()。

(2) 大小写没注意。C语言对大小写是敏感的。

(3) 所有的字符均是英文字符，尤其是逗号(英文逗号","与中文逗号"，"的不同)、分号(英文分号";"与中文分号"；"的不同)以及双引号(英文双引号""与中文引号"“”"的不同)。

(4) 没有空格。空格一定要有，例如int与main()之间，return与0之间。

上述程序有一个遗憾：计算的数据是事先确定的。为了计算1+2和2+3，即在运行过程中根据需要输入两个数据，我们不得不编写两个，甚至多个程序。可不可以让程序读取键盘输入，并根据输入内容计算结果呢？答案是可以。

【实例分析1-2】 A+B带有输入与输出的完整的程序代码，详细代码如程序1-3所示。

程序1-3 带有输入与输出的A+B问题

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",a+b);
    system("pause");
return 0;
}
```

常见错误举例

scanf()中的占位符和变量的数据类型没有一一对应。且每个变量前要在相应的位置加上&符号。

该程序复杂了许多。简单地说，第一条语句"int a,b"声明了两个整型(即整数类型)变量a和b，然后读取键盘输入，并放到a和b中。注意a和b前面的&符号，读者可将&去掉，看一下运行结果如何。

现在，程序已经读入了两个整数，可以在表达式中自由使用它们，就好比使用1 2，55 45这样的常数。

程序1-3为今后学习程序设计，尤其是程序1-3中阴影部分，为程序设计的基本框架。

此程序框架不一定要死记，但一定要学会灵活运用。

以后的C语言基本上以此框架为基础进行编写代码。

小技巧

从别的非编程环境中(网页或者其他文档中)粘贴代码到Dev-C++方法：

先粘贴到文本记事本上；

再粘贴到运行环境里面；

可以过滤掉一些特殊的字符；

可以说文本记事本是万能的文本过滤器。

记住：在C语言编程过程中，所有的事情，哪怕是一个变量，一定要自己创造(定义)，然后才能使用。

问题是：怎么知道哪些有，哪些没有？

这就是编程语言的语法规则，也是接下来学习的重点。

1.3　初学者方法

对于初学者，尤其是自学者来说，学习计算机语言最好的方法是“模仿程序，改造例程，举一反三”。

当然，对于没有学过任何计算机语言的初学者，最好还是先阅读教程，对于书本中提到的例程，一时不理解，可以先背会，在学习后面章节的过程中慢慢理解，以培养自学能力。

学习语言，必须注意每一个细节，书上的例子代码一定要亲自敲一遍，编译、执行并跟书上的一致才能算是学完了一个例子，如果不一致，要仔细找原因。

编写运行正确代码一段时间后，要会改造书本上的例子，学会举一反三，将改造加工的例程仔细地归类保存，并且在源代码中写上简短的注释，阐述例子的意图。所谓“好记性不如烂笔头”，就是这个道理。

例程之后就是练习了，建议初学者以OJ试题库(本书第六章讲解OJ系统编程的基本方法)作为练习的重点，找适合自己水平的习题，由浅入深，这是提高编程能力最重要的方法。

仔细读书、认真编写源代码、独立完成习题，外加更进一步的实验，最后将所有的代码保存，成为自己的经验和财富，绝对的辛苦能够换来事半功倍的效果。

最后，还有非常重要的一点——代码风格，从开始学习就必须强迫自己模仿最优秀的代码风格。

总之，要掌握一门语言，需要以下几点：

(1) 重视实验：哪怕不理解背后的道理，至少要清楚现象。

(2) 学会模仿：把学习的焦点集中在最有趣的地方。

(3) 如果直观的解决方案行得通，就不必追究其背后的机理。

（4）如果对一个东西不理解，就不要对它修改；如果非改不可，则应根据自己的直觉和猜测尝试各种改法，而不必过多考虑“为什么要这样”。

（5）当有一定的分析、解决问题的能力后，再寻找更多的资料进一步学习一些重要的概念和原理。

要想把事情做好，必须学得透彻，通过练习来提高，但没有必要操之过急。

第 2 章 数 据 类 型

本章首先通过实现 A＋B 程序的实例引出数据类型的概念，主要包括以下内容：

（1）A＋B 程序实例；

（2）A＋B 程序实例拓展实验。

然后介绍常用基本数据类型，主要包括以下内容：

（1）常量；

（2）变量；

（3）强制类型转换。

最后介绍变量的命名规则。

2.1 从 A＋B 说起

【实例分析 2-1】

题目描述：

给定两个整数，求它们之和。

输入：

输入两个数 a，b。

输出：

输出 a＋b 的值。

样例输入：

2 3

样例输出：

5

在编译环境 Dev C++下敲写此程序的代码，看看能否一次编写、编译和运行成功。详细代码如程序 2-1 所示。

程序 2-1　A＋B 问题

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",a+b);
    system("pause");
return 0;
}
```

【拓展训练 2-1】 修改程序 2-1，输出 1 － 2 的结果。

【拓展训练 2-2】 修改程序 2-1，输出 2 × 3 的结果。

【拓展训练 2-3】 修改程序 2-1，输出 4 ÷ 2 的结果。

【拓展训练 2-4】 修改程序 2-1，输出 6 ÷ 4 的结果。

直接把“1＋2”替换成“1－2”即可顺利解决拓展训练 2-1，但读者很快就会发现：无法在键盘上找到乘号和除号。解决方法是：用星号“*”代替乘号，用正斜线“/”代替除号。这样，4 个拓展训练都顺利完成了。

2.2　A＋B 继续

等一下，拓展训练 2-4 的输出结果居然是 1，而不是正确答案 1.5。这是怎么回事？计算机出问题了吗？

计算机没有出问题，问题出在程序上：这段程序的实际含义并非和我们所想的一致。

在 C 语言中，6/4 的确切含义是 6 除以 4 所得商值的整数部分。同样地，(－6)/4 的值是－1。如果非要得到 6÷4＝1.5 的结果怎么办？

【实例分析 2-2】 实现 A/B，详细代码如程序 2-2 所示。

程序 2-2　计算并输出 6/4 的值，保留小数点后 1 位

```
#include<stdio.h>
int main()
{
  printf("%.1lf\n",6.0/4.0);
  return 0;
}
```

注意：百分号后面是个小数点，然后是数字 1，再后是小写字母 l，最后是小写字母 f，千万不能打错，包括大小写——在 C 语言中，大写和小写字母代表的含义是不同的。

【拓展训练 2-5】 把%.1lf 中的数字 1 改成 2，结果如何？能猜想出“1”的确切意思吗？如果把小数点和 1 都删除，%lf 的含义是什么？

为什么这样？

原因并不重要，重要的是根据规范做事情。

特别提示　整数值用%d 输出，实数用%lf 输出。

课后自学　查找 C 语言中 printf()及 sacnf()的使用方法；查找 int 及 float 类型的详细用法。

这里的“整数值”指的是 1+2、6/4 这样“整数之间的运算”。只要运算符的两边都是整数，运算结果也会是整数。正因为如此，6/4 的值才是 1，而不是 1.5。

6.0 和 4.0 被看作是“实数”，或者更专业一点，叫“浮点数”。浮点数之间的运算结果是浮点数，因此 6.0/4.0=1.6。注意，这里的运算符“/”其实是“多面手”，既可以做整数除法，也可以做浮点数除法。

特别提示　整数/整数=整数，浮点数/浮点数=浮点数。

这条规则同样适用于加法、减法和乘法，不过没有除法这么容易出错——毕竟整数乘以整数的结果本来就是整数。

【实例分析 2-3】 算术表达式可以和数学表达式一样复杂，详细代码如程序 2-3 所示。

程序 2-3　复杂的表达式计算

```
#include<stdio.h>
#include<math.h>
int main()
{
  printf("%.8lf\n",1+2*sqrt(3)/(5-0.1));
  return 0;
}
```

编译成功后，运行程序 2-3，结果如图 2-1 所示。

```
1.70695951
请按任意键继续. . .
```

图 2-1　程序 2-3 运行结果

读者不难把它翻译成数学表达式：$1+\frac{2\sqrt{3}}{5-0.1}$。尽管如此，读者可能还是有一些疑惑：

（1）5−0.1 的值是什么？

（2）“整数−浮点数”是整数还是浮点数？

（3）#include<math.h>是做什么用的？

第 1 个问题相信读者能够“猜到”结果：整数−浮点数=浮点数。其实这个说法并不准

确。确切的说法是：整数先“变”成浮点数，然后浮点数－浮点数＝浮点数。

第 2 个问题的答案是：因为程序 2-3 中用到了数学函数 sqrt()。数学函数 sqrt(x)的作用是计算 x 的算术平方根(若不信，可输出 sqrt(9.0)的值试试)。一般来说，只要在程序中用到了数学函数，就需要在程序最开始的地方包含头文件 #include ＜math.h＞，并在编译时连接数学库。

2.3 基本数据类型

C 语言提供有丰富的数据类型，包含的数据类型如图 2-2 所示。

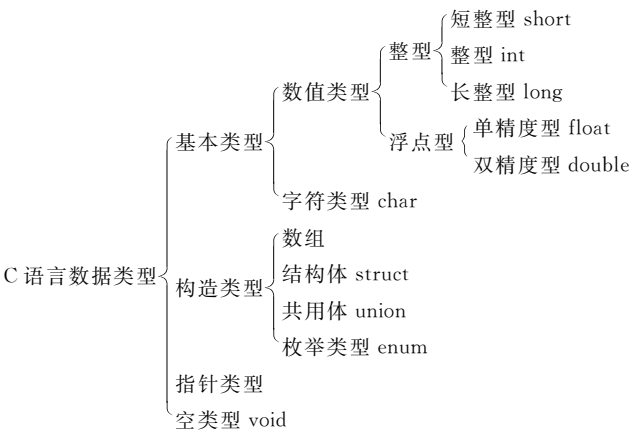


图 2-2 C 语言数据类型

short、int、long、char、float、double 这六个关键字代表 C 语言里的六种基本数据类型。C 语言基本数据类型及常见的表示方法如表 2-1 所示。

表 2-1 常见的数据类型(数值型和字符型)

类 型	符 号	关 键 字	所占位数	数的表示范围
整型	有	(signed)int	16	－32768～32767
		(signed)short	16	－32768～32767
		(signed)long	32	－2147483648～2147483647
	无	unsigned int	16	0～65535
		unsigned short	16	0～65535
		unsigned long	32	0～4294967295
实型	有	float	32	3.4e-38～3.4e38
	有	double	64	1.7e-308～1.7e308
字符型	有	char	8	－128～127
	无	unsigned char	8	0～255

说明：无符号型和符号型，分别用关键字 unsigned 和 signed 进行定义。signed 默认可去掉。

例如：整数 13 在内存中实际存放的情况，如图 2-3 所示。

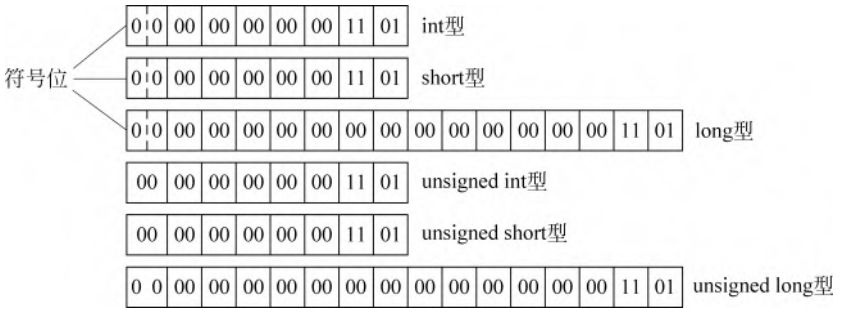


图 2-3　C 语言数据类型

2.3.1　数据类型与“模子”

short、int、long、char、float、double 这六个关键字代表 C 语言里的六种基本数据类型。

怎么去理解它们呢？

举个例子：见过蜂窝煤的模子吧？（没见过？蜂窝煤总见过吧）。拿着模子在煤堆里这么一“咔”，一块蜂窝煤就出来了。不同型号的蜂窝煤模子“咔”出来的蜂窝煤大小不一样，孔数也不一样。

现在联想一下，short、int、long、char、float、double 这六个关键字是不是很像不同类型的蜂窝煤模子啊？拿着它们在内存上“咔咔咔”，不同大小的内存就分配好了，当然别忘了给它们取个好听的名字。

在 32 位的系统上，short“咔”出来的内存大小是两个字节；

int“咔”出来的内存大小是 4 个字节；

long“咔”出来的内存大小是 4 个字节；

float“咔”出来的内存大小是 4 个字节；

double“咔”出来的内存大小是 8 个字节；

char“咔”出来的内存大小是 1 个字节。

注意：这里指一般情况，可能不同的平台还会有所不同，具体平台可以用 sizeof 关键字测试一下。

【实例分析 2-4】 测试不同类型的数据长度，详细代码如程序 2-4 所示。

程序 2-4　测定数据类型长度

```
#include<stdio.h>
#include<stdlib.h>
```

```
int main()
{
 int a,b;
 int i = 0;
 printf ("char: %d bytes.\n",sizeof(char));
 printf ("short: %d bytes.\n",sizeof(short));
 printf ("i: %d bytes\n",sizeof (i));                    /* 计算变量 i 的字节数 */
 printf ("long: %d bytes\n",sizeof(long));
 printf ("float: %d bytes\n",sizeof(float));
 printf ("double: %d bytes\n",sizeof(double));
 printf ("1.23456: %d bytes\n",sizeof(1.23456));         /* 计算常量的字节数 */
 printf ("double: %d bytes\n",sizeof(double));
 system("pause");
return 0;
}
```

编译成功后，运行程序 2-4，结果如图 2-4 所示。

图 2-4　程序 2-4 运行结果

很简单吧？“咔咔咔”很爽吧？但问题是“咔”出来这么多内存块，总不能给它取名字叫做 x1，x2，x3，x4，x5，…或者长江 1 号，长江 2 号……吧。它们长得这么像，过一阵子就会忘了到底哪个名字和哪个内存块匹配了。所以给它们取一个好的名字绝对重要。

下面来研究基本数据类型。

C 语言基本数据类型分为常量和变量两大类。

2.3.2　常量

常量是指在程序执行过程中其值保持不变的量。

常量不需要类型说明就可以直接使用，常量的类型是由常量本身隐含决定的。

常量通常用于定义具有如下特点的数据：在程序中保持不变，在程序内部频繁使用，需要用比较简单的方式替代某些值等情况。

C 语言的常量按其表现形式可以分为直接常量和符号常量两类。

1. 直接常量

C 语言规定的直接常量分为如下几类。

（1）整型常量：整数。

（2）实型常量：带有小数点的实数。

（3）字符型常量：包括可以全部在计算机上显示的符号。

（4）字符串常量：用一对双引号“ ”括起来的 0 个或多个字符组成的字符序列。

（5）符号常量：当某个常量引用起来比较复杂而又要经常使用时，可以将该常量定义为符号常量。

例如：

(1) 1,2,－1 这些值为整数，属于整型常量。

(2) 3.1,－4.0,3.1415926 的值为实数，属于实型常量。

(3) 'A','b'为字符常量。

(4) "i love hpu"是字符串常量

(5) ＃define PI 3.14159 定义 π 的符号常量

1) 整型常量

整型常量：就是一个整数，即整型常数。在计算机中一般占用两个字节。它可以用十进制、八进制、十六进制三种形式表示。

引申：字节的概念

字节(B)是计算机存储容量的最基本单位。二进制中一个"0"或一个"1"叫 1 位，一个字节由 8 个位组成，一个字节组成一个英文字母，两个字节组成一个汉字。

1024B＝1KB

1024KB＝1MB

1024MB＝1GB

1024GB＝1TB

进制也就是进位制，是规定的一种进位方法。对于任何一种进制——X 进制，表示某一位置上的数运算时是逢 X 进一位。十进制是逢十进一，十六进制是逢十六进一，二进制就是逢二进一，以此类推，X 进制就是逢 X 进位。

三种形式：

十进制整数：由数字 0～9 和正负号表示，如 123，－456，0。

八进制整数：由数字 0 开头，后跟数字 0～7 表示，如 0123，011。

十六进制整数：由 0X 或 0x 开头，后跟 0～9，a～f，A～F 表示，如 0x123，0Xff。

进制数的表示方式如表 2-2 所示。

表 2-2　进制数的表示举例

进　制　数	表 示 方 式	举　　例
二进制	以数字的最右端 B 结尾	110B
八进制整型	由数字 0 开头	034,065,057
十进制整型	如同数学中的数字	123,－78,90
十六进制整型	由 0X 或 0x 开头	0x23,0Xff,0xac

整型常量的类型，根据其值所在范围确定其数据类型，在整常量后加字母 l 或 L，是 long int 型常量。

例如：30000　　为 int 型

65536　　为 long int 型

注意：(1) 整型常数在不加说明时总是正值。如果需要的是负值，则负号'—'必须置于常数表达式的前面。

(2) 每个常数都要根据其值的需要给出一种数据类型。当整型常数应用于某一表达式时(或出现有负号)，常数类型自动进行相应的转换。十进制常数可等价于带符号的整型(或长整型)，这取决于所需的常数的尺寸。

(3) 八进制和十六进制常数可对应整型、无符号整型、长整型、无符号长整型，具体类型也取决于常数的大小，但必须注意取值范围。

2) 实型常量(实数或浮点数)

实型常量：就是通常带有小数点的实数，在计算机中占用 4 个字节。

实型常量又称浮点常量，是一个用十进制表示的符号实数，只有十进制一种表示方式(但可进一步分为十进制小数和十进制指数两种形式)。

实型常量有两种表示方法：

十进制数形式：由正负号、数字和小数点组成。

例如：0.123，.123，123.0，0.0，123

注意：必须有小数点。

指数形式：由尾数、字母 E 或 e、指数三部分组成。

符号实数的值包括整数部分、尾数部分和指数部分。

格式：[n1][.n2][E|e[+|-]n3]

例如：123.0E-1，1.23E3

其中 n1，n2，n3 分别是一位或多位十进制数字(0～9)，E 或 e 是数符号，小数点之前是整数部分，小数点之后是尾数部分，小数点在没有尾数时可以省略，指数部分用 E 或 e 开头，幂指数可以为负，当没有符号时视为正，指数的基数为 10。

这种形式类似数学中的指数形式。在数学中，可以用幂的形式来表示，如 2.3026 可以表示为 0.23026×10^1，2.3026×10^0，23.026×10^-1 等形式。在 C 语言中，则以"e"或"E"后跟一个整数来表示以"10"为底数的幂数。2.3026 可以表示为 0.23026E1、2.3026e0、23.026e-1。C 语言语法规定，字母 e 或 E 之前必须有数字，且 e 或 E 后面的指数必须为整数。例如 e3、5e3.6、.e、e 等都是非法的指数形式。但在字母 e 或 E 的前后以及数字之间不得插入空格。

注意：(1) 实型常量的类型默认为 double 型。

(2) 在实型常量后加字母 f 或 F，为 float 型。

3) 字符常量

C 语言中有两种类型的字符常量：普通字符和转义字符。

普通字符由所有那些未显式指定为无字符的打印和非打印字符组成。包括所有的大写和小写字母字符、数字、标点符号等。

例如：'A'——65，'a '——97

'0'——48，'\n'——10

'A'——'\101'——'\x41'——65

这里单引号仅起界定作用，并不表示字符本身。

单引号括起来的字符不能是单引号(')和反斜杠(\)，其特有的表示方法在转义字符中介绍。

在C语言中，字符是按其所对应的ASCII码值来存储的，一个字符占一个字节，详细对应关系如表2-3所示。

表2-3　字符与ASCII码对应关系

代码	字符	代码	字符	代码	字符	代码	字符	代码	字符
32	空格	51	3	70	F	89	Y	108	l
33	!	52	4	71	G	90	Z	109	m
34	"	53	5	72	H	91	[	110	n
35	#	54	6	73	I	92	\	111	o
36	$	55	7	74	J	93	]	112	p
37	%	56	8	75	K	94	^	113	q
38	&	57	9	76	L	95	_	114	r
39	'	58	:	77	M	96	`	115	s
40	(	59	;	78	N	97	a	116	t
41	)	60	<	79	O	98	b	117	u
42	*	61	=	80	P	99	c	118	v
43	+	62	>	81	Q	100	d	119	w
44	,	63	?	82	R	101	e	120	x
45	-	64	@	83	S	102	f	121	y
46	.	65	A	84	T	103	g	122	z
47	/	66	B	85	U	104	h	123	{
48	0	67	C	86	V	105	i	124	|
49	1	68	D	87	W	106	j	125	}
50	2	69	E	88	X	107	k	126	~

注意：(1) 字符数字('0'～'9')和数字(0～9)的含义和在计算机中的存储方式是截然不同的，前者是字符常量，存储的是相应的ASCII值；后者是整型常量，存储的是相应的二进制数。

(2) 由于C语言中字符常量是按短整数(short型)存储的，所以字符常量可以像整数一样在程序中参与相关的运算。

美国标准信息交换码128个字符，最为常用的是ISO(标准化组织)标准的字符集。在其字符集内，每个字符对应唯一的码值(次序值)，不同字符码值不同。例如，'0'表示数字字符0，其码值为48；'A'表示字母字符A，其码值为65等。ASCII字符集内，数字、大写字母、小写字母的大小关系为：

'0'<'9'<'A'<'Z'<'a'<'z'

转义字符是C语言中表示字符的一种特殊形式，是一种特殊的字符常量。

由于跟在'\'后的字符已不代表原来的字符含义，所以称其为转义字符。

例如：'\101'——'A'　　'\012'——'\n'

'\x61'——'a'　　'\60'——'0'

通常使用转义字符表示ASCII码字符集中不可打印的控制字符和特定功能的字符，转义字符用反斜杠'\'后面跟一个字符或一个八进制或一个十六进制数，如表2-4所示。

表2-4　转义字符

转义字符	意　义	ASCII码值(十进制)
\a	响铃(BEL)	007
\b	退格(BS)	008
\f	换页(FF)	012
\n	换行(LF)	010
\r	回车(CR)	013
\t	水平制表(HT)	009
\v	垂直制表(VT)	011
\\	反斜杠	092
\?	问号字符	063
\'	单引号字符	039
\"	双引号字符	034
\0	空字符(NULL)	000
\ddd	任意字符	三位八进制数
\xhh	任意字符	二位十六进制数

【拓展训练2-6】 将程序2-1中printf("%d\n",a+b);中的\n去掉，运行后看看有哪些不一样。

【实例分析2-5】 转义字符举例，详细代码如程序2-5所示。

程序2-5　转义字符举例-1

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  printf("\101 \x42 C\n");
```

```
    printf("I say:\"How are you?\"\n");
    printf("\\C Program\\\n");
    printf("Turbo \'C\'");
    system("pause");
  return 0;
}
```

编译成功后，运行程序 2-5，结果如图 2-5 所示，请分析一下，为何有此结果。

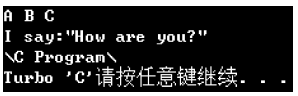


图 2-5　程序 2-5 运行结果

如果在字符常量中需要使用单引号、双引号和反斜杠，则必须使用转义字符来表示，即在这些字符前加上反斜杠‘\’。

【实例分析 2-6】 转义字符举例，详细代码如程序 2-6 所示。

程序 2-6　转义字符举例-2

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    printf("Y\b=\n");
    system("pause");
  return 0;
}
```

编译成功后，运行程序 2-6，结果如图 2-6 所示。请分析一下，为何有此结果。

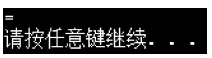


图 2-6　程序 2-6 运行结果

在 C 程序中使用转义字符“\ddd”(或者“\xhh”)可以表示任意字符。“\ddd”为反斜杠后面跟 3 位八进制数，该 3 位八进制数的值即为对应的八进制 ASCII 码值。“\xhh”为反斜杠后面跟两位十六进制数，该两位十六进制数的值即为对应的十六进制 ASCII 码值。

注意：(1) 转义字符中只能使用小写字母，每个转义字符只能看作一个字符。

(2) 使用不可打印字符时，通常用转义字符表示。

(3) “\v”垂直制表和“\f”换页符对屏幕显示没有任何影响，但会影响打印机执行的响应操作。

4）字符串常量

字符串常量是指用一对双引号括起来的一串字符。双引号仅起界定作用，双引号括起来的字符串中不能含双引号(")和反斜杠(\)，它们特有的表示法在转义字符中已做介绍。

例如"China"，"C program"，"YES&NO"，"333222-666999"，"A"等均为字符串常量。

字符串常量在内存中存储时，系统自动在字符串的末尾加一个"串结束标志"(即 ASCII 码值为 0 的字符 NULL)，常用'\0'表示。因此在程序中，含有 n 个字符的字符串常量，在内存中占有 n+1 个字节的存储空间。字符串的长度应为实际长度。

例如：

字符串"hello"在内存中

h	e	l	l	o	\0

空串 ""

\0

空格串" "

	\0

"a"是由一个字符 a 构成的字符串。

"Happy new Year"是由多个字符序列构成的字符串。

"abc\n"是由多个字符构成的字符串。

注意：(1) '\0'和'0'不同，'\0'是编码为 0 的字符，而'0'则是数字 0 所对应的字符。

(2) 在字符串中有转义字符。

例如："ab\072cdef"长度为 7。

"\\\n33abcd"长度为 8。

字符常量与字符串常量的区别：

在内存中，字符常量的存储只占用一个字节，而字符串常量存储时，C 语言编译系统将自动在字符串的尾部加上一个特殊的字符'\0'，作为字符串结束的标志。系统依据此标志进行判断该字符串是否结束。

(1) 'a '与"a"是不同的：'a '表示的是字符常量，在内存中占一个字节；而"a"表示的是字符串常量，在内存中占两个字节。

'a' | a |

"a" | a | \0 |

```
char   ch;
       ch="A";     ×
       ch='A';     √
```

（2）一个字符串常量的存储长度要比它实际的字符串长度多一个字节（字符）。所以字符串常量与字符常量的区别是：

① 书写格式不同：‘’与“”。

② 表现形式不同：一个与多个。

③ 存储方式不同：长度不同。

（3）""和‘’是不同的。前者是一个空字符串常量""，实际包含了一个空字符'\0'，在内存中占用一个字节的存储空间。而后者则是非法的用法。

2．符号常量

符号常量是用一个标识符来代表的常量。

格式：

```
#define <符号常量名> <常量>
```

功能：用被定义了的符号常量名来代替常量。

例如：

```
#define  AGE  35
#define  M  1.9734067e9
```

符号常量必须在使用前先定义。

优点：简化书写格式、减少出错率；可以和常量一样运算，一旦要求有所变化，只需更改宏定义。

注意：（1）符号常量不同于变量，在其作用域内不能被改变和重新赋值。

（2）习惯上，符号常量名用大写英文标识符，而变量名用小写英文标识符，以示区别。

（3）定义符号常量可以提高程序的可读性，便于程序的调试和修改。因此在定义符号常量名时，应尽量使其表达它所代表的常量的含义。

（4）对符号常量定义后，当一般常量使用。

【实例分析 2-7】 符号常量实例分析，详细代码如程序 2-7 所示。

程序 2-7　求一个圆柱体体积，用符号常量代替 π

```
#include <stdio.h>
#include <stdlib.h>
#define PI 3.14159
int main()
{
```

```
    float r,h,v;
    scanf(" %f, %f",&r,&h);
    v = PI * r * r * h;
    printf("Volume = %f\n",v);
    system("pause");
return 0;
}
```

编译成功后，运行程序 2-7，结果如图 2-7 所示。

图 2-7　程序 2-7 运行结果

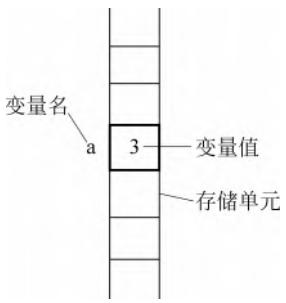


图 2-8　C 语言数据类型

2.3.3　变量

1）变量的定义

变量是指那些在程序执行过程中其值可以改变的量。

变量在内存中要占用一定的存储空间来存放变量的值，它们有名字和数据类型。编写程序时，常常需要将数据存储在内存中，方便后面使用或者修改这个数据的值。所有变量都具有数据类型，以决定能够存储哪种数据。变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中。所有的变量都要有一个名字，变量名必须是合法的标识符。

2）变量的定义格式

在 C 语言中，变量在使用前必须先定义，以确定它的数据类型（见图 2-8）和存储类型，进而确定该变量所能进行的运算。

格式：

```
[<存储类型>]<数据类型><变量名 1>[[ = <初值 1>],…,<变量名 n>[[ = <初值 n>]];
```

功能：定义变量名所指定的变量，并可以赋初值。

变量定义具有两个目的：

（1）定义变量名。

（2）定义变量的数据类型。

注意：(1) 变量名标明数据在内存中的地址，编译时系统为每个变量名分配一个内存地址。在程序中，对变量的存取实际上是通过变量名找到相应的内存地址，然后从其存储单元中读取数据。

(2) 声明变量类型的目的是告诉系统，变量需要占用的存储单元数目(不同类型的数据在内存中所占的存储单元数目是不同的)，以便系统为变量分配相应的存储单元。

(3) 一次可以定义多个同一类型的变量。

3) 变量的赋值

某些变量在定义的同时必须进行初始化，也就是给它们赋初值，以确保它们在程序运行时有确定的值。

变量的初始化通常在变量定义时通过赋值语句来实现。

赋值语句是由赋值表达式再加上分号构成的表达式语句。

一般形式：

```
变量 = 表达式;
```

赋值语句的功能和特点都与赋值表达式相同。它是程序中使用最多的语句之一。

在赋值语句的使用中需要注意以下几点：

(1) 由于在赋值符“=”右边的表达式也可以又是一个赋值表达式。

因此，下述形式：

```
变量 = (变量 = 表达式);
```

是成立的，从而形成嵌套的情形。

其展开之后的一般形式为：

```
变量 = 变量 = … = 表达式;
```

例如：

```
a = b = c = d = e = 5;
```

按照赋值运算符的右接合性，实际上等效于

```
e = 5;
d = e;
c = d;
b = c;
a = b;
```

(2) 注意在变量说明中给变量赋初值和赋值语句的区别。

给变量赋初值是变量说明的一部分，赋初值后的变量与其后的其他同类变量之间仍必须用逗号间隔，而赋值语句则必须用分号结尾。例如：

```
int a = 5,b,c;
```

(3) 在变量说明中,不允许连续给多个变量赋初值。如下述说明是错误的:

```
int a = b = c = 5;
```

必须写为

```
int a = 5,b = 5,c = 5;
```

而赋值语句允许连续赋值。

(4) 注意赋值表达式和赋值语句的区别。

赋值表达式是一种表达式,它可以出现在任何允许表达式出现的地方,而赋值语句则不能。

下述语句是合法的:

```
if((x = y + 5)> 0) z = x;
```

语句的功能是,若表达式 x=y+5 大于 0 则 z=x。

下述语句是非法的:

```
if((x = y + 5;)> 0) z = x;
```

因为 x=y+5;是赋值语句,不能出现在表达式中。

引申:变量

(1) 变量的变在哪里,量又在哪里?

变量名(用标识符表示)、变量在内存中占据的存储单元、变量值三者关系。

变量名在程序运行过程中不会改变,变量的值可以改变。

变量名遵守标识符准则。

(2) C 语言中变量:“先定义,后使用”。

C 语言要求对所有用到的变量做强制定义。

只有声明过的变量才可以在程序中使用,这使得变量名的拼写错误容易发现。

声明的变量属于确定的类型,编译系统可方便地检查变量所进行运算的合法性。

在编译时根据变量类型可以为变量确定存储空间,“先定义后使用”使程序效率高。

变量的数据类型分为数值型变量和字符型变量两大类。

1. 数值型变量

数值型变量又分为整型变量和实型变量两类。

整型变量为 int,系统默认类型 int。

实型变量分为单精度(float 型)、双精度(double 型)和长双精度型(long double 型)、float 单精度浮点型是 6 位有效数字。一般 float 用 32 位表示,double 用 64 位表示。

另外,还有无符号整型和无值型,它们分别用关键字 unsigned 和 void 进行定义。其详细的说明如表 2-5 所示。

表 2-5　实型变量所占内存

数据类型	类型符	占内存(字节)	占内存(位)	数值范围	有效数字
单精度	float	4	32	$10^{-37}\sim10^{38}$	7
双精度	double	8	64	$10^{-307}\sim10^{308}$	16
长双精度	long double	16	128	$10^{-4931}\sim10^{4932}$	19

C 编译系统通常把所有实型常数都默认为 double 型。绝对值小于 1 的浮点数,其小数点前面的零可以省略。Turbo C 默认格式输出浮点数,最多只保留小数点后 6 位。

注意:(1) 数据类型限制在本区间使用。

(2) 基本整型常量和短整型范围为－32768～32767。

(3) 长整型常量范围为－2147483648～2147483647,整数后面加上字母 l(或 L)。

(4) 无符号整型常量表示的数据全部是正数,没有符号位。

int,short int,long int,unsigned int unsigned short int,unsigned long int

【实例分析 2-8】 超出基本整型数据的最大允许值,详细代码如程序 2-8 所示。

程序 2-8　超出基本整型数据的最大允许值

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int a,b;
  a = 2147483647; //在本机中,所演示程序,整型 int 占 4 个字节,所能表示范围为 2147483647～2147483648
  b = 2147483648;
  system("pause");
  printf("%d,%d",a,b);
return 0;
}
```

编译成功后,运行程序 2-8,结果如图 2-9 所示。

图 2-9　程序 2-8 运行结果

【实例分析 2-9】 输出双精度数,详细代码如程序 2-9 所示。

程序 2-9　输出双精度数

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  double  x,y;
  x = 1111111111111.111111111;
  y = 2222222222222.222222222;
```

```
  printf("%f\n",x+y);
  system("pause");
return 0;
}
```

编译成功后，运行程序 2-9，结果如图 2-10 所示。

【实例分析 2-10】 整型变量与实型变量的使用举例——整型，详细代码如程序 2-10 所示。

程序 2-10 A/B——整型

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int a,b;
 scanf("%d%d",&a,&b);
 printf("%d\n",a/b);
 system("pause");
return 0;
}
```

编译成功后，运行程序 2-10，结果如图 2-11 所示。

图 2-10 程序 2-9 运行结果

图 2-11 程序 2-10 运行结果

【实例分析 2-11】 整型变量与实型变量的使用举例——浮点型，详细代码如程序 2-11 所示。

程序 2-11 A/B——浮点型

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 float a,b;
 scanf("%f%f",&a,&b);
 printf("%f\n",a/b);
 system("pause");
return 0;
}
```

编译成功后，运行程序 2-11，结果如图 2-12 所示。

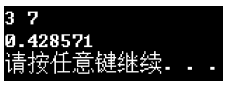


图 2-12　程序 2-11 运行结果

从定义阶段开始，程序设计一定要注意前后一致问题。

仔细观察程序 2-10 与 2-11，是否有不一致的地方；如果不一致，又能得出什么结果？

【拓展训练 2-7】 将程序 A/B——浮点型变量定义更改为

```
int a,b;
```

其他不变。

【拓展训练 2-8】 将程序 A/B——浮点型

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

更改为

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

其他不变。

【拓展训练 2-9】 将程序 A/B——浮点型

```
printf("%f\n",a/b);
```

更改为

```
printf("%d\n",a/b);
```

其他不变。

引申：实型数据在内存中的存放形式

浮点型数据是按照指数形式存储的。

系统把一个实型数据分成小数部分和指数部分分别存放。指数部分采用规范化的指数形式。C 语言数据类型如图 2-13 所示。

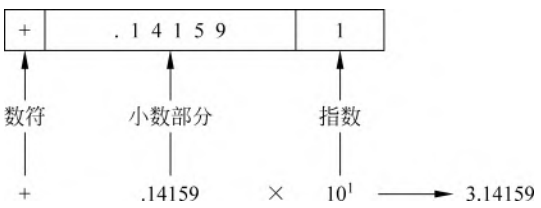


图 2-13　C 语言数据类型

【实例分析 2-12】 实型数据的舍入误差，详细代码如程序 2-12 所示。

程序 2-12 实型数据的舍入误差

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 float a,b;
 a=123456.789e5;
 b=a+20;
 printf("%f\n",b);
 system("pause");
return 0;
}
```

编译成功后，运行程序 2-12，结果如图 2-14 所示。

图 2-14 程序 2-12 运行结果

注意：一个实型变量只能保证有效数字是 7 位，后面的数字是无意义的，并不能准确地表示该数。应当避免将一个很大的数和一个很小的数直接相加或相减，否则就会“丢失”小的数。

2. 字符型变量

字符型变量是用来存放字符数据，同时只能存放一个字符。所有编译系统都规定以一个字节来存放一个字符，或者说，一个字符变量在内存中占一个字节。

字符型变量的值实质上是一个 8 位的整数值，因此取值范围一般是－128～127，char 型变量也可以加修饰符 unsigned，则 unsigned char 型变量的取值范围是 0～255（有些机器把 char 型当做 unsighed char 型对待，取值范围总是 0～255）。

特别提示 C 语言中有字符常量、字符串常量、字符变量，却没有字符串变量，字符串将用字符数组来存储。

【实例分析 2-13】 给字符变量赋以整数（字符型、整型数据通用），详细代码如程序 2-13 所示。

程序 2-13 给字符变量赋以整数（字符型、整型数据通用）

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
```

```
int main(){                          /* 字符a的各种表达方式 */
    char c1 = 'a';
    char c2 = '\x61';                /*  '\x..', '\...' */
    char c3 = '\141';
    char c4 = 97;
    char c5 = 0x61;                  /*  0x..,0...  */
    char c6 = 0141;
    printf("\nc1 = %c,c2 = %c,c3 = %c,c4 = %c,c5 = %c,c6 = %c\n",c1,c2,c3,c4,c5,c6);
    printf("c1 = %d,c2 = %d,c3 = %d,c4 = %d,c5 = %d,c6 = %d\n",c1,c2,c3,c4,c5,c6);
    getch();
    system("pause");
return 0;
}
```

编译成功后，运行程序 2-13，结果如图 2-15 所示。

```
c1=a,c2=a,c3=a,c4=a,c5=a,c6=a
c1=97,c2=97,c3=97,c4=97,c5=97,c6=97
```

图 2-15　程序 2-13 运行结果

运行结果及分析：

(1) 整型数→机内表示(两个字节)→取低 8 位赋值给字符变量。

(2) 函数 getch()的头文件为#include<conio.h>。

2.3.4　强制类型转换

在双向四车道的高速路上行车，前方正在修路，变成两车道，那么驾驶人必须并成两车道进行安全驾驶。

强制类型转换也有此道理。

1. 各类数值型数据间的混合运算

混合运算：整型(包括 int，short，long)、浮点型(包括 float，double)之间可进行混合运算。在进行运算时，不同类型的数据要先转换成同一类型，然后进行运算，其转换过程如图 2-16 所示。

图 2-16　C 语言数据类型

【实例分析 2-14】

```
int x = 2;float y = 1.6;char c = 'A';
```

c＋x＊y＝？的运算过程如图 2-17 所示。

2．数据类型间的强制转换

如果一个运算符两边的运算数类型不同，首先要将其转换为相同的类型，即较低类型转换为较高类型，然后再参加运算，转换规则如图 2-18 所示。

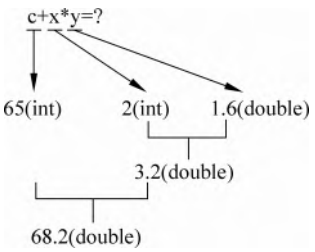


图 2-17　强制类型转换顺序

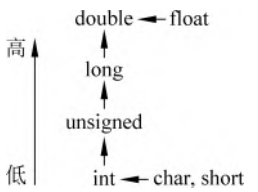


图 2-18　强制转换优先级

图 2-18 中**横向箭头**表示必须的转换，例如两个 float 型数参加运算，虽然它们类型相同，但仍须先转成 double 型再进行运算，结果为 double 型。**纵向箭头**表示当运算符两边的运算数为不同类型时的转换，例如一个 long 型数据与一个 int 型数据一起运算，需要先将 int 型数据转换为 long 型，然后两者再进行运算，结果为 long 型。所有这些转换都是由系统自动进行的，使用时只需从中了解结果的类型。这些转换是自动的，但是 C 语言也提供了以显式的形式强制转换类型的机制。

强制转换是通过类型转换运算来实现的。

一般形式：

（要转换成的数据类型）（被转换的表达式）

强制类型转换形式中的表达式一定要用括号括起来，否则强制转换仅对强制转换运算符的变量进行类型转换。

功能：把表达式的结果强制转换为类型说明符所表示的类型。

例如：

```
(int)a;                    //将 a 的结果强制转换为整型量
(int)(x + y);              //将 x + y 的结果强制转换为整型量
(float)(5 % 3);            //将 5 % 3 的值转换成 float 型
```

注意：强制类型转换时，得到所需类型的值，原来变量的类型和值都不变。

【实例分析 2-15】

```
int a = 2,b = 5; float x = 4.4;
```

(float)b/a 和(int)x/a 的运算结果如图 2-19 所示。

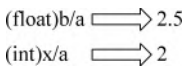


图 2-19　强制转换运算结果

注意：(1) 类型说明和表达式都需要加括号(单个变量可以不加括号)。

(2) 隐式转换和强制转换都是临时转换，不改变数据本身的类型和值。

【实例分析 2-16】 强制类型转换举例，详细代码如程序 2-14 所示。

程序 2-14　强制类型转换举例

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
     float f=5.75;
    printf("(int)f=%d\n",(int)f);          /* 将 f 的结果强制转换为整型,输出 */
    printf("f=%f\n",f);                    /* 输出 f 的值 */
    system("pause");
return 0;
}
```

编译成功后，运行程序 2-14，结果如图 2-20 所示。

图 2-20　程序 2-14 运行结果

当较低类型的数据转换为较高类型时，一般只是形式上的改变，而不影响数据的实际结果；而较高类型的数据转换为较低类型时，有些数据可能丢失。

3. 赋值中的类型转换

当赋值运算符两边的运算对象类型不同时，将会发生类型转换，转换的规则是：把赋值运算符右侧表达式的类型转换为左侧变量的类型。具体的转换如下：

1) 浮点型与整型

将浮点数(单双精度)转换为整数时，将舍弃浮点数的小数部分，只保留整数部分。

将整型值赋给浮点型变量，数值不变，只将形式改为浮点形式，即小数点后带若干个 0。注意：赋值时的类型转换实际上是强制的。

2）单、双精度浮点型

由于C语言中的浮点值总是用双精度表示的，所以float型数据只是在尾部加0延长为double型数据参加运算，然后直接赋值。double型数据转换为float型时，通过截尾数来实现，截断前要进行四舍五入操作。

3）char型与int型

int型数值赋给char型变量时，只保留其最低8位，高位部分舍弃。

char型数值赋给int型变量时，一些编译程序不管其值大小都作正数处理，而另一些编译程序在转换时，若char型数据值大于127，就作为负数处理。对于使用者来讲，如果原来char型数据取正值，转换后仍为正值；如果原来char型值可正可负，转换后也仍然保持原值，只是数据的内部表示形式有所不同。

4）int型与long型

long型数据赋给int型变量（这里假定int型占两个字节）时，将低16位值送给int型变量，而将高16位截断舍弃。

将int型数据赋给long型变量时，其外部值保持不变，而内部形式有所改变。

5）无符号整数

将一个unsigned型数据赋给一个占据同样长度存储单元的整型变量时（例如unsigned→int、unsigned long→long，unsigned short→short），原值照赋，内部的存储方式不变，但外部值却可能改变。

将一个非unsigned整型数据赋给长度相同的unsigned型变量时，内部存储形式不变，但外部表示时总是无符号的。

【实例分析2-17】 赋值运算符举例，详细代码如程序2-15所示。

程序2-15 赋值运算符举例

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    unsigned a,b;
    int i,j;
    a=65535;
    i=-1;
    j=a;
    b=i;
    printf("(unsigned)→(int)%d\n",a,j);
    printf("(int)→(unsigned)%u\n",i,b);
    system("pause");
return 0;
}
```

编译成功后，运行程序 2-15(64 位计算机)，结果如图 2-21 所示。

图 2-21　程序 2-15 运行结果

特别提示　不同的计算机由于机器字长不同，程序运行的结果可能不一致。

引申：强制类型转换

C 语言赋值时的类型转换形式可能会使人感到不精密和不严格，因为不管表达式的值怎样，系统都自动将其转为赋值运算符左部变量的类型。而转变后数据可能有所不同，在不加注意时就可能带来错误。但不应忘记的是：C 语言最初是为了替代汇编语言而设计的，所以类型变换比较随意。当然，用强制类型转换是一个好习惯，这样，至少从程序上可以看出想做什么转换。

2.4　变量的命名规则

C 语言允许将值存放在变量中，C 程序中出现的每个变量，都是由用户在程序设计时按照标识符的规则取名并定义的。每个变量都由一个变量名来标识。

1. 标识符的命名规则

(1) C 语言规定变量只能由字母、数字和下划线组成，且第一个字符必须是字母或下划线。

(2) 大、小写字母被认为是不同的变量名。Sun、sun、SUN 是三个不同的变量。为了避免混淆，应该为变量取不同的名字而不是用大小写区分。

(3) 变量名的长度无统一的规定，但在取名时长度应尽量在 31 位有效字符之内。

(4) 命名应当直观且可以拼读，可望文知意，便于记忆和阅读。

标识符最好采用英文单词或其组合，不允许使用拼音。程序中的英文单词一般不要太复杂，用词应当准确。

命名的长度应当符合“min-length && max-information”即最小长度和最大效益原则。

C 语言是一种简洁的语言，命名也应该是简洁的。例如变量名 MaxVal 就比 MaxValueUntilOverflow 好用。标识符的长度一般不要过长，较长的单词可通过去掉“元音”形成缩写。

另外，英文词尽量不缩写，特别是非常用的专业名词，如果有缩写，在同一系统中对同一单词必须使用相同的表示法，并且注明其意思。

当标识符由多个词组成时，每个词的第一个字母大写，其余全部小写。例如：

```
int CurrentVal;
```

这样的名字看起来清晰，远比一长串字符好得多。

尽量避免名字中出现数字编号，如 Value1、Value2 等，除非逻辑上的确需要编号。

初学者总是喜欢用带编号的变量名或函数名，这样看上去很简单方便，但其实是一颗颗定时炸弹。这个习惯初学者一定要改正。

程序中不得出现仅靠大小写区分的相似的标识符。例如：

```
int x,X;                                    //变量 x 与 X 容易混淆
 void foo(int x);                           //函数 foo 与 FOO 容易混淆
 void FOO(float x);
```

这里还有一个要特别注意的是 1(数字 1)和 l(小写字母 l)之间，0(数字 0)和 o(小写字母 o)之间的区别。这两对真是很难区分。

考虑到习惯性问题，局部变量中可采用通用的命名方式，仅限于 n、i、j 等作为循环变量使用。

一定不要写出如下的代码：

```
int      p;
char     i;
int      c;
char      * a;
```

一般来说，习惯上用 n、m、i、j、k 等表示 int(integer，整数)类型的变量；c、ch(char，字符)等表示字符类型变量；a(array，数组)等表示数组。当然，这仅仅是一般习惯。除了 i、j、k 等可以用来表示循环变量，别的字符变量名尽量不要使用。

(5) 定义变量的同时千万别忘了初始化。定义变量时编译器并不一定清空了这块内存，它的值可能是无效的数据。

2. 变量标符分类

(1) 关键字。例如 int、float、double、auto 等，详细的关键字见附录 E。

(2) 预定义标识符。包括函数名和预处理命令名。例如 printf、scanf、main、sin、include、define 等。

(3) 用户标识符。不能使用关键字，可以使用预定义标识符。例如：

```
int printf = 0;
int weight = 68;
```

【实例分析 2-18】 输入三角形的三边长，求三角形面积。

提示：已知三角形的三边长 a、b、c，则该三角形的面积公式为

$$area=(s(s-a)(s-b)(s-c))^{1/2}$$

其中 s = (a+b+c)/2。

提示 一定要考虑数据类型，详细代码如程序 2-16 所示。

程序 2-16　求三角形面积

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main(){
    float a,b,c,s,area;
    scanf("%f,%f,%f",&a,&b,&c);
    s=1.0/2*(a+b+c);
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("a=%7.2f,b=%7.2f,c=%7.2f,s=%7.2f\n",a,b,c,s);
    printf("area=%7.2f\n",area);
    system("pause");
return 0;
}
```

编译成功后，运行程序 2-16，结果如图 2-22 所示。

图 2-22　程序 2-16 运行结果

2.5　拓展训练

【拓展训练 2-10】 编写一个求正三角形的外接圆面积的程序，注意数据类型的定义。

题目描述：

给出正三角形的边长，π=3.1415926，求正三角形的外接圆面积。

输入：

只有一组测试数据，第一行输入一个整数 $n(1<n<1000)$ 表示接下来要输入 n 个边长 $m(1.0\leqslant m<1000.0)$。

输出：

输出每个正三角形的外接圆面积，保留两位小数，每个面积单独占一行。

样例输入：

5

1

13

22

62

155

样例输出：

1.05

176.98

506.84

4025.43

25158.92

第 3 章 数据的控制台输入与输出

本章首先介绍基本输入与输出。主要包括以下内容：

(1) printf()函数；

(2) scanf()函数；

(3) getchar()函数；

(4) putchar()函数。

然后介绍标准 C 语言程序框架。主要包括以下内容：

(1) 头文件；

(2) 函数；

(3) getchar()函数。

输入与输出的对象是数据，而数据是以介质为载体的，因此进行输入与输出操作就要与各种外部设备发生联系，要指定从哪个设备(文件)读入数据，将数据输出到哪个设备(文件)上去。本章只讨论从终端(键盘)输入和输出到终端(显示器)的输入与输出函数。通常也把这些函数称为控制台输入与输出函数。

主要介绍应用最广泛的 scanf()函数、printf()函数、getchar()函数和 putchar()函数等。其中 scanf()和 printf()用于格式化输入与输出。

3.1 printf()函数和 scanf()函数

3.1.1 printf()函数

printf()函数的功能是按指定的输出格式把相应的参数值在标准输出设备(通常是终端)上显示出来。

printf()函数是格式化输出函数，一般用于向标准输出设备按规定格式输出信息。在编写程序时经常会用到此函数。printf()函数的调用格式为：

```
printf("<格式化字符串>",<参量表>);
```

其中，格式化字符串包括两部分内容：一部分是正常字符，这些字符将按原样输出；另一部

分是格式化规定字符，以“%”开始，后跟一个或几个规定字符，用来确定输出内容格式。

参量表是需要输出的一系列参数，其个数必须与格式化字符串所说明的输出参数个数一样多，各参数之间用“,”分开，且顺序一一对应，否则将会出现错误。

说明：

(1) 可以在“%”和字母之间插进数字表示最大场宽。

例如：%3d 表示输出 3 位整型数，不足 3 位右对齐。

%9.2f 表示输出场宽为 9 的浮点数，其中小数位为 2，整数位为 6，小数点占一位，不足 9 位右对齐。

%8s 表示输出 8 个字符的字符串，不足 8 个字符右对齐。

如果字符串的长度或整型数位数超过说明的场宽，将按其实际长度输出。但对浮点数，若整数部分位数超过了说明的整数位宽度，将按实际整数位输出；若小数部分位数超过了说明的小数位宽度，则按说明的宽度以四舍五入输出。

另外，若想在输出值前加一些 0，就应在宽度前加个 0。

例如：%04d 表示在输出一个小于 4 位的数值时，将在前面补 0 使其总宽度为 4 位。

如果用浮点数表示字符或整型量的输出格式，小数点后的数字代表最大宽度，小数点前的数字代表最小宽度。

例如：%6.9s 表示显示一个长度不小于 6 且不大于 9 的字符串。若大于 9，则第 9 个字符以后的内容将被删除。

(2) 可以在“%”和字母之间加小写字母 l，表示输出的是长型数。

例如：%ld 表示输出 long 整数。

%lf 表示输出 double 浮点数。

(3) 可以控制输出左对齐或右对齐，即在“%”和字母之间加入一个“-”号说明输出为左对齐，否则为右对齐。

例如：%-7d　表示输出 7 位整数左对齐。

%-10s　表示输出 10 个字符左对齐。

【实例分析 3-1】 函数使用举例，详细代码如程序 3-1 所示。

程序 3-1　printf()函数使用举例

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(){
    char c;
    int a = 1234;
    float f = 3.141592653589;
    double x = 0.12345678987654321;
    c = '\x41';
    printf("a = %d\n",a);        /* 结果输出十进制整数 a = 1234 */
```

```
    printf("a = %6d\n",a);      /*结果输出6位十进制数 a =  1234*/
    printf("a = %06d\n",a);     /*结果输出6位十进制数 a = 001234*/
    printf("a = %2d\n",a);      /*a超过2位,按实际值输出 a = 1234*/
    printf("f = %f\n",f);       /*输出浮点数 f = 3.141593*/
    printf("f = 6.4f\n",f);     /*输出6位,其中小数点后4位的浮点数 f = 3.1416*/
    printf("x = %lf\n",x);      /*输出长浮点数 x = 0.123457*/
    printf("x = %18.16lf\n",x);
                    /*输出18位,其中小数点后16位的长浮点数 x = 0.1234567898765432*/
    printf("c = %c\n",c);       /*输出字符 c = A*/
    printf("c = %x\n",c);       /*输出字符的ASCII码值 c = 41*/
    system("pause");
return 0;
}
```

先手工分析再运行程序 3-1,看输出结果是什么?

编译成功后,运行程序 3-1,结果如图 3-1 所示。

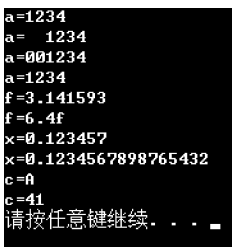


图 3-1　程序 3-1 运行结果

注意:上面结果中的地址值在不同计算机上可能不同。

printf()函数中常用的参数及其说明如表 3-1 和表 3-2 及图 3-2 所示。

表 3-1　printf()中常用的转换说明及其示例

转换说明	输出形式	应用例子	输出示例
%d	十进制 int 型	printf("sum=%d\n",sum);	sum=2008
%f	十进制 double 型	printf("a=%f\n",a);	a=6.280000
%c	单个字符	printf("It is %c\n",c);	It is M
%s	字符串	printf("**%s**\n",s);	**Hello!**
%o	无符号八进制数	printf("Oct=%o\n",oct);	Oct=176
%x	无符号十六进制数	printf("Hex=%x\n",hex);	Hex=96AF
%%	%	printf("a%%b=%d\n",d);	a%b=5

表 3-2　长度修正符

长度修正符	可修饰的格式码	参 数 类 型
l	d,i,o,u,x,X	long,
ll	d,i,o,u,x,X	long long int, unsigned long long int
h	d,i,o,u,x,X	short, unsigned short
hh	d,i,o,u,x,X	char, unsigned char
L	a,A,e,E,f,g,G	long double

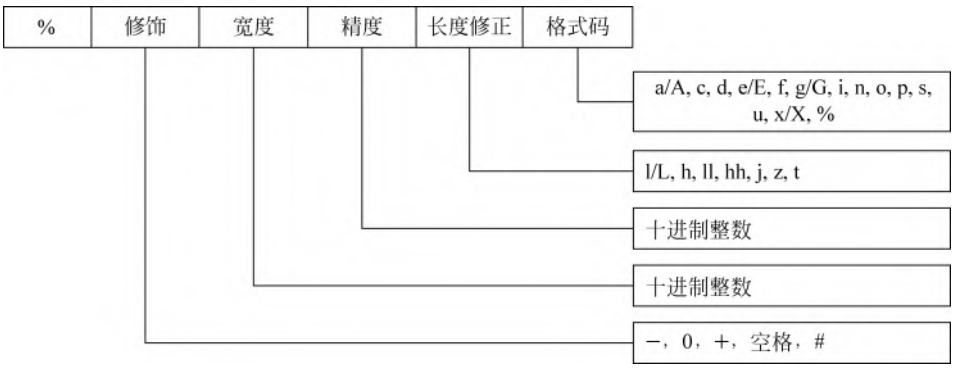


图 3-2　printf 格式字段结构

域宽与精度说明如下：

域宽与精度说明的格式为：m.n。

(1) m 为输出域宽，用字符数表示。对于实数，包括了一个小数点的位置。

(2) n 为精度，其用法有如下几种情形：

① 配合格式码 f、e/E 时，指定小数点后面的位数；未指定精度时，默认小数点后 6 位。

② 配合格式码 g/G 时，指定有效位的数目。

③ 作用于字符串时，精度符限制最大域宽。

④ 作用于整型数据时，指定必须显示的最小位数，不足时左侧补先导 0。

使用函数 printf()必须注意的问题如下：

(1) 格式控制串必须在双引号内。

(2) 格式控制字符串内的格式说明个数应与输出变量表里所列的变量个数吻合，类型一致。

(3) 对输出变量表里所列诸变量(表达式)，其计算顺序是自右向左进行的。因此，要注意右边的参数值是否会影响到左边的参数取值。

(4) 注意函数 printf()对输出变量表里所列诸变量(表达式)的计算顺序是自右向左进行的。

(5) 注意函数 printf()中格式说明%d 与输出变量的对应关系是从左往右一一对应的。

【实例分析 3-2】 printf 输出，详细代码如程序 3-2 所示。

程序 3-2　printf 输出

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 printf("%12.5f\n",123.1234567);
 printf("%12f\n",123.1234567);
 printf("%12.5g\n",123.1234567);
 printf("%5.10s%s\n","abcdefghijklm","a");
 printf("%12.8d\n",12345);
 system("pause");
return 0;
}
```

编译成功后，运行程序 3-2，结果如图 3-3 所示。

```
  123.12346
  123.123457
      123.12
abcdefghija
    00012345
请按任意键继续. . .
```

图 3-3　程序 3-2 运行结果

程序 3-2 运行结果分析如图 3-4 所示。

			1	2	3	.	1	2	3	4	6	域宽12，精度5
		1	2	3	.	1	2	3	4	5	7	域宽12，未指定精度，默认6位精度
						1	2	3	.	1	2	域宽12，有效位5位
a	b	c	d	e	f	g	h	i	j	a		最少5个字符，最大域宽10
				0	0	0	1	2	3	4	5	域宽12，必须最少显示8位，不足时左侧补0

图 3-4　printf 格式字段结构

3.1.2　scanf()函数

scanf()函数的功能是接受用户从键盘上输入的数据，按照格式控制符的要求进行类型转换，然后送到由对应参数指示的变量单元(内存空间)中去。其调用格式为：

```
scanf("<格式化字符串>",<地址表>);
```

格式输入函数 scanf()和格式输出函数 printf(),都在头文件 stdio.h 里。使用它们时,在程序的开始处,应该书写包含命令:#include <stdio.h>。由于这两个函数在 C 语言程序设计中使用得太频繁了,所以即使在程序中没有写这条包含命令,编译也不会出错(在 Visual C++中会出现警告)。

参数<格式化字符串>是用双引号括起的字符串常量,里面列出输入数据的格式说明和分隔符。

参数<地址表>列出存放输入数据的各个变量的地址。

格式化字符串包括以下三类不同的字符:

(1) 格式化说明符:格式化说明符与 printf()函数中的格式说明符基本相同。

(2) 空白字符:空白字符会使 scanf()函数在读操作中略去输入中的一个或多个空白字符。

(3) 非空白字符:一个非空白字符会使 scanf()函数在读入时剔除与这个非空白字符相同的字符。

特别提示 地址表是需要读入的所有变量的地址,而不是变量本身。这与 printf()函数完全不同,要特别注意。各个变量的地址之间用","分开。

【实例分析 3-3】 scanf()输出举例,详细代码如程序 3-3 所示。

程序 3-3 scanf()输出举例

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
     int i,j;
    printf("i,j=?\n");
    scanf("%d,%d",&i,&j);
    system("pause");
return 0;
}
```

程序 3-3 中的 scanf()函数先读一个整型数,然后把接着输入的逗号剔除,最后读入另一个整型数。如果","这一特定字符没有找到,scanf()函数就终止。若参数之间的分隔符为空格,则参数之间必须输入一个或多个空格。

scanf()函数中常用的参数及其说明如表 3-3 所示。

表 3-3　scanf()中常用的转换说明及其示例

转换说明	输入形式	应用例子	输入示例
%d	匹配可带符号的十进制整数	scanf("%d",&a);	输入 100,则 a 为 100
%f	匹配可带符号的浮点数	scanf("%f",&f);	输入 3.14, 则 a 为 3.140000
%c	匹配一个(默认)字符	scanf("%c",&c);	输入 A,则 c 为'A'
%s	匹配非空白字符序列	scanf("%s",line);	输入 string,则数组 line 中放置 string,末尾自动加上空格符
%o	匹配可带符号的八进制整数	scanf("%o",&u);	输入 754, 则 u 的值为八进制 754
%x	匹配可带符号的十六进制整数	scanf("%x",&x);	输入 123, 则 x 值为十六进制 123

1. 地址参数

C 语言允许程序员间接地使用内存地址,这个地址是通过对变量名"求地址"运算得到的。求地址的运算符为 &。

引申：&

如果将变量定义为平时的喝汤,总不至于直接用手捧着喝吧,一般都要用一个容器(比如碗)盛放吧,可以将 & 理解为一个容器(地址)。

例如：

```
short a;
float b;
```

&a 给出的是变量 a 两字节空间的首地址,&b 给出的是变量 b 四字节空间的首地址。

2. 格式说明字段

格式说明字段如图 3-5 所示,参数列表如表 3-4 所示。

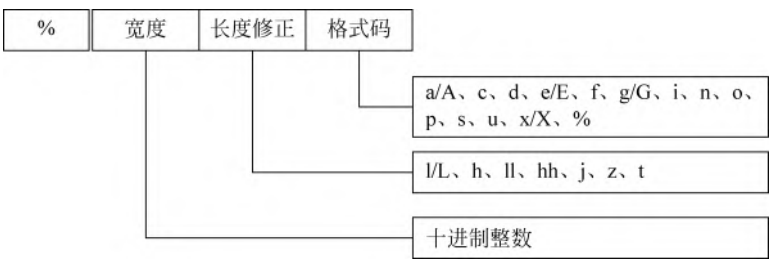


图 3-5　格式说明字段

表 3-4 格式说明字段参数列表

格式码	长度修正符	输入数据类型	说　明
d	无 hh h l ll	int char short long long long	输入带符号十进制整数,可选+或-
i	hh h l ll	char short long long long	输入带符号整数,可选+或-以及0(八进制)、0x(十六进制)
u	无	unsigned	输入带符号十进制整数
o	hh	unsigned char	输入无符号八进制整数,可选+或-
x	h l ll	unsigned short unsigned long unsigned long long	输入无符号十六进制整数,可选+或-
c	**无**	**char**	**读取字符**
s	无	字符串	连续读取字符,直到遇到文件结束符、空白或达到指定字段宽度
n	无 hh h l ll	int char short long long long	不读取字符,而是把scanf()所处理的字符总数写入到对应参数指定的变量中
p	无	地址	读取地址
f,e,g,a	无 l L	float double long double	读取带符号十进制实数
[]	无	字符搜索集合	只能输入定义在搜索集合中的字符,例如%[abcdefgh]或%[a-h]
%	无	%	读取%

注:表3-4中粗体标识的为常用的格式。

输入数据时,判断一个数据输入结束的几种方法。

(1) 在scanf()的格式控制字符串里,安排起数据分隔作用的一般字符。用户输入时,必须按照安排键入这些分隔字符。

【实例分析 3-4】 a、b为字符型变量。执行"scanf ("a=%c,b=%c",&a,&b);"后,要使a为'A'、b为'B',试问在键盘上的正确输入是什么。

解:scanf()中,格式控制字符串"a=%c,b=%c"里的a=,b=都是一般字符。输入时,

这部分字符必须按原样从键盘输入。

因此，在键盘上的正确输入应该是

```
a=A,b=B↙      /*用"↙"表示回车换行符*/
```

用户在键盘上输入后，变量 a 里就存放字母 A 的 ASCII 码值，b 里就存放字母 B 的 ASCII 码值。

(2) 在 scanf()的格式控制字符串里，不安排任何数据分隔符，这时就默认空格符、制表符或回车换行符为数据输入完毕的分隔符。例如：

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

中的格式控制字符串里没有一般字符。

这时，输完一个数据后，就可按空格键(或 Tab 键，或 Enter 键)，再输下一个数据，直到最后输入回车换行表示整个输入结束。

(3) 在格式符前冠以附加格式符，指明输入数据的域宽(正整数)。例如：

```
scanf ("%3d%2d%4d",&a,&b,&c);
```

%3d 里的 3 就是附加格式符，表示由 d 限定的十进制数为 3 位数。

因此，在键盘上连续输入 245321258 后回车，scanf()会把 245 存入变量 a，把 32 存入变量 b，把 1258 存入变量 c。

注意：(1) 所有数据从键盘输入完毕后，必须以回车换行(即键盘上的 Enter 键)作为整个数据输入的结束。

(2) <输入地址列表>中给出的必须是一个变量地址，因此在变量名的前面不要忘记加上取地址符 &。

(3) <格式控制字符串>中给出的格式说明个数(即给出的"%"个数)，必须与输入地址列表中所列变量地址的个数相一致，因为它们之间是一一对应的。

【实例分析 3-5】 scanf()函数，详细代码如程序 3-4 所示。

程序 3-4　scanf()函数

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 double a,b,c,d;
 scanf("%f%f",&a,&b);
 printf("\na=%1f,b=%1f\n",a,b);
 scanf("%lf%lf",&c,&d);
 printf("\nc=%lf,d=%lf\n",c,d);
```

```
 system("pause");
return 0;
}
```

输入图 3-6 所示数据，编译成功后，运行程序 3-4，结果也如图 3-6 所示。

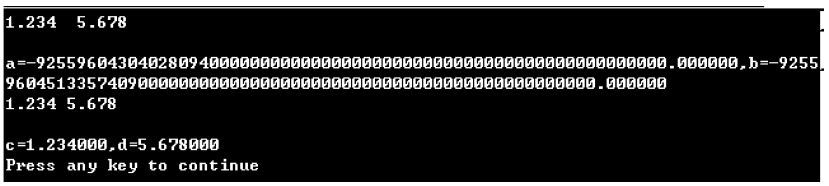


图 3-6　程序 3-4 结果

实际使用 scanf()函数时，若涉及多个字符变量输入，易出现问题。

【实例分析 3-6】 多个 scanf()函数连续给多个字符变量输入举例，详细代码如程序 3-5 所示。

程序 3-5　多个 scanf()函数连续给多个字符变量输入举例

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char c1,c2;
    scanf("%c",&c1);
    scanf("%c",&c2);
    printf("c1 is %c,c2 is %c",c1,c2);
    system("pause");
return 0;
}
```

运行该程序，输入一个字符 A 后回车(要完成输入必须回车)，在执行 scanf ("%c", &c1)时，给变量 c1 赋值"A"，但回车符仍然留在缓冲区内，执行输入语句 scanf("%c", &c2)时，变量 c2 输出的是一空行，如果输入 AB 后回车，那么输出结果为：c1 is A，c2 isB。

要解决以上问题，可以在输入函数前加入清除函数 fflush()。

引申：fflush()函数

在 C 语言中使用 scanf()函数在键盘上接受字符型数据时，必须用 fflush(stdin)清除上一次存储的数据，否则字符型数据不会正确地存储。

fflush()函数详细解释见 MSDN(Microsoft Developer Network)文档。

【实例分析 3-7】 修改程序 3-5 以消除回车符留在缓冲区内的情况，详细代码如程序 3-6 所示。

程序 3-6　程序 3-5 的改进

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char c1,c2;
scanf("%c",&c1);
fflush(stdin);
    scanf("%c",&c2);
    printf("c1 is %c,c2 is %c",c1,c2);
    system("pause");
return 0;
}
```

3.2　getchar()函数与 putchar()函数

为了方便字符的输入与输出，标准 C 还提供了两个简单的库函数：getchar()和 putchar()。它们的原型分别为

```
int getchar(void);
int putchar(int c);
```

字符输入函数 getchar()和字符输出函数 putchar()，都在头文件“stdio.h”里。程序中使用时，必须在开始处书写一条包含命令

```
#include<stdio.h>
```

3.2.1　字符输入函数 getchar()

字符输入函数：getchar()

```
<变量> = getchar();
```

即把由 getchar()返回的第 1 个字符，存入赋值语句左边的＜变量＞。

功能：使程序处于等待用户从键盘进行输入的状态。输入以在键盘上按回车换行键结束，随之返回输入的第 1 个字符。该函数没有参数。

【实例分析 3-8】 编写程序，从键盘接收一个字符的输入，然后打印输出，详细代码如程序 3-7 所示。

程序 3-7　getchar（）举例

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char ch;
    ch = getchar();
    printf("ch = %c\n",ch);
    system("pause")
return 0;
}
```

编译成功后，运行程序 3-7，结果如图 3-7 所示。

图 3-7　程序 3-7 运行结果

注意：(1) 在执行 getchar()时，虽然读入一个字符，就回显一个字符，但并不是从键盘按一个字符，该字符就立即送给一字符变量，而是等到按回车键后，才将一行字符输入缓冲区，然后 getchar()函数从缓冲区中取一个字符给一个字符变量。这种情况称为行缓冲。

(2) 程序里通过调用字符输入函数 getchar ()，将键盘输入的字符序列中的第 1 个字符存入到 ch。

(3) getchar()函数应以 Enter 键作为输入的结束。该函数只把输入的第 1 个字符返回。按 Enter 键前输入的其他字符（连同 Enter 键在内）可被另外的 getchar()函数接收。

3.2.2　字符输出函数：putchar()

字符输出函数 putchar()

调用形式：

```
putchar (<字符变量名>);
```

或

```
putchar (<字符常量>);
```

功能：将“字符变量名”或“字符常量”在显示器上显示（输出）。

【实例分析 3-9】　putchar()应用举例，详细代码如程序 3-8 所示。

程序 3-8　putchar ()应用举例-1

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char ch;
    ch = getchar ();
    printf ("ch = %c\n",ch);              //与 putchar (ch); 功能类似
    system("pause")
return 0;
}
```

【实例分析 3-10】　putchar()应用举例，详细代码如程序 3-9 所示。

程序 3-9　putchar ()应用举例-2

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char ch1,ch2,ch3;
    ch1 = getchar();
    ch2 = getchar();
    ch3 = getchar();
    putchar (ch1);
    putchar (ch2);
    putchar (ch3);
    putchar ('\n');
    system("pause");
return 0;
}
```

编译成功后，运行程序 3-9，结果如图 3-8 所示。

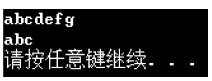


图 3-8　程序 3-9 运行结果

【实例分析 3-11】 putchar()应用举例，详细代码如程序 3-10 所示。

程序 3-10 putchar ()应用举例-3

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char c;                          /*定义字符变量*/
    c='B';                           /*给字符变量赋值*/
    putchar(c);                      /*输出该字符*/
    putchar('\x42');                 /*输出字母 B*/
    putchar(0x42);                   /*直接用 ASCII 码值输出字母 B*/
    system("pause")
return 0;
}
```

从程序 3-10 中的连续 4 个字符输出函数语句，可以分清字符变量的不同赋值方法。

编译成功后，运行程序 3-10，结果如图 3-9 所示。

```
BBB请按任意键继续. . .
```

图 3-9 程序 3-10 运行结果

3.3 标准程序解读

【实例分析 3-12】 实现 A+B，详细代码如程序 3-11 所示。

程序 3-11 A+B 问题

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",a+b);
    system("pause");
return 0;
}
```

程序 3-11 要求一定要记住，到此为止，可以对此程序做较为全面的解释。

编程遵循“有始有终、前后一致、先定义(引用)后使用”的原则。同时编程过程中，系统中没有提供的函数，自己要定义并且实现。

3.3.1　头文件

#include 包含的是头文件，引用它的源文件或者函数可以使用这个头文件所声明的数据，再到定义的源文件中去找定义文件。

例如，在程序中出现 printf()、scanf()、getchar()和 putchar()等输入与输出类型函数就一定要在头文件中加入#include<stdio.h>。

stdio 意思是基本输入与输出函数，是由 standard(标准)和 input/output(输入/输出)两个单词中 standard 的前三个字母和 input/output 两个单词的首字母组成。这也符合变量的命名规则。

引申：头文件

头文件就如同盖房子，如果没有砖，可以自己造，但代价太大，一般都是从砖厂取砖，因此“砖”就像在 main 函数里面使用的数据，直接把“砖厂”的名字在最上面引用，那么使用砖就直接从指定的砖厂拿过来就行了。

在程序中出现数学类的公式，就一定要在头文件中加入#include<math.h>。

注意：#include<xxx.h> 与 #include"xxx.h"的区别：

#include "xxx.h"用引号，代表编译程序会优先在程序的本地目录搜索这个文件，找不到时再搜索系统目录。

#include <xxx.h>用尖括号，代表编译程序只会在系统目录(系统环境变量和编译本身设置的默认搜索目录)搜索这个文件。

总体来说，一般是用双引号来引用自己编写的文件，而用尖括号引用系统标准的文件。

特别提示　具体的不同头文件包含哪些函数，见附录 D。

3.3.2　函数

C 语言的设计原则是把函数作为程序的构成模块。main()函数称之为主函数，一个 C 程序总是从 main()函数开始执行的。

1. main()函数的形式

在最新的 C99 标准中，只有以下两种定义方式是正确的。

```
int main()                                  /* 无参数形式 */
{
    …
    return 0;
}
int main( int argc,char *argv[] )           /* 带参数形式 */
{
    …
    return 0;
}
```

int 指明了 main()函数的返回类型，函数名后面的圆括号一般包含传递给函数的信息。void 表示没有给函数传递参数，一般情况不写 void 也可以。关于带参数的形式，我们不做讨论。

浏览老版本的 C 代码，将会发现程序常常以

```
main()
```

这种形式开始。C90 标准允许这种形式，但是 C99 标准不允许。因此即使当前的编译器允许，也不要这么写。

你还可能看到过另一种形式：

```
void main()
```

有些编译器允许这种形式，但是还没有任何标准考虑接受它。C++之父 Bjarne Stroustrup 在他的主页上的 FAQ 中明确地表示：void main()的定义从来就不存在于 C++或者 C。所以，编译器不必接受这种形式，并且很多编译器也不允许这么写。

坚持使用标准的意义在于：当把程序从一个编译器移到另一个编译器时，照样能正常运行。

2. main()函数的返回值

从前面我们知道 main()函数的返回值类型是 int 型，而程序最后的 return 0；正与之遥相呼应，0 就是 main()函数的返回值。那么这个 0 返回到那里呢？返回给操作系统，表示程序正常退出。因为 return 语句通常写在程序的最后，不管返回什么值，只要到达这一步，就说明程序已经运行完毕。而 return 的作用不仅在于返回一个值，还在于结束函数的执行。

3. 函数体

花括号内的是函数体，要实现的主要功能都包含在里面。一定要记住，编程是所有变量、函数先定义后使用的原则。

4. system 函数

在有些编译环境例如 Dev-C++中，如果没有 system("pause")语句，当程序正确运行需要查看输出结果时，屏幕会一闪而过，主要原因是在人机交互过程中，为了让人能够查看到结果，需要用 system("pause")(此条语句放在 return 0 的前一条语句的位置)语句将屏幕截取下来，当然用此语句最好加上头文件 #include<stdlib>，有些编译环境中不需要加入头文件 #include<stdlib>和 system("pause")，例如 Visual C++ 6.0。但为了保险起见，建议加入此语句。

此语句在后面也会提到，如果是使用在线编译系统，需要将此语句去掉，否则提交无法成功通过。

第 4 章 控 制 结 构

本章首先介绍自增和自减运算。主要包括以下内容：

(1) 自增运算，即＋＋；

(2) 自减运算，即－－。

然后通过一个“灌汤包”的例子，形象地描述理解控制结构的含义。

最后介绍控制结构。主要包括以下内容：

(1) 顺序结构；

(2) 分支结构；

(3) 循环结构；

(4) continue 语句和 break 语句。

4.1 从＋1 开始

自增 1 运算符记为“＋＋”，其功能是使变量的值自增 1；自减 1 运算符记为“－－”，其功能是使变量值自减 1。

自增 1，自减 1 运算符均为单目运算，都具有右结合性。有以下几种形式：

(1) ＋＋i：i 自增 1 后再参与其他运算。

(2) －－ i：i 自减 1 后再参与其他运算。

(3) i＋＋：i 参与运算后，i 的值再自增 1。

(4) i－－：i 参与运算后，i 的值再自减 1。

(1)和(2)被称为前置运算：先自增/减 1，再参与运算；

(3)和(4)被称为后置运算：先参与运算，再自增/减 1。

在理解和使用上容易出错的是后置运算(i＋＋和 i－－)。特别是当它们出现在较复杂的表达式或语句中时，常常难以弄清，因此应仔细分析。

【实例分析 4-1】 自增与自减运算，详细代码如程序 4-1 所示。

程序 4-1　自增与自减运算举例

```
第 1 行    #include<stdio.h>
第 2 行    #include<stdlib.h>
第 3 行    int main(){
第 4 行        int i=8;
第 5 行        printf("%d\n",++i);
第 6 行        printf("%d\n",--i);
第 7 行        printf("%d\n",i++);
第 8 行        printf("%d\n",i--);
第 9 行        printf("%d\n",-i++);
第 10 行       printf("%d\n",-i--);
第 11 行       system("pause");
第 12 行   return 0;
第 13 行   }
```

编译成功后,运行程序 4-1,结果如图 4-1 所示。

图 4-1　程序 4-1 运行结果

程序 4-1 结果分析:

i 的初值为 8;

第 5 行 i 加 1 后输出故为 9;

第 6 行 i 减 1 后输出故为 8;

第 7 行输出 i 为 8 之后再加 1(为 9);

第 8 行输出 i 为 9 之后再减 1(为 8);

第 9 行输出－8 之后再加 1(为 9);

第 10 行输出－9 之后再减 1(为 8)。

【实例分析 4-2】 自增与自减混合运算,详细代码如程序 4-2 所示。

程序 4-2　自增与自减混合运算举例

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int i=5,j=5,p,q;
    p=(i++)+(i++)+(i++);
    q=(++j)+(++j)+(++j);
    printf("p=%d\n q=%d\n i=%d\n j=%d\n",p,q,i,j);
    system("pause");
return 0;
}
```

编译成功后并运行程序 4-2,结果如图 4-2 所示。

程序 4-2 结果分析：

对于 p=(i++)+(i++)+(i++),先计算(i++)+(i++),因为是“后自加”,等价于 5+5,结果为 10；然后计算 10+(i++),等价于 10+5,结果为 15。

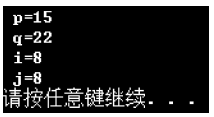


图 4-2　程序 4-2 运行结果

对于 q=(++j)+(++j)+(++j),先计算(++j)+(++j),因为是“前自加”,要先计算两次++j,此时 j=7,然后相加,相当于 7+7,结果为 14；然后计算 14+(++j),相当于 14+8,结果为 22。

注意：(1) 自增、减运算符只用于变量,而不能用于常量或表达式。

例如：

6++,(a+b)++,(-i)++都不合法。

(2) ++,--的结合方向是“自右向左”(与一般算术运算符不同)。

例如：

-i++ -(i++)合法。

【实例分析 4-3】 自增/减输出顺序,详细代码如程序 4-3 所示。

程序 4-3　printf()函数与自增/减输出顺序

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int x = 4;
    printf("%d\t %d\t %d\n",++x,++x,--x);
    system("pause");
return 0;
}
```

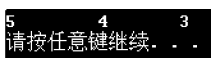


图 4-3　程序 4-3 运行结果

编译成功后,运行程序 4-3,结果如图 4-3 所示。

程序 4-3 结果分析：

注意函数 printf()对输出变量表里所列诸变量(表达式)的计算顺序是自右向左进行的,如图 4-4 所示。

因此,在程序 4-3 中 printf() 输出前,应该先计算--x,再计算中间的++x,最后计算左边的++x。所以,该程序执行后的输出是

```
5    4    3
```

而不是

```
5    6    5
```

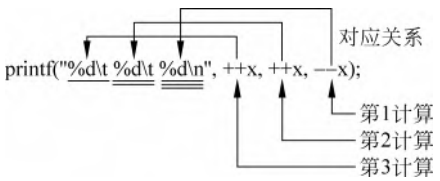


图 4-4　printf()函数与自增/减运算符输出顺序

【拓展训练 4-1】 实验 4-1 分析并编程实现：printf ("%d\t %d\t %d\n",x ++,x ++,x --);。

4.2　灌汤包(如图 4-5)

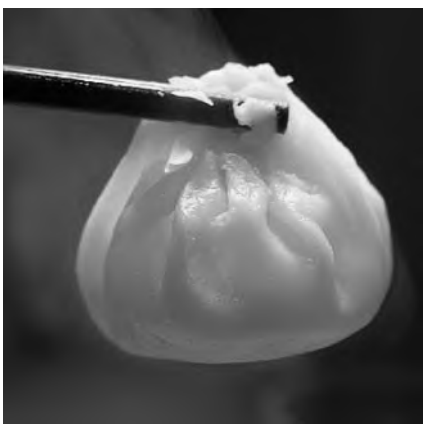


图 4-5　灌汤包

吃灌汤包方法：

轻轻提
慢慢移
先开窗
后喝汤
===
张开嘴
慢慢咬

将吃包子过程分为两个阶段：喝汤和吃包子。

一个大人带着一个小孩走进包子铺，吃包子。

(1) 吃包子的过程分为六个步骤，每一步骤的顺序不能打乱或者颠倒，如何解释？

——顺序结构

(2) 大人要吃葱香猪肉包，小孩想吃葱香牛肉包。

——分支结构

（3）老板问吃几个包子，大人说吃十个包子，小孩说吃饱就行。

——循环结构

（4）大人吃到第三个包子时，刚把包子汤喝完，小孩不小心把大人的包子碰到地上，大人接着吃剩余的包子。

——continue 语句

（5）然而大人吃到第六个包子时，刚把包子汤喝完，小孩不仅把大人的包子打掉，而且把桌子上的所有包子全部打掉在地。

——break 语句

讲完本章的理论知识后，大家再回头看这个引子，就会容易理解控制结构语句的使用方法和确切含义。

4.3　顺序结构

顺序结构是最简单的程序结构，也是最常用的程序结构，只要按照解决问题的顺序写出相应的语句就行，它的执行顺序是自上而下，依次执行。

【实例分析 4-4】　顺序结构来实现吃包子的过程，详细代码如程序 4-4 所示。

程序 4-4　吃包子

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i;
    printf("轻轻提\n");
    printf("慢慢移\n");
    printf("先开窗\n");
    printf("后喝汤\n");
    printf("张开嘴\n");
    printf("慢慢咬\n");
    system("pause");
return 0;
}
```

编译成功后，运行程序 4-4，结果如图 4-6 所示。

图 4-6　程序 4-4 运行结果

4.4 分支结构

4.4.1 if 语句

用 if 语句可以构成分支结构。它根据给定的条件进行判断，以决定执行某个分支程序段。C 语言的 if 语句有三种基本形式。

(1) 第一种形式为基本形式：if

```
if(表达式)语句
```

功能：如果表达式的值为真，则执行其后的语句，否则不执行该语句。其执行过程如图 4-7 所示。

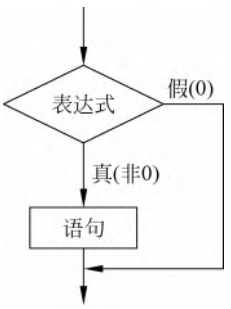


图 4-7 if 形式流程图

【实例分析 4-5】 if 形式——寻找两个数最大者，详细代码如程序 4-5 所示。

程序 4-5 if 形式——寻找两个数最大者

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a,b,max;
    printf("请输入两个整数:\n ");
    scanf(" %d %d",&a,&b);
    max = a;
    if (max<b) max = b;
    printf("max = %d\n",max);
    system("pause");
return 0;
}
```

编译成功后，运行程序 4-5，结果如图 4-8 所示。

程序 4-5 中，输入两个数 a、b。把 a 先赋予变量 max，再用 if 语句判别 max 和 b 的大小，如 max 小于 b，则把 b 赋予 max。因此 max 中总是大数，最后输出 max 的值。

(2) 第二种形式为：if-else

```
if(表达式)
    语句 1;
else
    语句 2;
```

功能：如果表达式的值为真，则执行语句 1，否则执行语句 2。其执行过程如图 4-9 所示。

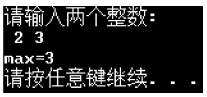


图 4-8　程序 4-5 运行结果

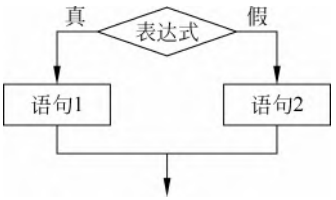


图 4-9　if-else 形式流程图

【实例分析 4-6】 if-else 形式——输入两个整数，输出其中的大数，详细代码如程序 4-6 所示。

程序 4-6　if-else 形式——输入两个整数，输出其中的大数

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a,b,max;
    printf("请输入两个整数:\n ");
      scanf(" %d %d",&a,&b);
    if(a > b)
        printf("max = %d\n",a);
    else
        printf("max = %d\n",b);
    system("pause");
return 0;
}
```

输入两个整数，输出其中的大数。改用 if-else 语句判别 a，b 的大小，若 a 大，则输出 a，否则输出 b。

(3) 第三种形式为：if-else-if

前两种形式的 if 语句一般都用于两个分支的情况。当有多个分支选择时，可采用 if-else-if 语句，其一般形式为

```
if(表达式 1)
    语句 1;
else  if(表达式 2)
    语句 2;
else  if(表达式 3)
    语句 3;
    …
else  if(表达式 m)
    语句 m;
else
```

语句 n;

功能：依次判断表达式的值，当出现某个值为真时，则执行其对应的语句。然后跳到整个 if 语句之外继续执行程序。如果所有的表达式均为假，则执行语句 n。然后继续执行后续程序。if-else-if 语句的执行过程如图 4-10 所示。

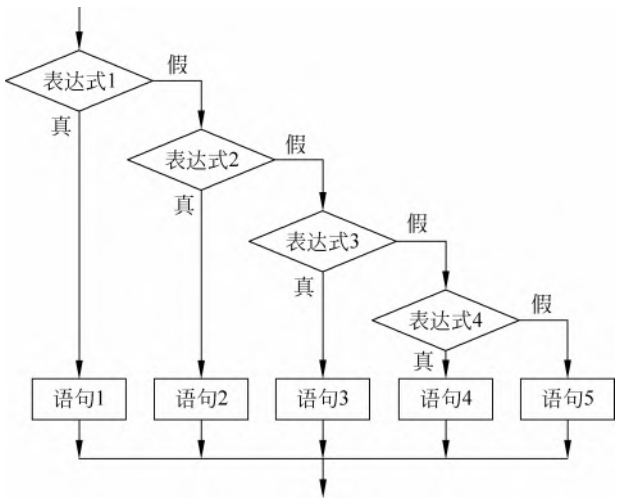


图 4-10　if-else-if 形式流程图

【实例分析 4-7】 if-else-if 形式——判别键盘输入字符的类别，详细代码如程序 4-7 所示。

程序 4-7　if-else-if 形式-判别键盘输入字符的类别

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    char c;
    printf("请输入一个字符");
    c = getchar();
    if(c < 32)
        printf("输入的是控制字符\n");
    else if(c >= '0'&&c <= '9')
        printf("输入的是数字\n");
    else if(c >= 'A'&&c <= 'Z')
        printf("输入的是一个大写字母\n");
    else if(c >= 'a'&&c <= 'z')
        printf("输入的是一个小写字母\n");
    else
        printf("输入的是其他字符\n");
        system("pause");
return 0;
}
```

程序 4-7 要求判别键盘输入字符的类别。可以根据输入字符的 ASCII 码来判别类型。由 ASCII 码表可知 ASCII 值小于 32 的为控制字符。在“0”和“9”之间的为数字，在“A”和“Z”之间为大写字母，在“a”和“z”之间为小写字母，其余为其他字符。这是一个多分支选择的问题，用 if-else-if 语句编程，判断输入字符 ASCII 码所在的范围，分别给出不同的输出。例如输入为“g”，输出显示它为小写字符。

注意：(1) 在三种形式的 if 语句中，在 if 关键字之后均为表达式。该表达式通常是逻辑表达式或关系表达式，也可以是其他表达式，如赋值表达式等，甚至也可以是一个变量。例如：

```
if(a = 5)语句;
if(b)语句;
```

都是允许的。只要表达式的值为非 0，即为“真”。例如：

```
if(a = 5)…;
```

中表达式的值永远为非 0，所以其后的语句总是要执行的，当然这种情况在程序中不一定会出现，但在语法上是合法的。

又如，有程序段：

```
if(a = b) printf(" % d",a);
else printf("a = 0");
if(a = b)
printf(" % d",a);
else
printf("a = 0");
```

本语句的语义是把 b 值赋予 a，如为非 0 则输出该值，否则输出“a＝0”字符串。这种用法在程序中是经常出现的。

(2) 在 if 语句中，条件判断表达式必须用括号括起来，在语句之后必须加分号。

(3) 在 if 语句的三种形式中，所有的语句应为单个语句，如果想在满足条件时执行一组(多个)语句，则必须把这一组语句用{}括起来组成一个复合语句。但要注意的是在}之后不能再加分号。例如：

```
if(a > b){a++;
    b++;
}else{
    a = 0;
    b = 10;
}
```

(4) if 语句的嵌套

当 if 语句中的执行语句又是 if 语句时，则构成了 if 语句的嵌套。其一般形式可表示如下：

```
if(表达式)
  if 语句;
```

或者为

```
if(表达式)
 if 语句;
    else
if 语句;
```

在嵌套内的 if 语句可能又是 if-else 型的，这将会出现多个 if 和多个 else 重叠的情况，这时要特别注意 if 和 else 的配对问题。例如：

```
if(表达式 1)
if(表达式 2)
  语句 1;
    else
  语句 2;
```

其中的 else 究竟是与哪一个 if 配对呢？应该理解为

```
if(表达式 1)
if(表达式 2)
  语句 1;
    else
  语句 2;
```

还是应理解为

```
if(表达式 1)
if(表达式 2)
  语句 1;
    else
  语句 2;
```

为了避免这种二义性，C 语言规定，else 总是与它前面最近的 if 配对，因此对上述例子应按前一种情况理解。

【实例分析 4-8】 if 语句的嵌套结构——比较两个数的大小关系，详细代码如程序 4-8 所示。

程序 4-8　if 语句的嵌套结构——比较两个数的大小关系

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a,b;
```

```
    printf("输入两个数字 A,B:");
    scanf("%d%d",&a,&b);
    if(a!=b)
    if(a>b)   printf("A>B\n");
    else       printf("A<B\n");
    else       printf("A=B\n");
    system("pause");
return 0;
}
```

比较两个数的大小关系。程序 4-8 中用了 if 语句的嵌套结构。采用嵌套结构实质上是为了进行多分支选择，有三种选择即 A＞B、A＜B 或 A＝B。这种问题用 if-else-if 语句也可以完成，而且程序更加清晰。因此，在一般情况下较少使用 if 语句的嵌套结构，以使程序更便于阅读理解。

【实例分析 4-9】 if 语句的嵌套结构——比较两个数的大小关系改进，详细代码如程序 4-9 所示。

程序 4-9　if 语句的嵌套结构——比较两个数的大小关系改进

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int a,b;
    printf("please input A,B:       ");
    scanf("%d%d",&a,&b);
    if(a==b) printf("A=B\n");
    else if(a>b)   printf("A>B\n");
    else  printf("A<B\n");
    system("pause")
return 0;
}
```

4.4.2　switch 语句

C 语言还提供了另一种用于多分支选择的 switch 语句，其一般形式为

```
switch(表达式){
    case 常量表达式 1:  语句 1;
    case 常量表达式 2:  语句 2;
    …
    case 常量表达式 n:  语句 n;
    default:  语句 n+1;
}
```

流程：

(1) 先求表达式的值。

(2) 依次比较表达式和各常量表达式的值。

(3) 如果和第 i 个常量表达式相等，则执行第 i 条以后的语句。

(4) 如果不相等，则执行 default 以后的语句。

【实例分析 4-10】 switch 语句计算表达式的值，详细代码如程序 4-10 所示。

程序 4-10　switch 语句计算表达式的值

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a;
    printf("输入一个大于 0 的正整数:");
    scanf("%d",&a);
    switch (a){
        case 1:printf("Monday\n");
        case 2:printf("Tuesday\n");
        case 3:printf("Wednesday\n");
        case 4:printf("Thursday\n");
        case 5:printf("Friday\n");
        case 6:printf("Saturday\n");
        case 7:printf("Sunday\n");
        default:printf("error\n");
    }
    system("pause");
return 0;
}
```

程序 4-10 是计算表达式的值，并逐个与其后的常量表达式值相比较，当表达式的值与某个常量表达式的值相等时，即执行其后的语句，不再进行判断，继续执行后面所有 case 后的语句。如表达式的值与所有 case 后的常量表达式值均不相同时，则执行 default 后的语句。

图 4-11　程序 4-10 运行结果

编译成功后，运行程序 4-10，结果如图 4-11 所示。

程序 4-10 要求输入一个大于 0 的正整数，输出一个英文单词。但是当输入 3 之后，却执行了 case3 及以后的所有语句，输出了 Wednesday 及以后的所有单词。这当然是不希望的。为什么会出现这种情况呢？这恰恰反应了 switch 语句的一个特点。在 switch 语句中，“case 常量表达式”只相当于一个语句标号，表达式的值和某标号相等则转向该标号执行，但不能在执行完该标号的语句后，自动跳出整个 switch 语句，所以出现了继续执行所有后面 case 语句的情况。这与前面介绍的 if 语句完全不同，应特别注意。

为了避免上述情况，C 语言还提供了一种 break 语句，专用于跳出 switch 语句。break 语句只有关键字 break，没有参数。在 4.6.2 将详细介绍。修改程序 4-10，在每一 case 语句之后增加 break 语句，使每一次执行之后均可跳出 switch 语句，从而避免输出不应有的结果。

【实例分析 4-11】 switch 语句计算表达式的值的改进，详细代码如程序 4-11 所示。

程序 4-11　计算表达式的值的改进

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a;
    printf("input integer number:    ");
    scanf("%d",&a);
    switch (a){
        case 1:printf("Monday\n");        break;
        case 2:printf("Tuesday\n");       break;
        case 3:printf("Wednesday\n");     break;
        case 4:printf("Thursday\n");      break;
        case 5:printf("Friday\n");        break;
        case 6:printf("Saturday\n");      break;
        case 7:printf("Sunday\n");        break;
        default:printf("error\n");
    }
   system("pause")
return 0;
}
```

编译成功后，运行程序 4-11，结果如图 4-12 所示。

图 4-12　程序 4-11 运行结果

在使用 switch 语句时还应注意以下几点：

(1) 在 case 后的各常量表达式的值不能相同，否则会出现错误。

(2) 在 case 后，允许有多个语句，可以不用{}括起来。

(3) 各 case 和 default 子句的先后顺序可以变动，而不会影响程序执行结果。

(4) default 子句可以省略。

【实例分析 4-12】 计算器程序。用户输入运算数和四则运算符，输出计算结果。详细代码如程序 4-12 所示。

程序 4-12　计算器程序

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    float a,b;
    char c;
    printf("输入表达式: a+(-,*,/)b \n");     //或加(+)或减(-)或乘(*)或除(/)
    scanf("%f%c%f",&a,&c,&b);
    switch(c){
        case '+': printf("%f\n",a+b);break;
        case '-': printf("%f\n",a-b);break;
        case '*': printf("%f\n",a*b);break;
        case '/': printf("%f\n",a/b);break;
        default: printf("输入错误\n");
    }
   system("pause");
return 0;
}
```

编译成功后,运行程序 4-12,结果如图 4-13 所示。

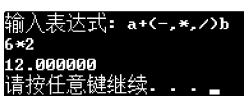


图 4-13　程序 4-12 运行结果

程序 4-12 可用于四则运算求值。switch 语句用于判断运算符,然后输出运算值。当输入运算符不是+、-、* 或/时给出错误提示。

4.5　循环结构

循环结构是程序中一种很重要的结构。其特点是,在给定条件成立时,反复执行某程序段,直到条件不成立为止。给定的条件称为循环条件,反复执行的程序段称为循环体。

C 语言提供了多种循环语句,可以组成各种不同形式的循环结构。

while 语句;

do-while 语句;

for 语句。

4.5.1　while 语句

while 语句的一般形式为

```
while(表达式)语句;
```

其中表达式是循环条件,语句为循环体。

功能：计算表达式的值，当值为真（非 0）时，执行循环体语句。其执行过程如图 4-14 所示。

【实例分析 4-13】 大人吃十个包子，吃包子的过程屏蔽，直接用 printf("吃第%d 个包子\n",i)；表示吃包子的过程。详细代码如程序 4-13 所示。

程序 4-13　while 语句——吃十个包子

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i = 1;
    while(i <= 10)
    {
        printf("吃第 %d 个包子\n",i);
        i++;
    }
    printf("\n");
    system("pause");
return 0;
}
```

【实例分析 4-14】 用 while 语句计算从 1 加到 100 的值。用传统流程图和 N-S 结构流程图（即盒图或 CHAPIN 图）表示算法，如图 4-15 所示。

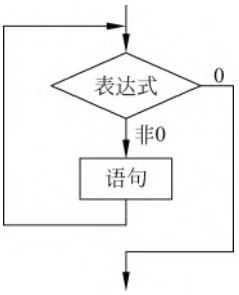


图 4-14　while 形式流程图

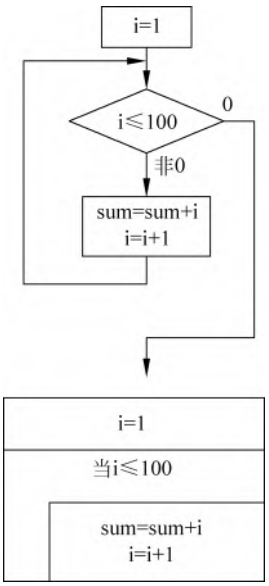


图 4-15　传统流程图和 N-S 结构流程图

详细代码如程序 4-14 所示。

程序 4-14　用 while 语句计算从 1 加到 100 的值

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i,sum = 0;
i = 1;
while(i <= 100){
    sum = sum + i;
    i++;
}
printf("%d\n",sum);
system("pause");
return 0;
}
```

编译成功后,运行程序 4-14,结果如图 4-16 所示。

```
5050
请按任意键继续. . .
```

图 4-16　程序 4-14 运行结果

使用 while 语句应注意以下两点:

(1) while 语句中的表达式一般是关系表达或逻辑表达式,只要表达式的值为真(非 0)即可继续循环。

【实例分析 4-15】 超出基本整型数据的最大允许值,详细代码如程序 4-15 所示。

程序 4-15　while 语句范例

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a = 0,n;
    printf("请输入一个整数: \n");
    scanf("%d",&n);
    while (n-- ) printf("%d  ",a++ * 2);
    system("pause");
return 0;
}
```

```
请输入一个整数:6
0  2  4  6  8  10
请按任意键继续. . .
```

图 4-17　程序 4-15 运行结果

编译成功后,运行程序 4-15,结果如图 4-17 所示。

程序 4-15 分析:输入一个 n(此例 n 为 6),将执行 n 次循环,每执行一次,n 值减 1。循环体输出表达式 a++ * 2 的值。该表达式等效于(a * 2; a++)。

(2) 循环体如果包括有一个以上的语句,则必须用{}括起来,组成复合语句。

4.5.2　do-while 语句

do-while 语句的一般形式为

```
do
语句
while(表达式);
```

这个循环与 while 循环的不同在于：它先执行循环中的语句,然后再判断表达式是否为真,如果为真则继续循环；如果为假,则终止循环。因此,do-while 循环至少要执行一次循环语句。其执行过程如图 4-18 所示。

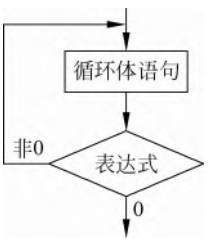


图 4-18　do-while 循环流程图

【实例分析 4-16】 用 do-while 语句计算从 1 加到 100 的值,详细代码如程序 4-16 所示。

程序 4-16　用 do-while 语句计算从 1 加到 100 的值

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i,sum = 0;
    i = 1;
    do{
        sum = sum + i;
        i++;
        }while(i <= 100);
    printf("%d\n",sum);
    system("pause");
return 0;
}
```

同样,当有许多语句参加循环时,要用“{”和“}”把它们括起来。

【实例分析 4-17】 while 和 do-while 循环比较之 while 语句。详细代码如程序 4-17 所示。

程序 4-17　while 和 do-while 循环比较之 while

```
#include <stdio.h>
#include <stdlib.h>
int main(){
int sum = 0,i;
scanf("%d",&i);
while(i <= 10){
    sum = sum + i;
    i++;
}
printf("sum = %d",sum);
system("pause");
return 0;
}
```

【实例分析 4-18】 while 和 do-while 循环比较之 do-while 语句。详细代码如程序 4-18 所示。

程序 4-18　while 和 do-while 循环比较之 do-while

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int sum = 0,i;
    scanf("%d",&i);
    do{
        sum = sum + i;
        i++;
        } while(i <= 10);
    printf("sum = %d",sum);
    system("pause");
return 0;
}
```

从程序 4-17 和程序 4-18，可以看出 while 循环与 do-while 循环的区别与联系：

(1) 第一次条件为真时，while，do-while 等价。

(2) 第一次条件为假时，二者不同。

while 循环先判条件，后执行循环体。

do-while 循环先执行循环体，后判条件。

4.5.3　for 语句

在 C 语言中，for 语句使用最为灵活，它完全可以取代 while 语句。它的一般形式为

```
for(表达式 1;表达式 2;表达式 3)语句
```

for 语句执行过程

(1) 求解表达式 1。

(2) 求解表达式 2,若其值为真(非 0),则执行 for 语句中指定的内嵌语句,然后执行下面第(3)步;若其值为假(0),则结束循环,转到第(5)步。

(3) 求解表达式 3。

(4) 转回上面第(2)步继续执行。

(5) 循环结束,执行 for 语句下面的一个语句。

其执行过程如图 4-19 所示。

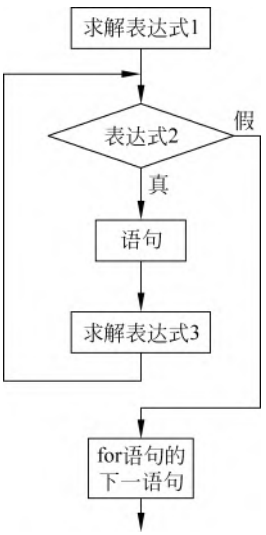


图 4-19　for 循环流程图

for 语句最简单的应用形式也是最容易理解的形式,如下:

```
for(循环变量赋初值;循环条件;循环变量增量) 语句
```

循环变量赋初值总是一个赋值语句,用来给循环控制变量赋初值;循环条件是一个关系表达式,决定什么时候退出循环;循环变量增量,定义循环控制变量每循环一次后按什么方式变化。这三个部分之间用分号";"分开。例如:

```
for( i = 1; i <= 100; i++)   sum = sum + i;
```

先给 i 赋初值 1,判断 i 是否小于等于 100,若是则执行语句,之后值增加 1。再重新判断,直到条件为假,即 i>100 时,结束循环。相当于

```
i = 1;
while(i <= 100){
    sum = sum + i;
    i++;
}
```

对于 for 循环中语句的一般形式,就是如下的 while 循环形式:

```
表达式 1;
    while(表达式 2){
        语句
        表达式 3;
    }
```

使用 for 语句应该注意:

(1) for 循环中的"表达式 1(循环变量赋初值)"、"表达式 2(循环条件)"和"表达式 3(循环变量增量)"都是选择项,即可以缺省,但分号";"不能缺省。

(2) 省略了"表达式 1(循环变量赋初值)",表示不对循环控制变量赋初值。

(3) 省略了"表达式 2(循环条件)",则不做其他处理时便成为死循环。例如:

```
for( i = 1; ; i++)   sum = sum + i;
```

相当于

```
    i = 1;
while(1){
    sum = sum + i;
    i++;
}
```

(4) 省略了“表达式 3(循环变量增量)”,则不对循环控制变量进行操作,这时可在语句体中加入修改循环控制变量的语句。例如:

```
for( i = 1; i <= 100 ; ){
sum = sum + i;
i++;
}
```

(5) 省略了“表达式 1(循环变量赋初值)”和“表达式 3(循环变量增量)”。例如:

```
for( ; i <= 100 ; ){
sum = sum + i;
i++;
}
```

相当于

```
while(i <= 100){
sum = sum + i;
i++;
}
```

(6) 3 个表达式都可以省略。例如:

```
for( ; ; ) 语句
```

相当于

```
while(1) 语句
```

(7) 表达式 1 可以是设置循环变量的初值的赋值表达式,也可以是其他表达式。例如:

```
for( sum = 0; i <= 100; i++)  sum = sum + i;
```

表达式 1 和表达式 3 可以是一个简单表达式,也可以是逗号表达式。

```
for( sum = 0,i = 1; i <= 100; i++)  sum = sum + i;
```

或

```
for( i = 0,j = 100; i <= 100; i++,j -- )  k = i + j;
```

表达式 2 一般是关系表达式或逻辑表达式,但也可以是数值表达式或字符表达式,只要

其值非零，就执行循环体。例如：

```
for( i = 0; (c = getchar())!= '\n'; i += c );
```

又如：

```
for( ; (c = getchar())! = '\n'; )
printf("%c",c);
```

【实例分析 4-19】 循环嵌套的应用。详细代码如程序 4-19 所示。

程序 4-19　循环嵌套的应用

```
#include <stdio.h>
#include <stdlib.h>
int main(){
int i,j,k;
printf("i j k\n");
for (i = 0; i<2; i++)
    for(j = 0; j<2; j++)
        for(k = 0; k<2; k++)
            printf("%d %d %d\n",i,j,k);
system("pause");
return 0;
}
```

编译成功后，运行程序 4-19，结果如图 4-20 所示。

图 4-20　程序 4-20 运行结果

概括起来，C 语言有三种循环：while 循环、do-while 循环和 for 循环。

三种循环都可以用来处理同一个问题，一般可以互相代替。

while 和 do-while 循环，循环体中应包括使循环趋于结束的语句。

for 语句功能最强，也最常用，可以代替其他循环。

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。

特别提示　for 与 while 语句的区别：for 语句知道执行多少次，while 语句不知道执行多少次，但遇到某个条件会结束。

如果以“吃灌汤包”为例，店铺老板问吃几个包子，大人说吃十个包子，小孩说吃饱就行。那么小孩子吃包子就可以用 while 语句，而大人一开始就知道吃十个包子，可用 for 语句。

4.6 continue 语句和 break 语句

continue 语句和 break 语句都可以用在循环中，用来跳出循环（结束循环）；break 语句还可以用在 switch 语句中，用来跳出 switch 语句。

4.6.1 continue 语句

continue 语句的作用是跳过循环体中剩余的语句而强行执行下一次循环。continue 语句只用在 for、while、do-while 等循环体中，常与 if 条件语句一起使用，用来加速循环。

对比一下 break 和 continue。

break 的用法：

```
while(表达式 1){
    …
    if(表达式 2) break;
    …
}
```

continue 的用法：

```
while(表达式 1){
    …
    if(表达式 2) continue;
    …
}
```

【实例分析 4-20】 continue 语句

大人总共要吃五个包子，吃到第三个包子时，刚把包子汤喝完，小孩不小心把大人的包子碰到地上，大人接着吃第四个包子直到最后把剩余的包子吃完，要用到 continue 语句。详细代码如程序 4-20 所示。

程序 4-20 continue 语句

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i;
    for(i=1;i<=5;i++){
        printf("喝汤\n");
        if(i==3)continue;
        printf("吃包子\n");
        printf("第%d个包子吃完\n",i);
```

```
}
    system("pause");
return 0;
}
```

编译成功后，运行程序 4-20，结果如图 4-21 所示。

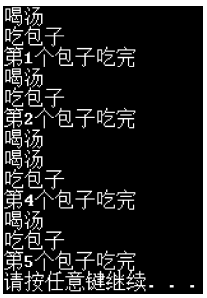


图 4-21　程序 4-20 运行结果

4.6.2　break 语句

break 语句通常用在循环语句和开关语句中。当 break 用于开关语句 switch 中时，可使程序跳出 switch 而执行 switch 以后的语句；如果没有 break 语句，则将成为一个死循环而无法退出。

当 break 语句用于 do-while、for、while 循环语句中时，可使程序终止循环而执行循环后面的语句，通常 break 语句总是与 if 语句联在一起，即满足条件时便跳出循环。

【实例分析 4-21】 break 语句

然而大人吃到第三个包子时，刚把包子汤喝完，小孩不仅把大人的包子打掉，而且把桌子上的所有包子全部打掉在地，剩余的包子就不会再吃了，此时要用到 break 语句。详细代码如程序 4-21 所示。

程序 4-21　break 语句

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i;
    for(i=1;i<=5;i++){
        printf("喝汤\n");
        if(i==3)continue;
        printf("吃包子\n");
        printf("第%d个包子吃完\n",i);
```

```
}
    system("pause");
return 0;
}
```

编译成功后，运行程序 4-21，结果如图 4-22 所示。

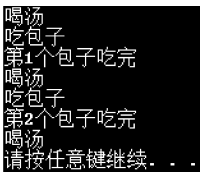


图 4-22　程序 4-21 运行结果

注意：(1) break 语句对 if-else 的条件语句不起作用。
(2) 在多层循环中，一个 break 语句只向外跳一层。

特别提示　C 语言 32 个关键字和 9 种控制语句详见附录 E。

4.7　实例分析

【实例分析 4-22】　用下面的公式求 π：

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots+\frac{1}{4n-3}-\frac{1}{4n-1}$$

分析：系数为正数的项的分母是 $4n-3$（n 为正数项的项数），为负数的项的分母为 $4n-1$（n 为负数项的项数），即分母的变化规律是 1,3,5,7,...的奇数数列，则第 n 项的分母为 $2n-1$，第 10000 项的分母为 2 * 10000－1。详细代码如程序 4-22 所示。

程序 4-22　用下面的公式求 π

```
#include <stdio.h>
#include <math.h>
int main()
{
    double p = 0,j = 1;
    int i;
    for( i = 1;i < 10000;i++)                    //此处 i 为项数
    {
        j = pow( - 1.0,i + 1)/(2 * i - 1);       //pow(x,y)求 x 的 y 次幂
        p += j;
        printf(" % lf\n",4 * p);                 //输出每一项的值
```

```
    }
printf("%lf\n",4*p);                    //输出最终 pi 值
system("pause");
return 0;
}
```

i 的值越大，结果越精准，同样计算时间也越长。

【实例分析 4-23】 判断 m 是否为素数。

采用以下方法分析一个数是否为素数：

(1) 首先要知道什么是素数。只能被自身和 1 整除的正整数是素数。0 不是素数；1 既不是素数，也是不合数；2 是最小的素数，也是唯一一个是偶数的素数。

(2) 具体判断一个正整数是否为素数有以下三种方法。

① 让 m 依次被 2、3、…、m－1 除，如果 m 不能被 2～m－1 中的任何一个整数整除，则 m 是素数。

② 让 m 依次被 2、3、…、m/2 除，如果 m 不能被 2～m/2 中的任何一个整数整除，则 m 是素数。

③ 让 m 依次被 2、3、…、sqrt(m)除，如果 m 不能被 2～sqrt(m)中的任意一个整数整除，则 m 为素数。sqrt(m)为 m 的平方根。

本题采用方法①解决。

让 m 被 2～m－1 除，如果 m 能被 2～m－1 之中任何一个数整除，说明 m 不是素数，则提前结束循环；如果 m 不能被 2～m－1 之中任何一个数整除，则 m 是素数。在循环结束后判断循环变量 i 的值来确定 m 是否为素数。详细代码如程序 4-23 所示。

程序 4-23　判断 m 是否素数

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main(){
  int m,i,k;
  scanf("%d",&m);
  k=m-1;
  for(i=2;i<=k;i++)
  if(i>k)                               /*或 if(i>=k+1) */
  printf("\n%d 是素数\n",m);
else
  printf("\n%d 不是素数\n",m);
  system("pause");
return 0;
}
```

【拓展训练 4-2】 求 100 至 200 间的全部素数。

第 5 章 运算符和表达式

本章主要讲解运算符和表达式的内容。包括以下内容：

（1）算术运算符；

（2）逻辑运算符；

（3）位运算；

（4）表达式。

5.1 算术运算符

C 语言基本算数运算符如表 5-1 所示。

表 5-1 C 语言基本算数运算符

名　　称	符号	说　　明
加法运算符	＋	双目运算符，即应有两个量参与加法运算。如 a＋b，4＋8 等。具有右结合性
减法运算符	－	双目运算符。但“－”也可作负值运算符，此时为单目运算，如－x，－5 等具有左结合性
乘法运算符	*	双目运算符，具有左结合性
除法运算符	/	双目运算符，具有左结合性。参与运算量均为整型时，结果也为整型，舍去小数。如果运算量中有一个是实型，则结果为双精度实型
求余运算符(模运算符)	%	双目运算符，具有左结合性。要求参与运算的量均为整型，不能应用于单精度实型或双精度实型。求余运算的结果等于两数相除后的余数，整除时结果为 0
自增运算	＋＋	其功能是使变量值自增 1
自减运算	－－	其功能是使变量值自减 1

双目运算符，即自增运算(＋＋)和自减(－－)运算，基本内容在 4.1“从＋1 开始”已经讲解。自增和自减运算具有相同的优先级，它们的优先级比运算符 * 、/和％的优先级低，而运算符 * 、/和％的优先级又比单目运算符＋(正号)和－(负号)的优先级低。

【实例分析 5-1】　求余运算符，输入任意一个整数，将其逆序输出，例如输入 1234，输出 4321。详细代码如程序 5-1 所示。

程序 5-1　求余的应用

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  long y,n;
  scanf("%ld",&y);
  while(y!=0)
      {n=y%10;
        printf("%ld",n);
        y=y/10;
      }
  system("pause");
return 0;
}
```

编译成功后，运行程序 5-1，结果如图 5-1 所示。

图 5-1　程序 5-1 运行结果

【实例分析 5-2】

题目描述：

“中国剩余定理”又叫“鬼谷算”，又名“隔墙算”；杨辉叫它“剪管术”，而比较通行的名称是“韩信点兵”。最初记述这类算法的是一本名叫《孙子算经》的书，后来在宋朝经过数学家秦九韶的推广，又发现了一种算法，叫做“大衍求一术”。

今有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二，问物几何？——《孙子算经》

有歌谣流传：

三人同行七十稀，

五树梅花廿一枝，

七子团圆正半月，

除百零五便得知。

以此算法，便可算出人数。

输入：

输入三个数，分别是三三之余，五五之余，七七之余。

输出：

输出总人数(在0～105之间)。

样例输入：

1 2 2

1 1 1

样例输出：

37

1

提示 本题不能用穷举法！

程序 5-2 中国剩余定理

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b,c;
    int i;
    while(scanf("%d%d%d",&a,&b,&c))
    {

    for(i=0;i<=105;i++)
    {
     if(i%3==a&&i%5==b&&i%7==c)printf("%d\n",i);
    }
    }
  system("pause");
return 0;
}
```

【拓展训练 5-1】

题目描述：

相传韩信才智过人，从不直接清点自己军队的人数，只要让士兵先后以三人一排、五人一排、七人一排地变换队形，而他每次只掠一眼队伍的排尾就知道总人数了。输入3个非负整数a,b,c,表示每种队形排尾的人数(a<3,b<5,c<7)，输出总人数的最小值(或报告无解)。已知总人数不小于10，不超过100。

输入：

输入3个非负整数a,b,c,表示每种队形排尾的人数(a<3,b<5,c<7)。

例如：输入2 4 5。

输出：

输出总人数的最小值(或报告无解，即输出 No answer)。

例如：输出 89。

样例输入：

2 1 6

样例输出：

4 1

提示　本题不要用穷举法！

5.2　逻辑运算符

5.2.1　逻辑代数基础

逻辑代数是按照一定的逻辑规则进行逻辑运算的代数，是分析数字电路的数学工具。逻辑代数是分析和设计逻辑电路的数学基础。

1849 年英国数学家乔治·布尔(George Boole)首先提出，用来描述客观事物逻辑关系的数学方法——称为布尔代数。后来被广泛用于开关电路和数字逻辑电路的分析与设计，所以也称为开关代数或逻辑代数，处理二值逻辑问题。

逻辑代数中用字母表示变量——逻辑变量，每个逻辑变量的取值只有两种可能——0 或 1。它们也是逻辑代数中仅有的两个常数。0 和 1 表示两种不同的逻辑状态，不表示数量大小。

逻辑代数的基本运算有三种：逻辑与、逻辑或和逻辑非。

1. 逻辑与

逻辑与函数表达式为

$$Y = A \cdot B = AB = A \quad \text{and} \quad B$$

式中，A 和 B 是输入变量，Y 是输出变量，“·”表示逻辑乘运算。

例如：两个开关串联控制电灯的电路中(见图 5-2)，设开关闭合为“1”、断开为“0”，电灯亮为“1”、不亮为“0”，则明显可以看出：只有当 $A(S_1) = 1$ 并且 $B(S_2) = 1$ 时，才有 $Y=1$；A 和 B 中只要有一个为 0 时，则 $Y=0$。

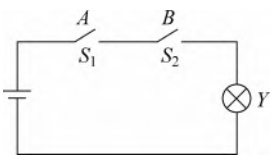


图 5-2　逻辑与开关电路图

将以上运算规则列表，即为逻辑与的逻辑函数真值表，如表 5-2 所示。

表 5-2　逻辑与的逻辑函数真值表

输　入		输　出
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

逻辑与举例：

$0 \cdot 0=0 \cdot 1=1 \cdot 0=0$

$1 \cdot 1=1$

图 5-2 代表的与逻辑关系是：决定事件的全部条件都满足时，事件才会发生。

2．逻辑或

逻辑或函数表达式为

$$Y = A + B = A \text{ or } B$$

式中，A 和 B 是输入变量，Y 是输出变量，“+”表示逻辑或运算。

逻辑或的意义是：A 和 B 中只要有一个或一个以上为“1”时，Y 即为“1”；只有 A 和 B 都为“0”时，Y 才为“0”。

例如：两个开关串联控制电灯的电路中（见图 5-3），设开关闭合为“1”、断开为“0”，电灯亮为“1”、不亮为“0”，则明显可以看出：只要当 $A(S_1)=1$ 或者 $B(S_2)=1$ 或者 A、B 都＝1 时，就有 $Y=1$；只有 A 和 B 都为 0 时，才有 $Y=0$。

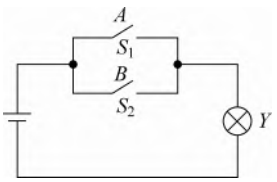


图 5-3　逻辑或开关电路图

将以上运算规则列表，即为逻辑或的逻辑函数真值表，如表 5-3 所示。

表 5-3　逻辑或的逻辑函数真值表

输　入		输　出
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

逻辑或举例：

$0+1=1+0=1+1=1$

$0+0=0$

图 5-3 代表的或逻辑关系是：决定事件的全部条件只要有一个满足时，事件就会发生。

3. 逻辑非

逻辑非函数表达式为：

$$Y = \overline{A}$$

式中，A 是输入变量，Y 是输出变量，“A”上面加一横线（$\overline{A}$）表示对变量 A 进行逻辑非运算。

逻辑非的意义是：A 为“1”时，Y 即为“0”；A 为“0”时，Y 即为“1”；Y 总是与 A 相反。

例如：两个开关串联控制电灯的电路中（见图 5-4），设开关闭合为“1”、断开为“0”，电灯亮为“1”、不亮为“0”，则明显可以看出：当 $A(S)=1$ 时，$Y=0$；当 $A(S)=0$ 时，$Y=1$。

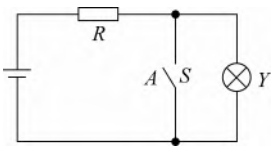


图 5-4　逻辑非开关电路图

将以上运算规则列表，即为逻辑非的逻辑函数真值表，如表 5-4 所示。

表 5-4　逻辑非的逻辑函数真值表

输入 A	输出 Y
0	1
1	0

逻辑非举例：

$$Y = \overline{A} = A' = not\ A$$

图 5-4 代表的逻辑非关系是：决定事件的条件满足时，事件反而不发生。

5.2.2　逻辑运算符

任何类型的量都有逻辑值，逻辑运算得到整型值。运算结果为真时，得到整型值 1；运算结果为假时，得到整型值 0。即逻辑运算结果，非 0 即为 1。

C 语言中提供了三种逻辑运算符，如表 5-5 所示。

表 5-5　逻辑运算符与逻辑关系对照表

运　算　符	逻 辑 关 系	举　　例
&&	逻辑与运算	a>2&&a<3
||	逻辑或运算	s<2||s>6
!	逻辑非运算	! a

其中，! 运算是单目运算。单目运算指的是运算符只在左边或者右边有数字。

1. 运算符优先级的关系

与运算符（&&）和或运算符（||）均为双目运算符，具有左结合性。

非运算符(!)为单目运算符,具有右结合性。

逻辑运算符和其他运算符优先级的关系如图 5-5 所示。

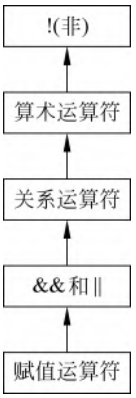


图 5-5　运算符优先级的关系图

“&&”和“||”低于关系运算符,“!”高于算术运算符。

按照运算符的优先顺序可以得出

```
a>b && c>d          等价于 (a>b)&&(c>d)
! b==c||d<a         等价于 ((! b)==c)||(d<a)
a+b>c&&x+y<b        等价于 ((a+b)>c)&&((x+y)<b)
```

2. 逻辑运算的值

逻辑运算的值也为“真”和“假”两种,用“1”和“0 ”来表示。其求值规则如下:

1) 与运算(&&)

参与运算的两个量都为真时,结果才为真,否则为假。例如:

5>0 && 4>2

由于 5>0 为真,4>2 也为真,相与的结果也为真。

2) 或运算(||)

参与运算的两个量只要有一个为真,结果就为真。两个量都为假时,结果为假。例如:

5>0||5>8

由于 5>0 为真,相或的结果也就为真。

3) 非运算(!)

参与运算量为真时,结果为假;参与运算量为假时,结果为真。例如:

!(5>0)

的结果为假。

虽然 C 语言编译在给出逻辑运算值时,以“1”代表“真”,“0 ”代表“假”。但反过来判断一个量是为“真”还是为“假”时,以“0”代表“假”,以非“0”的数值作为“真”。例如:

由于 5 和 3 均为非“0”,因此 5&&3 的值为“真”,即为 1。

5||0 的值为“真”，即为 1。

3. 逻辑表达式

逻辑表达式的一般形式为：

表达式　逻辑运算符　表达式

其中的表达式可以又是逻辑表达式，从而组成了嵌套的情形。例如：

(a&&b)&&c

根据逻辑运算符的左结合性，上式也可写为

a&&b&&c

逻辑表达式的值是式中各种逻辑运算的最后值，以“1”和“0”分别代表“真”和“假”。

【实例分析 5-3】 逻辑运算符举例，详细代码如程序 5-3 所示。

程序 5-3　逻辑运算符举例

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
char c = 'k';
int i = 1,j = 2,k = 3;
    float x = 3e + 5,y = 0.85;
    printf( " %d, %d\n",!x * !y,!!!x );
    printf( " %d, %d\n",x||i&&j - 3,i < j&&x < y );
printf( " %d, %d\n",i == 5&&c&&(j = 8),x + y||i + j + k );
system("pause");
return 0;
}
```

编译成功后，运行程序 5-3，结果如图 5-6 所示。

图 5-6　程序 5-3 运行结果

结果分析：程序 5-3 中！x 和！y 分别为 0，！x * ！y 也为 0，故其输出值为 0。由于 x 为非 0，故!!! x 的逻辑值为 0。对 x|| i && j-3 式，先计算 j-3 的值为非 0，再求 i && j-3 的逻辑值为 1，故 x||i&&j-3 的逻辑值为 1。对 i<j&&x<y 式，由于 i<j 的值为 1，而 x<y 为 0，故表达式的值为 1，0 相与，最后为 0，对 i==5&&c&&(j=8)式，由于 i==5 为假，即值为 0，该表达式由两个与运算组成，所以整个表达式的值为 0。对于式 x+y||i+j+k 由于 x+y 的值为非 0，故整个或表达式的值为 1。

5.3 关系运算符

在程序中经常需要比较两个量的大小关系，以决定程序下一步的工作。比较两个量的运算符称为关系运算符。

在C语言中常见的关系运算符如表5-6所示。

表5-6 常见的关系运算符

运 算 符	运 算 关 系	举 例
>	大于	a>b
>=	大于等于	a>=b
<	小于	1<2
<=	小于等于	c<=d
==	等于	1==c
!=	不等于	1!=2

关系运算符都是双目运算符，其结合性均为左结合。

关系运算符的优先级低于算术运算符，高于赋值运算符。在六个关系运算符中，<、<=、>、>=的优先级相同，高于==和!=，==和!=的优先级相同。

关系表达式的一般形式为：

表达式 关系运算符 表达式

例如：

a+b>c-d

x>3/2

'a'+1<c

-i-5*j==k+1

都是合法的关系表达式。由于表达式也可以又是关系表达式。因此允许出现嵌套的情况。例如：

a>(b>c)

a!=(c==d)

关系运算规则：参加运算的表达式从左到右按关系运算符提供的关系进行比较，满足关系得到整型值1，不满足关系得到整型值0。

```
int  a=1,b=3,c,d;
(a=1)>(b=3);                    /*由于1>3不成立,故其值为假,即为0。*/
c=a>b;                          /*c的值为0*/
d=a+2<=b+3;                     /*d的值为1*/
```

【实例分析 5-4】 关系运算符举例，详细代码如程序 5-4 所示。

程序 5-4　关系运算符举例

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char c = 'k';
    int i = 1,j = 2,k = 3;
    float x = 3e + 5,y = 0.85;
    printf( "%d,%d\n",'a' + 5 < c,- i - 2 * j >= k + 1 );
    printf( "%d,%d\n",1 < j < 5,x - 5.25 <= x + y );
    printf( "%d,%d\n",i + j + k == - 2 * j,k == j == i + 5 );
    system("pause");
return 0;
}
```

编译成功后，运行程序 5-4，结果如图 5-7 所示。

程序 5-4 中求出了各种关系运算符的值。字符变量是以其对应的 ASCII 码参与运算的。对于含多个关系运算符的表达式，例如 k==j==i+5，根据运算符的左结合性，先计算 k==j，该式不成立，其值为 0，再计算 0==i+5，也不成立，故表达式值为 0。

图 5-7　程序 5-4 运行结果

5.4　位运算

位运算是指按二进制进行的运算。在系统软件中，常常需要处理二进制位的问题。C 语言提供了 6 个位操作运算符。这些运算符只能用于整型操作数，即只能用于带符号或无符号的 char，short，int 与 long 类型。

C 语言提供的位运算符，如表 5-7 所示。

表 5-7　位运算符

运算符	含义	描　述
&	按位与	如果两个相应的二进制位都为 1，则该位的结果值为 1，否则为 0
|	按位或	两个相应的二进制位中只要有一个为 1，该位的结果值为 1
^	按位异或	若参加运算的两个二进制位值相同则为 0，否则为 1
~	取反	~是一元运算符，用来对一个二进制数按位取反，即将 0 变 1，将 1 变 0
<<	左移	用来将一个数的各二进制位全部左移 N 位，右补 0
>>	右移	将一个数的各二进制位右移 N 位，移到右端的低位被舍弃，对于无符号数，高位补 0

5.4.1 按位与运算

按位与运算符“&”是双目运算符。其功能是参与运算的两数各对应的二进位相与。只有对应的2个二进位均为1时，结果位才为1，否则为0。参与运算的数以补码方式出现。

例如，9&5可写算式如下：

```
 00001001    9的二进制补码
&00000101    5的二进制补码
--------------------------
 00000001    1的二进制补码
```

可见9&5=1。

按位与运算通常用来对某些位清0或保留某些位。

例如：把a的高八位清0，保留低八位，可作a&255运算(255的二进制数为0000000011111111)。

【实例分析5-5】 按位与运算符举例，详细代码如程序5-5所示。

程序5-5 按位与运算符举例

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int a = 9,b = 5,c;
 c = a&b;
 printf("a = %d\nb = %d\nc = %d\n",a,b,c);
 system("pause");
return 0;
}
```

编译成功后，运行程序5-5，结果如图5-8所示。

图5-8 程序5-5运行结果

5.4.2 按位或运算

按位或运算符“|”是双目运算符。其功能是参与运算的两数各对应的二进位相或。只要对应的2个二进位有一个为1时，结果位就为1。参与运算的两个数均以补码出现。

例如，9|5可写算式如下：

```
 00001001
|00000101
--------------------------
 00001101    十进制为13
```

可见 9|5＝13

【实例分析 5-6】 按位或运算符举例，详细代码如程序 5-6 所示。

程序 5-6　按位或运算符举例

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int a = 9,b = 5,c;
 c = a|b;
 printf("a = %d\nb = %d\nc = %d\n",a,b,c);
 system("pause");
return 0;
}
```

编译成功后，运行程序 5-6，结果如图 5-9 所示。

图 5-9　程序 5-6 运行结果

5.4.3　按位异或运算

按位异或运算符“^”是双目运算符。其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为 1。参与运算数仍以补码出现。

例如，9 ^5 可写成算式如下：

```
  00001001
^ 00000101
----------
  00001100      十进制为 12
```

【实例分析 5-7】 按位异或运算符举例，详细代码如程序 5-7 所示。

程序 5-7　按位异或运算符举例

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int a = 9,b = 5,c;
 c = a^b;
 printf("a = %d\nb = %d\nc = %d\n",a,b,c);
```

```
 system("pause");
return 0;
}
```

编译成功后，运行程序 5-7，结果如图 5-10 所示。

图 5-10　程序 5-7 运行结果

5.4.4　求反运算

求反运算符～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。

例如，～9 的运算为

～(0000000000001001)

结果为

1111111111110110

5.4.5　左移运算

左移运算符“＜＜”是双目运算符。其功能把“＜＜ ”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：

a＜＜2

是指把 a 的各二进位向左移动 2 位。

例如 a＝00000011(十进制的 3)，左移 2 位后为 00001100(十进制 12)，对于无符号数，左移一位相当于原数值乘以 2。

【实例分析 5-8】 左移运算符举例，详细代码如程序 5-8 所示。

程序 5-8　左移运算符举例

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    char a = 'a',b = 'b';
    int p,c,d;
    p = a;
    p = (p<<8)|b;
    d = p&0xff;
    c = (p&0xff00)>>8;
```

```
    printf("a = %d\nb = %d\nc = %d\nd = %d\n",a,b,c,d);
    system("pause");
return 0;
}
```

编译成功后，运行程序 5-8，结果如图 5-11 所示。

图 5-11　程序 5-8 运行结果

5.4.6　右移运算

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，由“>>”右边的数指定移动的位数。

例如，设 a=15

a>>2

表示把 000001111 右移为 00000011(十进制 3)。对于无符号数，右移一位相当于除以 2(取整数部分)。

对于有符号数，在右移时，符号位将随同移动。当为正数时，最高位补 0，而为负数时，符号位为 1，最高位是补 0 或是补 1 取决于编译系统的规定。Turbo C 和很多系统规定为补 1。

【实例分析 5-9】　右移运算符举例，详细代码见程序 5-9 所示。

程序 5-9　右移运算符举例

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    unsigned a,b;
    printf("请输入一个数:   ");
    scanf("%d",&a);
    b = a >> 5;
    b = b&15;
    printf("a = %d\tb = %d\n",a,b);
    system("pause");
return 0;
}
```

编译成功后，运行程序 5-9，结果如图 5-12 所示。

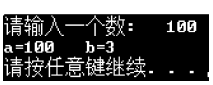


图 5-12　程序 5-9 运行结果

5.5 表达式

用算术运算符和括号将运算对象(也称操作数)连接起来的、符合C语法规则的式子,称为算术表达式。运算对象可以是常量、变量、函数等。

例如:

a * b / c－1.5 ＋ ‘a’

注意:(1) C语言算术表达式的乘号(*)不能省略。

例如:数学式b2－4ac,相应的C表达式应该写成:b*b－4*a*c。

(2) C语言表达式中只能出现字符集允许的字符。

例如:数学 πr^2 相应的C表达式应该写成:PI*r*r(其中PI是已经定义的符号常量)。

(3) C语言算术表达式不允许有分子分母的形式。

例如:(a＋b)/(c＋d)。

(4) C语言算术表达式只使用圆括号改变运算的优先顺序(不要指望用{}[])。可以使用多层圆括号,此时左右括号必须配对,运算时从内层括号开始,由内向外依次计算表达式的值。

5.5.1 (算术)运算符的优先级与结合性

C语言规定了进行表达式求值过程中,各运算符的“优先级”和“结合性”。

(1) C语言规定了运算符的“优先级”和“结合性”。在表达式求值时,先按运算符的“优先级别”高低次序执行。

例如:

a－b*c 等价于 a－(b*c)

“*”运算符优先级高于“－”运算符。

(2) 如果在一个运算对象两侧的运算符的优先级别相同,则按规定的“结合方向”处理。

例如:a－b＋c,到底是(a－b)＋c还是a－(b＋c)?b先与a参与运算还是先于c参与运算?

＋/－运算优先级别相同,结合性为“自左向右”,就是说b先与左边的a结合。所以a－b＋c等价于(a－b)＋c。

(3) 在书写多个运算符的表达式时,应当注意各个运算符的优先级,确保表达式中的运算符能以正确的顺序参与运算。对于复杂表达式为了清晰起见,可以加圆括号“()”强制规定计算顺序。

5.5.2　赋值运算符

赋值运算符：

赋值符号“＝”就是赋值运算符。

赋值表达式：

由赋值运算符组成的表达式称为赋值表达式。

一般形式：

```
<变量><赋值符><表达式>
```

赋值表达式的求解过程：

将赋值运算符右侧的表达式的值赋给左侧的变量，同时整个赋值表达式的值就是刚才所赋的值。

赋值的含义：

将赋值运算符右侧的表达式的值存放到左侧变量名标识的存储单元中。

例如：

```
x = 10 + y;
```

执行赋值运算（操作），将 10＋y 的值赋给变量 x，同时整个表达式的值就是刚才所赋的值。

说明：

（1）赋值运算符左侧必须是变量，右侧可以是常量、变量、函数调用或者常量、变量、函数调用组成的表达式。

例如：x＝10，y＝x＋10，y＝func()都是合法的赋值表达式。

（2）赋值符号“＝”不同于数学的等号，它没有相等的含义（“＝＝”相等）。

例如：C 语言中 x＝x＋1 是合法的（数学上不合法），它的含义是取出变量 x 的值加 1，再存放到变量 x 中。

（3）赋值运算时，当赋值运算符两侧数据类型不同时，将由系统自动进行类型转换。

转换原则：先将赋值号右边表达式类型转换为左边变量的类型，然后赋值。

（4）C 语言的赋值符号“＝”除了表示一个赋值操作外，还是一个运算符，也就是说赋值运算符完成赋值操作后，整个赋值表达式还会产生一个所赋的值，这个值还可以利用。赋值表达式的求解过程是：

① 先计算赋值运算符右侧的“表达式”的值；

② 将赋值运算符右侧“表达式”的值赋给左侧的变量；

③ 整个赋值表达式的值就是被赋值变量的值。

【实例分析 5-10】 分析 x＝y＝z＝3＋5 这个表达式。运算步骤如表 5-8 所示。

表 5-8　表达式运算过程

序　　号	表　达　式	变 量 及 值	表达式的值
1	z=3+5	z(8)	8
2	y=(z=3+5)	y(8)	8
3	x=(y=(z=3+5))	x(8)	8

5.5.3　逗号运算符和逗号表达式

逗号表达式的一般形式：

表达式 1,表达式 2,…,表达式 n

例如：3+5,6+82

逗号表达式的求解过程：

自左向右,求解表达式 1,求解表达式 2,…,求解表达式 n。

整个逗号表达式的值是表达式 n 的值。

例如：逗号表达式 3+5,6+8 的值为 14。

【实例分析 5-11】　逗号表达式举例,详细代码如程序 5-10 所示。

程序 5-10　逗号表达式

```
#include <stdio.h>
int main()
{
    int x,a;
    x = (a = 3,6 * 3);          /* a = 3 x = 18 */
    printf("%d,%d\n",a,x);
    x = a = 3,6 * a;            /* a = 3 x = 3 */
    printf("%d,%d\n",a,x);
    system("pause");
    return 0;
}
```

编译成功后,运行程序 5-10,结果如图 5-13 所示。

图 5-13　程序 5-10 运行结果

逗号表达式主要作用：将若干表达式“串联”起来,表示一个顺序的操作(计算),在许多情况下,使用逗号表达式的目的只是想分别得到各个表达式的值,而并非一定需要得到和使

用整个逗号表达式的值。

5.5.4　运算符优先级总结

C语言允许所有基本类型的值参加同一表达式的运算，也允许所有类型的运算符出现在一个表达式中。因此，表达式值的类型如何确定，运算的先后顺序如何确定，必须通过一套规则解决。

运算优先级原则：

(1) 单目运算的优先级高于双目运算。

(2) 算术运算→关系运算→逻辑运算→赋值运算，优先级从高到低。

为了便于调整优先级，设置括号()为最高优先级。相同优先级存在一个顺序称为结合顺序，结合顺序有从右向左或从左向右。

表达式运算的优先级，如图5-14所示。

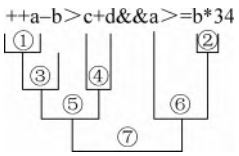


图5-14　表达式运算的优先级

注意：在无法确定优先级时，加()区分，简化表达式。

例如：c=b*=a+2 等价于 c=(b*=(a+2))

常见的运算符如表5-9所示。

表5-9　运算符归类

归　类	运　算　符	结合方向
元素	()　[]　->　.	自左向右
单目运算符	!　~　++　--	自右向左
算数运算符	*　/　%	自左向右
	+　-	自左向右
移位运算符	<<　>>	自左向右
关系运算符	<　<=　>=　>	自左向右
	==　!=	自左向右
位逻辑运算符	&	^	自左向右
逻辑运算符	&&	自左向右
	||	自左向右
条件运算符	?　:	自右向左
赋值运算符	=　+=　-=　*=　/=　%=　&=　|=　<<=　>>=	自右向左

运算符的运算级别，如表 5-10 所示。

表 5-10　运算符的运算级别

级　别	运　算　符	结合顺序
1	() [] → .	从左向右
2	! - ++ -- (type) sizeof * &	从右向左
3	* / %	从左向右
4	+ -	从左向右
5	<< >>	从左向右
6	< <= > >=	从左向右
7	== !=	从左向右
8	&	从左向右
9	^	从左向右
10	|	从左向右
11	&&	从左向右
12	||	从左向右
13	? :	从右向左
14	= op =	从右向左
15	,	从右向左

优先级有以下两种特例。

(1) 自加、自减运算优先级遵循以下原则：

前置：先运算后引用。

后置：先引用后运算。

【实例分析 5-12】 自加、自减运算优先级，详细代码如程序 5-11 所示。

程序 5-11　自加、自减运算优先级举例

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int a = 3,b;
 b = a++ + a++;
 printf("b = %d\n",b);
 b = ++a + (++a);
 printf("b = %d\n",b);
 system("pause");
return 0;
}
```

编译成功后，运行程序 5-11，结果如图 5-15 所示。

图 5-15　程序 5-11 运行结果

(2) 在逻辑运算中,如果逻辑值能确定,则不需要再进行运算。

```
int  a = 0,b = 0++a || b++;  /* b 的值? */
a = 0;a&&++b;                /* b 的值? */
```

编写程序,运行一下看看结果是什么?

5.6　实例分析

【实例分析 5-13】

题目描述:

编写 C 语言程序,实现交换两个变量值的操作。

输入:

输入两个数 a,b。

输出:

输出 b,a。

样例输入:

5 9

样例输出:

9,5

详细代码如程序 5-12 所示。

程序 5-12　交换两个变量的值

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a,b,temp;
    printf("请输入两个数 a,b:\n");
    scanf("%d,%d",&a,&b);
    temp = a;
    a = b;
    b = temp;
    printf("交换后的值为:\n");
    printf("a = %d,b = %d\n",a,b);
    system("pause");
return 0;
}
```

编译成功后，运行程序 5-12，结果如图 5-16 所示。

```
请输入两个数a,b:
5,9
交换后的值为：
a=9,b=5
请按任意键继续. . .
```

图 5-16　程序 5-12 运行结果

程序 5-12 用到了临时变量 temp，如果不使用临时变量 temp，能否实现交换两个变量值的操作呢？下面给出两种方法。

1．加减法

```
a = a + b;
b = a - b;
a = a - b;
```

首先从简单的加减法来进行学习测试。

```
a = a + b;
```

此时 a 为两个数之和，用 sum 来表示，这样就变成了

```
sum = a + b;
```

当 b＝a－b；相当于

```
b = sum - b;
```

即

```
b = a + b - b = a;
```

最后一步

```
a = a - b;
```

也就是 a＝sum-b，这个时候，b 已经变成了 a，也就是

```
a = sum - a = a + b - a = b;
```

【拓展训练 5-2】 请编程并实现上面加减法的举例。

2．异或法

```
a = a^b;
b = b^a;
a = a^b;
```

首先是 a＝a ^ b；

同理，设 sum＝a ^ b；

当 b＝b ^ a；相当于

```
b = b^sum = b^a^b = a;
```

最后一步

```
a = a^b = sum^b = a^b^a = b;
```

当然，要注意是不是所有的数据类型都能这样用，你可以自己试试。

【拓展训练 5-3】 当两个变量分别是 float、double 会出现什么结果？如何实现？

【拓展训练 5-4】 实现交换两个变量值的操作，还有其他方法吗？

【实例分析 5-14】 将十六进制数按二进制输出，详细代码如程序 5-13 所示。

程序 5-13　十六进制数按二进制输出

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i,a;
     printf("请输入一个十六进制数: ");
     scanf("%x",&a);
     printf("其二进制表示为: ");
     for (i=15;i>=0;i--)
          printf("%1d",a&1<<i?1:0);  //注意优先级,先移位<<,结果再与 a 按位与
     printf("\n");
    system("pause");
return 0;
}
```

编译成功后，运行程序 5-13，结果如图 5-17 所示。

```
请输入一个十六进制数: 99
其二进制表示为: 0000000010011001
请按任意键继续. . .
```

图 5-17　程序 5-13 运行结果

【实例分析 5-15】 取一个整数 a 从右端开始的 4～7 位。

分析：可以如下考虑：

(1) 先使 a 右移 4 位。如图 5-18(a)是未右移时的情况，(b)是右移 4 位后的情况。目的是使要取出的那几位移到最右端。

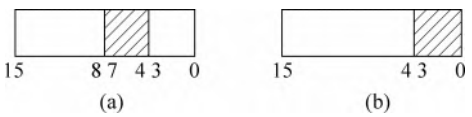


图 5-18　移位示意图

右移到右端可以用下面方法实现 a>>4。

(2) 设置一个低 4 位全为 1，其余全为 0 的数。可用下面方法实现～(～0<<4)。

～0 的全部二进制为全 1，左移 4 位，这样右端低 4 位为 0。过程如下面所示。

0：0000…000000
～0：1111…111111
～0<<4：1111…110000
～(～0<<4)：0000…001111

(3) 将上面二数进行 & 运算。即(a>>4)&～(～0<<4)。

与低 4 位为 1 的数进行 & 运算，就能将这 4 位保留下来。详细代码如程序 5-14 所示。

程序 5-14　取一个整数 a 从右端开始的 4～7 位

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    unsigned a,b,c,d;
    scanf("%d",&a);
    b = a >> 4;
    c = ～(～0 << 4);
    d = b&c;
    printf("输入数 = %d\n 该数右端开始的 4～7 位数 = %d\n",a,d);
    system("pause");
return 0;
}
```

编译成功后，运行程序 5-14，结果如图 5-19 所示。

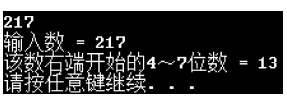


图 5-19　程序 5-14 运行结果

总结：可以任意指定从右面第 m 位开始向左取 n 位。只需将程序中的“b=a>>4”改成“b=a>>(m-n+1)”，以及将“c=～(～0<<4)”改成“c=～(～0<<n)”即可。

引申：位运算应用口诀

清零取反要用与，某位置一可用或

若要取反和交换，轻轻松松用异或

【实例分析 5-16】 xxx 定律

题目描述：

对于一个数 n，如果是偶数，就把 n 砍掉一半；如果是奇数，把 n 变成 3 * n+ 1 后砍掉一半，直到该数变为 1 为止。

请计算需要经过几步才能将 n 变到 1。

输入：

测试包含多个用例，每个用例包含一个整数 n，当 n 为 0 时，表示输入结束(1<=n<=

10000)

输出：

对于每组测试用例请输出一个数，表示需要经过的步数，每组输出占一行。

样例输入：

3

1

0

样例输出：

5

0

xxx 定律-算术运算的详细代码如程序 5-15 所示。

程序 5-15　xxx 定律-算术运算

```
#include <stdio.h>
int main()
{
    int n,s;
    while (scanf("%d",&n),n)
    {
          s = 0;
          while (n!= 1)
          {
                if (n % 2 == 0)   n/ = 2;
                else        n = (3 * n + 1)/2;
                s++;
          }
          printf("%d\n",s);
    }
    system("pause");
    return 0;
}
```

xxx 定律-位运算的详细代码如程序 5-16 所示。

程序 5-16　xxx 定律-位运算

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
int n,time;
```

```
    while (scanf("%d",&n),n) {
        time = 0;
         while (n != 1) {
            if (n & 1) {    //判断是否为奇数
                n = 3 * n + 1;
                n >>= 1;
            } else {
                n >>= 1;  //把 n 砍掉一半
            }
            time++;
        }
        printf("%d\n",time);
    }
    return 0;
}
```

编译成功后，运行程序 5-15、5-16，结果如图 5-20 所示。

图 5-20　程序 5-13，5-14 运行结果

实例分析 5-15 要求记住奇偶判断方法，以及把一个数变成 1/2 的方法（a%2 等价于 a & 1），以后在做题过程中，使用此方法，可以提高做题的效率。

第 6 章 基本输入与输出

通过前面章节的学习，读者应该对 C 语言的基本语法有了感性的认识。本章将通过 OJ (Online Judge)系统基本输入与输出的讲解，使读者真实地使用 OJ 系统来编写代码，并且能够在国内常用的 OJ 系统上完成本书的相关程序，以及读者自己通过 OJ 系统的练习来提高自身编写程序的水平。

本章主要包括以下内容：

(1) OJ 系统简介；

(2) OJ 系统使用说明；

(3) OJ 系统的基本输入类型；

(4) OJ 系统的基本输出类型。

6.1 OJ 系统简介

OJ 系统是一个在线的判题系统。用户可以在线提交多种程序(例如 C、C++及 Java 等语句)源代码，系统对源代码进行编译和执行，并通过预先设计的测试数据来检验程序源代码的正确性。

OJ 系统最初应用于 ACM/ICPC 国际大学生程序设计竞赛和 OI(Olympiad in Intormatics)信息学奥林匹克竞赛中的自动判题和排名。现广泛应用于世界各地高校学生程序设计的训练、参赛队员的训练和选拔、各种程序设计竞赛，以及数据结构和算法的学习和作业的自动提交判断中。

国内比较著名的 OJ 系统有

杭州电子科技大学 OJ，简称杭电 OJ：http://acm.hdu.edu.cn。

北京大学 OJ：http://poj.org。

浙江大学 OJ：http://acm.zju.edu.cn。

6.2 OJ 系统使用说明

6.2.1 OJ 系统注册

(1) 登录 OJ 系统(以杭电 OJ 为例)

OJ 网址：http://acm.hdu.edu.cn

(2) 注册用户如图 6-1 所示。

单击“Register”(注册)按钮,显示注册界面如图 6-2 所示。

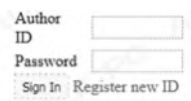


图 6-1 用户注册

图 6-2 注册界面

Author ID 为账号,按自己的情况填写,带 * 号的必须填写。

(3) 注册好 OJ 账号以后,登录自己的账号,如图 6-3 所示。

(4) 选择题目,开始做题,如图 6-4 所示。

图 6-3 登录完成

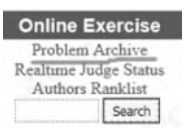
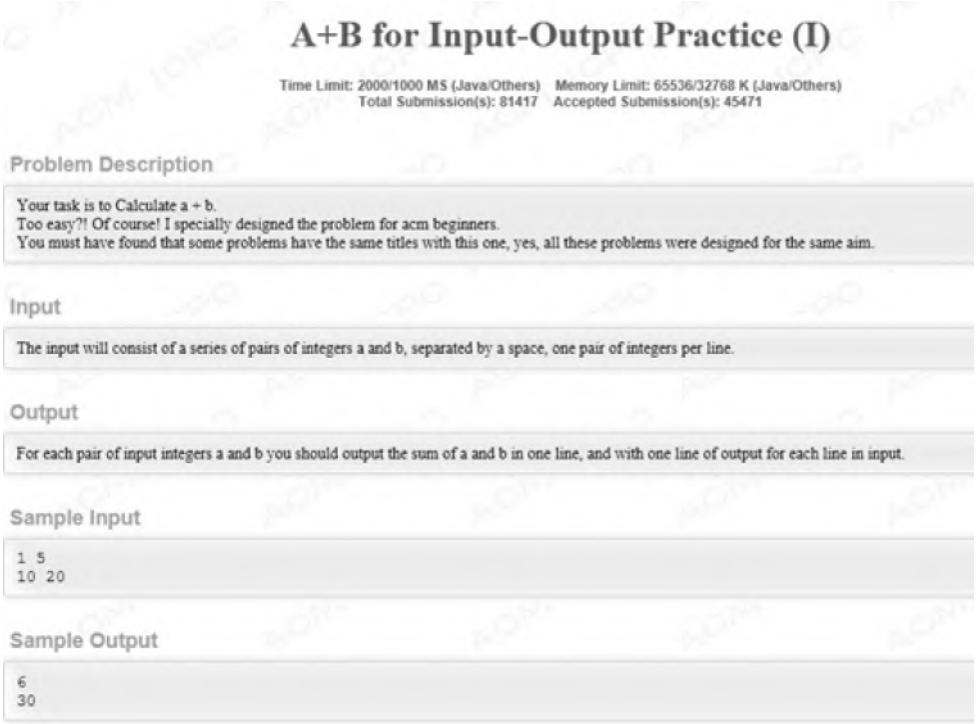


图 6-4 选择题目列表

这里以一道最简单的“1089”为例(http://acm.hdu.edu.cn/showproblem.php? pid=1089),介绍做题方法。如图 6-5 所示。



A+B for Input-Output Practice (I)

Time Limit: 2000/1000 MS (Java/Others)　Memory Limit: 65536/32768 K (Java/Others)
Total Submission(s): 81417　Accepted Submission(s): 45471

Problem Description

Your task is to Calculate a + b.
Too easy?! Of course! I specially designed the problem for acm beginners.
You must have found that some problems have the same titles with this one, yes, all these problems were designed for the same aim.

Input

The input will consist of a series of pairs of integers a and b, separated by a space, one pair of integers per line.

Output

For each pair of input integers a and b you should output the sum of a and b in one line, and with one line of output for each line in input.

Sample Input

```
1 5
10 20
```

Sample Output

```
6
30
```

图 6-5　题目

做题基本方法：首先读完题目,根据题目的意思和给出的样例,设计详细的解题方案,在 Dev C++环境中进行程序的编译、调试和运行。

(5) 提交代码,如果在 Dev C++中编译通过(样例的输入通过了,不一定编写的程序就是对的,在 OJ 系统平台的背后还有大量的数据,必须全部都通过,这需要长期积累经验),单击页面下方的“Submit”按钮(见图 6-6)。

图 6-6　提交代码按钮

出现如图 6-7 所示的提交代码界面。

在“Language”中选择“C”,将在 Dev C++中成功运行的代码粘贴到 OJ 系统的文本框中,然后单击“submit”按钮。

(6) 提交之后便自动跳转到“Status”(状态)页面,会看到刚才提交的题目的评判状态图(Judege Status),如图 6-8 所示。

如果处于 Oueuing 状态,说明后台正在编译,可以按“F5”键刷新,图 6-9 题目编译成功。

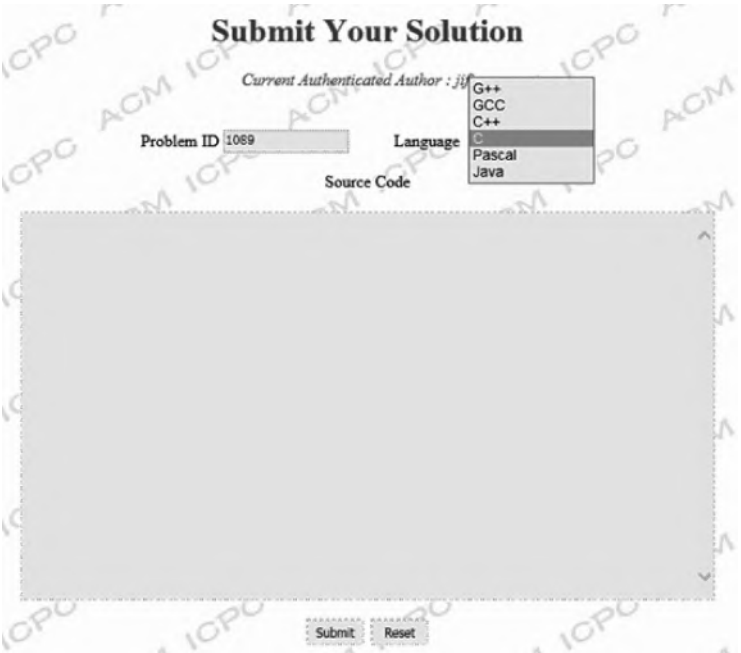


图 6-7　提交代码界面

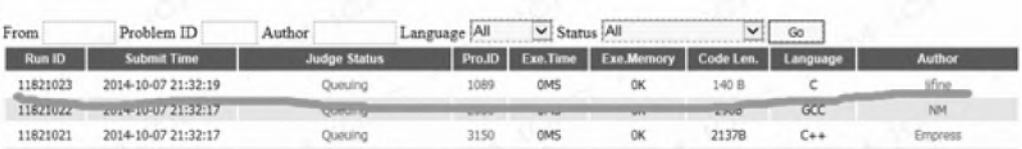


图 6-8　题目状态图

Run ID	Submit Time	Judge Status	Pro.ID	Exe.Time	Exe.Memory	Code Len.	Language	Author
11821032	2014-10-07 21:33:48	Presentation Error	2010	15MS	200K	519B	C	甜甜（#一#）
11821031	2014-10-07 21:33:41	Accepted	1090	0MS	256K	271B	C	武复
11821030	2014-10-07 21:33:27	Accepted	1087	31MS	236K	613B	C++	小白
11821029	2014-10-07 21:33:16	Accepted	1001	0MS	200K	177B	C	xiaoxu
11821028	2014-10-07 21:33:11	Accepted	3153	0MS	2192K	1837B	C++	Empress
11821027	2014-10-07 21:32:59	Accepted	2022	15MS	204K	668B	C	qiu
11821026	2014-10-07 21:32:52	Wrong Answer	4091	15MS	224K	1090B	C++	buyaolian
11821025	2014-10-07 21:32:44	Accepted	2076	15MS	284K	390B	C++	谁的青春不迷茫
11821024	2014-10-07 21:32:35	Wrong Answer	1000	0MS	200K	101B	G++	周铄月
11821023	2014-10-07 21:32:19	Accepted	1089	0MS	200K	140 B	C	jifine
11821022	2014-10-07 21:32:17	[illegible]	3050	0MS	272K	250B	GCC	NM
11821021	2014-10-07 21:32:17	Accepted	3150	0MS	2232K	2137B	C++	Empress
11821020	2014-10-07 21:32:11	Accepted	1166	234MS	1012K	1465B	C++	黄睿
11821019	2014-10-07 21:32:01	Accepted	1091	0MS	200K	131B	C	笑看走完自己的猫
11821018	2014-10-07 21:31:58	Accepted	2076	0MS	284K	391B	C++	谁的青春不迷茫

图 6-9　题目编译成功图

如成功，“Judge Status”栏出现“Accepted”状态，表示提交成功，其他状态均为错误。

6.2.2　常见评判结果

OJ 系统提交后，常见的评判结果如下：

(1) Queuing：提交太多了，OJ 系统无法在第一时间给所有提交以评判结果，后面提交的程序将暂时处于排队状态等待 OJ 系统的评判。这个过程一般不会很长。

(2) Compiling：提交的代码正在编译。

(3) Running：程序正在 OJ 系统上运行。

(4) Judging：OJ 系统正在检查程序的输出是否正确。

(5) Accepted (AC)：程序是正确的，恭喜！

(6) Presentation Error (PE)：虽然程序貌似输出了正确的结果，但是这个结果的格式有点问题。请检查程序的输出是否多了或者少了空格(' ')、制表符('\t')或者换行符('\n')。

(7) Wrong Answer (WA)：输出结果错，这个一般认为是算法有问题。

(8) Runtime Error (RE)：运行时错误，这个一般是程序在运行期间执行了非法的操作造成的。以下列出常见的错误类型：

① ACCESS_VIOLATION 程序想从一些非法的地址空间读取或向其中写入内容。一般指针、数组下标越界都会造成这个错误的。

② ARRAY_BOUNDS_EXCEEDED 程序试图访问一个超出硬件支持范围的数组单元。

③ FLOAT_DENORMAL_OPERAND 进行了一个非正常的浮点操作。一般是由于一个非正常的浮点数参与了浮点操作所引起的，比如这个数的浮点格式不正确。

④ FLOAT_DIVIDE_BY_ZERO 浮点数除法出现除数为零的异常。

⑤ FLOAT_OVERFLOW 浮点溢出。要表示的数太大，超出了浮点数的表示范围。

⑥ FLOAT_UNDERFLOW 浮点下溢。要表示的数太小，超出了浮点数的表示范围。

⑦ INTEGER_DIVIDE_BY_ZERO 在进行整数除法的时候，出现了除数为零的异常。

⑧ INTEGER_OVERFLOW 整数溢出。要表示的数值太大，超出了整数变量的范围。

⑨ STACK_OVERFLOW 栈溢出。一般是由于无限递归或者在函数里使用了太大的数组变量。

⑩ 其他错误，包括 C++标准库/STL 运行时库错误等。

(9) Time Limit Exceeded (TLE)：程序运行的时间已经超出了题目的时间限制。

(10) Memory Limit Exceeded (MLE)：程序运行的内存已经超出了题目的内存限制。

(11) Output Limit Exceeded (OLE)：程序输出内容太多，超过了这个题目的输出限制。

(12) Compilation Error (CE)：程序语法有问题，编译器无法编译。具体的出错信息可以点击链接查看。

(13) System Error (SE)：OJ 内部出现错误。由于 OJ 可能存在一些小问题，所以出现这个信息请原谅，请及时与管理员联系。

(14) Out Of Contest Time：超出比赛时间，这个信息只有在比赛的时候才会出现。

6.2.3 简单题

如何寻找简单题？

一般在试题列表的每道题目后面都有一个比率，这个比率一般为提交成功数/提交次数。比如杭电 OJ 系统中每道题目后面都有一个 Ratio(Accepted/Submissions)，该比率为(提交成功/提交次数)，如图 6-10 所示。这个值越高，说明题目越简单。

Solved	Pro. ID	Problem Title	Ratio(Accepted/Submissions)
√	1000	A + B Problem	31.73%(132529/417629)
	1001	Sum Problem	24.98%(71533/286331)
√	1002	A + B Problem II	19.26%(41885/217431)
	1003	Max Sum	23.37%(34666/148357)
	1004	Let the Balloon Rise	37.51%(28353/75580)
	1005	Number Sequence	24.30%(25975/106898)

图 6-10 题目列表及比率说明图

初学者可选择 Ratio 值高的题目。

6.3 基本输入与输出

Presentation Error (PE)是常见的错误也是最不应该出现的错误，一般是由格式引起，尤其是输入与输出的基本格式。

各种程序设计大赛使用的输入与输出，一般都有多组输入与输出，且输入与输出对于格式的规范性要求很高(便于判题自动测试验证)。

下面介绍常见的基本输入与输出类型，主要涉及格式问题。

【实例分析 6-1】

题目描述：

给定两个整数，求它们之和。

输入：

输入两个数 a，b。

输出：

输出 a＋b 的值。

样例输入：

2 3

样例输出：

5

题目来源：

http://acm.hdu.edu.cn/showproblem.php? pid＝1089

这个题目似曾相识，对了，就是程序 2-1，试试用程序 2-1 的源代码提交到 OJ 系统能得到 AC 吗？

思考一下出现这种情况的原因在哪里？

特别提示　本书讨论常见输入与输出类型为数值型，常见的为 6.3.1 中的第二、第三、第四大类。

6.3.1　基本输入类型

输入类主要以 A＋B 这类问题进行讲解。

1. 第一类

【实例分析 6-2】　程序要想正确运行只需要一组输入数据即可。这种类型也是平时书本上的例子中常见的类型。就是运行一次只能够输入一次数据，如果想更换数据，需重新运行，详细代码如程序 2-1 所示。

题目描述：

给定两个整数 a 和 b，求 a＋b 之和。

输入：

多组由两个整数(a 和 b)构成的输入，a 和 b 之间用空格隔开，每组输入单独占一行。

输出：

每组的两个整数(a 和 b)求和并输出，每组的求和结果独占一行。

样例输入：

1 2

样例输出：

3

代码与程序 2-1 的 A＋B 问题代码一致。

注意：程序 2-1 代码中的 system("pause")，在提交到 OJ 系统的时候一定要去掉。建议将其去掉。因为，system("pause")这一行代码是为了方便人机交互时观察计算结果，如果提交到 OJ 系统中，此条语句多余。但#include<stdlib>可以不去掉，因为在程序里面，可能有语句的头文件在里面包含。

---

2. 第二类

一次运行，要输入多组数据，直到读至输入文件末尾(EOF，End of File)为止。

特别提示　在输入参数前如果有 printf 提示信息，不要打印提示信息。输出完毕后立即终止程序，不要等待用户按键。

【实例分析 6-3】

题目描述：

给定两个整数 a 和 b，求 a＋b 之和。

输入：

多组由两个整数(a 和 b)构成的输入,a 和 b 之间用空格隔开,每组输入单独占一行。

输出:

每组的两个整数(a 和 b)求和并输出,每组的求和结果独占一行。

样例输入:

1 2

100 200

样例输出:

3

300

题目来源:

http://acm.hdu.edu.cn/showproblem.php? pid=1089

详细代码如程序 6-1 所示。

程序 6-1 输入示例-多组输入与输出

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int a,b;
       while(scanf("%d %d",&a,&b) != EOF)
       printf("%d\n",a+b);
       //system("pause");   //如果正式提交到 OJ 系统中,一定要去掉
return 0;
}
```

编译成功后,运行程序 6-1,结果如图 6-11 所示。

图 6-11 程序 6-1 运行结果

程序 6-1 和程序 2-1 的代码区别在于,多加了一条 while 判别语句。

```
while(scanf("%d %d",&a,&b) != EOF)
    {
    …
}
```

从题目的输入可以看出,每次输入两个整数(可以输入无数组测试用例),只要不遇到文件结束符 EOF 或者在键盘上同时按“Ctrl+z”键,就能够输入两个数 a,b,得到其求和的结果。而不像之前在 Dev-C++中运行的程序,当输入两个整数时,结果显示出来,再按一下键盘上的任意键,运行结果的屏幕就消失。

说明:

(1) Scanf()函数返回值就是读出的变量个数,例如 scanf(“%d %d”,&a,&b);。

（2）如果只有一个整数输入，返回值是1；如果有两个整数输入，返回值是2；如果一个都没有，则返回值是－1。

（3）EOF是一个预定义的常量，等于－1，在键盘上同时按“Ctrl＋z”键即可退出运行画面。

3．第三类

一次运行要输入多组数据，组数由第一个输入数据决定（在开始的时候输入一个N，接下来是N组数据）。

【实例分析6-4】

题目描述：

给定两个整数a和b，求a＋b之和。

输入：

先输入需要输入的数据组数，然后输入由所规定组数的两个整数（a和b）构成的输入，a和b之间用空格隔开，每组输入单独占一行。

输出：

每组的两个整数（a和b）求和并输出，每组的求和结果独占一行。

样例输入：

```
2
1 5
10 20
```

样例输出：

```
6
30
```

题目来源：

http://acm.hdu.edu.cn/showproblem.php?pid=1090

详细代码如程序6-2所示。

程序6-2　输入示例-N

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n,i,a,b;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
      scanf("%d %d",&a,&b);
      printf("%d\n",a+b);
    }
    //system("pause");      //如果正式提交到OJ系统中,一定要去掉
return 0;
}
```

编译成功后，运行程序 6-2，结果如图 6-12 所示。

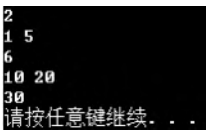


图 6-12　程序 6-2 运行结果

本类型输入解决方案：

```
scanf("%d",&n) ;
    for( i=0 ; i<n ; i++)
{
    …
}
```

因为一开始要求确定有几组输入用例，因此先用 scanf()输入一个整数，然后再用 for 语句进行计算。

4. 第四类

输入不说明有多少组数据，但以某个特殊输入作为结束标志。

【实例分析 6-5】

题目描述：

给定两个整数 a 和 b，求 a+b 之和。

输入：

多组由两个整数(a 和 b)构成的输入，a 和 b 之间用空格隔开，每组输入单独占一行，但遇到 a，b 均为 0 时，自动结束输入，并输出结果。

输出：

每组的两个整数(a 和 b)求和并输出，每组的求和结果独占一行。

样例输入：

1 5

10 20

0 0

样例输出：

6

30

题目来源：

http://acm.hdu.edu.cn/showproblem.php? pid=1091

详细代码如程序 6-3 所示。

程序 6-3　输入示例-特殊标志

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a,b;
    while(scanf("%d %d",&a,&b) &&(a||b))
```

```
    {
        printf("%d\n",a+b);
}
//system("pause");          //如果正式提交到OJ系统中,一定要去掉
return 0;
}
```

编译成功后,运行程序 6-3,结果如图 6-13 所示。

程序 6-3 输入解决方案:

```
while(scanf("%d %d",&a,&b) &&(a||b))
    {
        printf("%d\n",a+b);
}
```

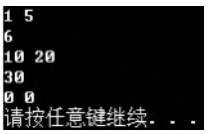


图 6-13　程序 6-3 运行结果

因为当遇到两个整数都是 0 时,输入结束,因此要加上判断输入的两个数是否为 0 的逻辑判断语句 scanf("%d %d",&a,&b) &&(a||b)。

6.3.2　基本输出

在初次接触 ACM 程序设计竞赛时,可能认为样例中都是输入数据和输入数据在一起,输出结果和输出结果在一起,可能会开个数组,把每组的结果存起来,等输入完成再一起输出。但遇到不知有多少组测试数据的题,就难以处理了。甚至有些新手会选择先将答案存储在一个数组里,等程序运行完再输出,其实是没有必要的,机器判决是逐个字符匹配,所以完全可以处理一组输入后,再输出结果。

在 ACM 程序设计竞赛中,输入数据和输出数据是分别在两个不同的文件中进行的,程序的输入和输出是相互独立的,所以读入一组数据就输出一组结果,跟先读入所有数据再输出所有的结果,效果是完全一样的。

因此,每当处理完一组测试数据,就应当按题目要求进行相应的输出操作,而不必将所有结果储存起来一起输出。

在处理输出时,一般要注意每行输出均以回车符结束,包括最后一行。

1. 关于空行(Blank line)

很多题目都要求在输出数据的恰当位置加空行。一个空行就是一个单独的"\n"。这里,有的题目说:"After each test case,you should output one blank line"或者"followed by a blank line.",而有的题目说:"Between each test case,you should ouput one blank line"。要注意 After 和 Between 的区别,因为多了或少了一空行,将导致 Presentation Error 甚至 Wrong Answer。

1) 第一类

输出结果之间没有空行,前面输入的第一类、第二类、第三类的例子均为此种情况。

2）第二类

输出结果之间有空行。

【实例分析 6-6】

题目描述：

给定两个整数 a 和 b，求 a+b 之和。

输入：

多组由两个整数(a 和 b)构成的输入，a 和 b 之间用空格隔开，每组输入单独占一行。

输出：

每组的两个整数(a 和 b)求和并输出，每组的求和结果独占一行，但每组结果之间有一行空行。

样例输入：

1 5

10 20

样例输出：

6

30

题目来源：

http://acm.hdu.edu.cn/showproblem.php? pid=1095

详细代码如程序 6-4 所示。

程序 6-4　输出示例-结果之间有空行

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a,b;
    while(scanf("%d %d",&a,&b) &&(a||b))
    {
        printf("%d\n\n",a+b);
}
//system("pause");          //如果正式提交到 OJ 系统中,一定要去掉
return 0;
}
```

编译成功后，运行程序 6-4，结果如图 6-14 所示。

这种情况最简单，只需要输出结果后，printf("\n")语句后再加一个换行符“\n”。

```
while(scanf("%d %d",&a,&b) &&(a||b))
```

```
    {
        printf("%d\n\n",a+b);
}
```

```
1 5
6

10 20
30
```

图 6-14　程序 6-4 运行结果

2. 关于空格、逗号及其他分隔符

这种情况与空行的情况相似，处理方法也是一样的，只不过把"\n"改成相应的分隔符就行了。

3. 带格式的字符串输出

【实例分析 6-7】 有些题目要求输出这样的字符串 abc＊＊＊＊＊de＊＊＊＊f，其中“＊”代表空格。

要求 str1 在前 5 个字符中左对齐，str2 在第 6 到第 10 个字符中右对齐，str3 在第 11 到第 15 个字符中右对齐。

可行的做法是先初始化一个数组，用′′(空格)填充，再在相应的位置填内容。表述如程序 6-5 所示。

程序 6-5　带格式的字符串输出

```
1.char str[1000];
2.char str1[] = "abc",str2[] = "de",str3[] = "f";
3.memset(str,′′,1000 * sizeof(char));
4.sprintf(str,"%s",str1);
5.str[strlen(str1)] = ′′;
6.sprintf(str + 5,"%5s",str2);
7.str[10] = ′′;
8.sprintf(str + 10,"%5s",str3);
9.str[15] = ′\0′;
10.puts(str);
```

此代码仅为说明，不能运行。

注意：(1) 在调用 sprintf()后，要清除不恰当字符串结束符(第 5,7 行)。

(2) 在恰当的位置添加字符串结束符(第 9 行)。

4. 二维数组的输出

可在第 7 章数组学习后，再掌握此部分内容。

首先要考虑数组是按行排列还是按列排列，如果是按行排列，就应该写成程序 6-6 所示。

程序 6-6　二维数组的输出

```
int i,j;
1: for (i = 0; i < nRow; i++)
2: {
```

```
3:      for (j = 0; j < nCol; j++)
        {
            if (j > 0)
                printf(" ");
            printf("%d",a[j]);
        }
        puts("");
    }
```

此代码仅为说明，不能运行。

如果是按列，就要把 1 行和 3 行交换。

5. 模拟屏幕输出

【实例分析 6-8】 在一些模拟题中，要求输出一幅画，只不过画是由字符组成的。对于这种情况，可以采用和带格式的字符串输出相似的方法，先开一个字符数组（二维数组），然后把数组当成屏幕输出，屏幕的(i,j)点就是数组的(i,j)号元素。最后，输出这个二维数组就行了。

一般来说，输出一个二维字符数组的方法和输出一般数组的方法是一样的，用双重循环来做。不过，可以只用一个循环，原因是在数组每行的恰当位置（一般是末尾）加了一个 '\0'，数组的每一行就成了一个字符串，于是输出程序就变成了画。

程序 6-7　二维字符数组的输出

```
int i;
char str[100][100];
...
for (i = 0; i < nRow; i++)
    puts(str);
```

此代码仅为说明，不能运行。

6.4　解题报告

申请一个技术博客（比如 cnblogs、CSDN 等），将做过的有特点的题目写成解题报告，保存下来，解题报告可参考下面的格式。

解题报告格式

```
/*
//代码要有详细的注释
解题者
```

```
解题时间:
提交时间 1:
失败原因 1:
提交时间 2:
失败原因 2:
提交时间 3:
失败原因 3:
⋮
提交时间 n:
失败原因 n:
题目描述:
编程环境:
解题思路:
难点:
关键点:
个人体会:
备注:
*/
此部分为有详细注释的源代码
```

如果从一开始就坚持将做过的题目或者遇到的问题总结归纳，一段时间后定有所收获。

第 7 章 数　　组

前面章节所讲述的数据类型都属于基本数据类型，用于处理一些数据量比较小的问题。而数组属于构造数据类型的一种，由基本类型按某种规则构造而成，常用于处理数据量较大的问题。

在本章中，将学到数组的基本概念和使用方法，主要包括以下内容：

（1）一维数组；

（2）二维数组；

（3）字符数组；

（4）动态数组；

（5）运算符；

（6）测试程序运行时间方法。

在程序设计中，为了处理方便，把具有相同类型的若干变量按有序的形式组织起来。这些按序排列的同类数据元素的集合称为**数组**。

C 语言中的数组可分为一维、二维及多维，在使用中遵循“先定义，后使用”的原则。

在 C 语言中，数组属于构造数据类型。一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型或是构造类型。

按数组元素的类型不同，数组又可分为数值数组、字符数组、指针数组、结构数组等类别。

本章介绍数值数组和字符数组，其余的内容将在以后章节陆续介绍。

7.1　一维数组

7.1.1　一维数组的定义

在 C 语言中使用数组也必须先进行定义。下面介绍一维数组的定义方式。

格式：

```
数据类型　数组名　[数组大小];
```

例如：

```
int a[10];                    //定义了有 10 个数组元素的 int 型数组 a
float f[20];                  //定义了有 20 个数组元素的 float 型数组 f
char str1[10],str2[20];       //分别定义了有 10 个和 20 个数组元素的 char 型数组 str1 和 str2
```

说明：

(1) 数组的实质是一组变量，其基本组成成员称为数组元素，数组中的每个元素就是一个变量，可完全当作普通的单个变量使用。

(2) 定义数组时，必须指定数组的大小(或称长度，即数组元素的个数)，数组大小必须是大于 0 的整型常量或整型常量表达式，不能是变量或变量表达式，即数据元素的个数必须是定值(属于静态分配)。

(3) 同一个数组一般都包含多个元素，为了加以区分，系统自动给每个元素一个唯一的编号，称作数组元素的下标。C 语言中的下标都是整数，取值范围为 0～(数组元素个数－1)。

(4) 数组定义后，程序运行过程中系统将给其分配一定数量的连续的内存单元，其所占内存单元的个数＝数组的长度×每个数组元素所占内存单元数，例如数组“int a[100]”就占用 100×4＝400 个内存单元，即 400 个字节。

(5) 同一数组中各数组元素的类型均相同，它们占用内存中连续的存储单元，其中第一个数组元素的地址是整个数组所占内存块的开始地址(低地址)，也是数组所占内存块的首地址，最后一个数组元素的地址是整个数组所占内存块的末地址(高地址)。

(6) 地址即是赋给内存单元的从 0 开始的一个序号，用于对各内存单元加以区分，相当于现实中对各类事物所编的顺序号。前面所讲的 scanf()函数中要求在变量名前加取地址运算符“&”，其作用就是获取变量所对应的内存单元的地址。

(7) 不同数组所占内存单元不一定连续，不同数组的数据类型允许不同。

7.1.2 一维数组元素的引用

数组中的每个元素其实就是一个变量，可完全当作普通的单个变量使用，其引用格式如下：

```
数组名[下标];
```

说明：

(1) 下标可以是整型常量、整型变量或整型表达式。C 语言规定，下标的最小值是 0，最大值是数组大小减 1。

(2) 只能逐个引用数组元素，不能一次引用整个数组。

(3) 数组定义以后，数组中的每一个元素其实就是一个变量，可完全当作普通变量去使用。

(4) 数组元素的引用要注意越界问题(下标不要超出其取值范围)，否则会出现意外结果。

【实例分析 7-1】 要求输入某个班数学课的成绩，求出该课程的平均成绩，并将不低于平均成绩的那一部分成绩输出。

分析：要解决此问题，可先输入所有成绩，再求出平均成绩，最后将不低于平均成绩的那一部分成绩输出。

此问题中，要求将成绩输入后进行存放，这就需要很多个变量（一个变量任何时刻只能存放一个数），按前面所讲的知识，需要定义很多个变量，程序中要输入很多变量的标识符（变量名），这显然是不现实的。对于此类需要多个变量的问题，可借助数组来解决。详细代码如程序 7-1 所示。

程序 7-1 求课程的平均成绩

```
#include<stdio.h>
#include<stdlib.h>
#define RENSHU 20                          //人数为 20
int main()
{
    int chengji[RENSHU],i;
    //sum 用于存放总分，初值为 0，average 用于存放平均分
    float sum=0,average;
    printf("\n 请逐个输入全班所有人(%d 个)的数学课成绩：",RENSHU);
    for(i=0;i<RENSHU;i++)                  //输入原始成绩，从 0 号元素开始存放
        scanf("%d",&chengji[i]);
    for(i=0;i<RENSHU;i++)                  //累加求总分
        sum+=chengji[i];                   //等价于 sum=sum+chengji[i];
    average=sum/RENSHU;                    //计算平均成绩
    printf("\n 平均成绩为：%.2f",average);
    printf("\n 不低于平均成绩的那一部分成绩如下：\n");
    for(i=0;i<RENSHU;i++)                  //输出不低于平均分的那部分成绩
        if(chengji[i]>=average)
            printf("%4d",chengji[i]);
    printf("\n");
    //用 system()函数调用 OS 系统命令 pause 实现暂停
    //以方便用户看清结果
system("pause");                           //如果正式提交到 OJ 系统中，一定要去掉
return 0;
}
```

程序 7-1 中用一个符号常量 RENSHU 代表这个班的总人数，如果要将人数改为其他值（例如 50），只需要将初值 20 改为 50 即可。从此程序中可看出，使用符号常量一方面可使要用到的常量值尽可能有意义（起一个有意义的符号常量名），另一方面可以做到一改全改。

7.1.3 一维数组的初始化赋值

和单个变量类似，一维数组也可以在定义的同时初始化（赋初值），其格式如下：

```
数据类型  数组名[数组大小]={表达式 1,…,表达式 n};
```

例如：

```
int a[5] = {1,2,3,4,5};
int b[5] = {1,2,3};
int c[ ] = {1,2,3,4,5,6,7,8,9,10};
```

说明：

（1）“＝”后面的表达式列表一定要用“{}”括起来，被括起来的表达式列表称为初值列表，表达式之间用“,”分隔。

（2）表达式的个数不能超过数组元素的个数。

（3）表达式 1 是 0 号元素的值，表达式 2 是 1 号元素的值，依此类推。

（4）如果表达式的个数小于数组的大小，则未指定值的数组元素被赋值为 0。

（5）当对全部数组元素赋初值时，可以省略数组的大小，此时数组的实际大小就是初值列表中表达式的个数。

7.1.4　实例分析

【实例分析 7-2】 给定一个数组及其元素，求其平均值，详细代码如程序 7-2 所示。

程序 7-2　求数组平均值

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int i,a[10] = {12,23,34,89,90,78,67,56,45,29};
   double ave,sum;
   sum = 0;
   for(i = 0;i<10;i++)
       sum += a[i];
   ave = sum/10;
   printf("此组数的平均值为：%.4f\n",ave);
   printf("\n\n");
 system("pause");
return 0;
}
```

编译成功后，运行程序 7-2，结果如图 7-1 所示。

图 7-1　程序 7-2 运行结果

【实例分析 7-3】 求某天是一年中第几天。

分析：

(1) 首先判断是否为闰年，若为闰年则 2 月为 29 天。

判断闰年的充分必要条件是年份能被 400 整除，或者年份能被 4 整除，但不能被 100 整除。

C 语言表示闰年的方法：

```
(year % 4 == 0&&year % 100!= 0)||(year % 400 == 0)
```

(2) 其次计算之前的几个月一共有多少天。

(3) 最终加上该天在本月中是第几天。

详细代码如程序 7-3 所示。

程序 7-3 某天是一年中第几天

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int days[]={31,28,31,30,31,30,31,31,30,31,30,31};
    int month,day,year;
    printf("输入年月日,年月日格式为 YYYY-MM-DD:\n");          //输入举例 2014-10-10
    scanf("%d-%d-%d",&year,&month,&day);
    if((year % 4 == 0&&year % 100!= 0)||(year % 400 == 0)) days[1]++; //闰年检测
    for(year=0;year<month-1;year++) day+=days[year];
    printf("No.%d\n",day);
    system("pause");
return 0;
}
```

编译成功后，运行程序 7-3，结果如图 7-2 所示。

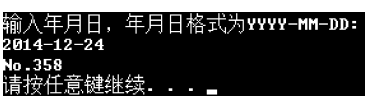


图 7-2 程序 7-3 运行结果

【拓展训练 7-1】

题目描述：

现在使用的日历中，闰年年份被定义为能被 4 或 400 整除但不能被 100 整除。例如 1700，1800，1900 和 2100 不是闰年，而 1600，2000 和 2400 是闰年。给定从公元 2000 年 1 月 1 日(周六)开始逝去的天数，任务是给出这一天是哪年哪月哪日星期几。

输入：

输入包含若干行，每行包含一个正整数，表示从 2000 年 1 月 1 日开始逝去的天数。输入最后一行是－1，不必处理。可以假设结果的年份不会超过 9999。

输出：

对每个测试样例，输出一行，该行包含对应的日期和星期几。

格式为“YYYY-MM-DD DayOfWeek”，其中“DayOfWeek”必须是下面中的一个：“Sunday”，“Monday”，“Tuesday”，“Wednesday”，“Thursday”，“Friday”或“Saturday”。

样例输入：

1730

1740

1750

1751

－1

样例输出：

2004-09-26 Sunday

2004-10-06 Wednesday

2004-10-16 Saturday

2004-10-17 Sunday

题目来源：

http://poj.org/problem?id＝2080

【实例分析 7-4】 输入一组数，按从大到小的顺序降序排序后输出。

分析：排序既是现实中，也是计算机内经常用到的一类操作，具有普遍使用价值。此问题的核心是如何实现多个数的排序。

先来分析两个数的排序，两个数排序可按如图 7-3 所示方式进行。

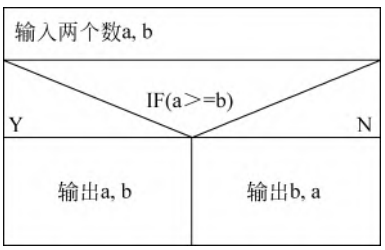


图 7-3 两个数的排序流程图

可看出，两个数有两种情况，需要比较一次，条件是单条件。

同理，对于三个数的排序也可按此思路进行，算法如图 7-4 所示方式进行。

三个数，共计有六种情况，用了五个分支结构加以判断处理，条件是由两个单条件组成的复杂条件。

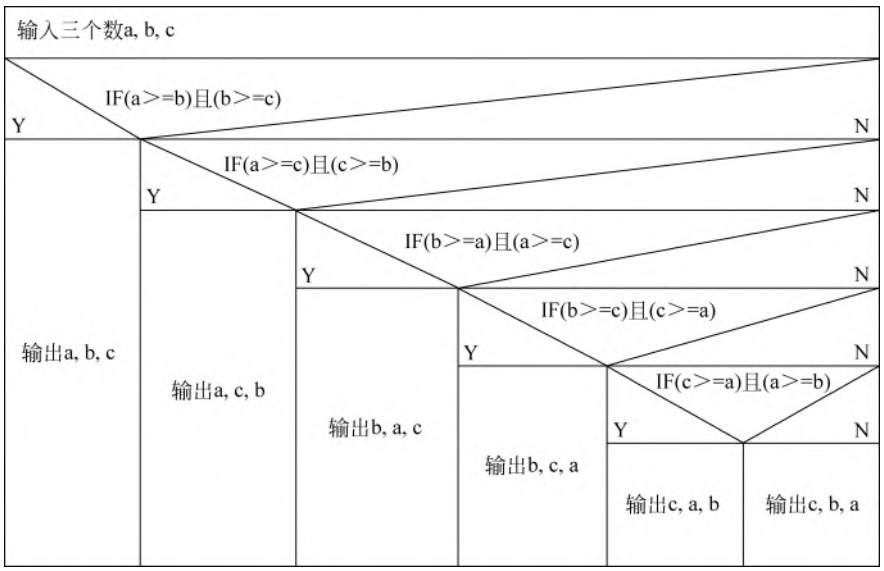


图 7-4　三个数的排序流程图

n 个数的时候，经分析共有 n! 种情况，在 n 比较大的时候，情况非常多。另外，随着数据个数的增加，条件也越来越复杂，n 个数的时候每个条件包含 n-1 个子条件。

很明显，这种排序方式在数据量大的情况下，很难编程实现（情况太多，条件太复杂）。而实际上，只有在数据量大的情况下才有必要去编程序利用计算机实现排序。那么，该如何改进呢？我们换种做法。

仍然先分析两个数的排序，两个数排序可按如图 7-5 所示方式进行。

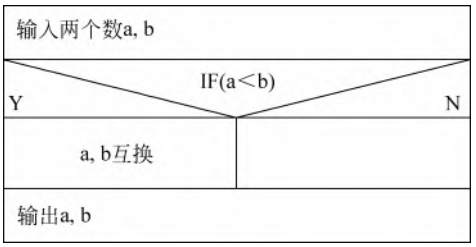


图 7-5　两个数的排序流程图

两个数，比较一次，条件是单条件。

再分析三个数的排序，三个数排序可按如图 7-6 所示方式进行。

经分析发现，a 与 b 经过比较处理后，就可保证 a≥b。而 a 与 c 经过比较处理后，就可保证 a≥c。这样，通过将 a 与 b 和 c 两次比较处理，就可保证 a 是三个数中最大的一个。最后，再将 b 和 c 进行比较处理，使 b≥c，就可得出 a≥b≥c 的结论，按 a，b，c 的次序输出，即为降序。

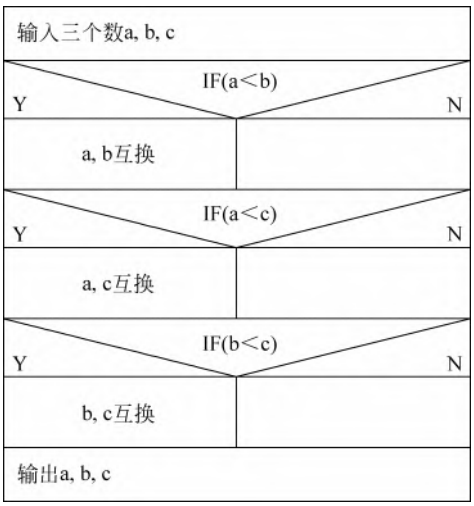


图 7-6 三个数的排序流程图

三个数,比较三次,条件是单条件。

详细代码如程序 7-4 所示。

程序 7-4 三个数的排序

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int a,b,c,t;
   printf("请输入三个整数:\n");
   scanf("%d%d%d",&a,&b,&c);
   if(a<b)
   {
       t=a;
       a=b;
       b=t;
   }
   if(a<c)
   {
       t=a;
       a=c;
       c=t;
   }
   if(b<c)
   {
       t=b;
```

```
        b = c;
        c = t;
    }
    printf(" %6d %6d %6d\n",a,b,c);
 system("pause");
return 0;
}
```

编译成功后，运行程序 7-4，结果如图 7-7 所示。

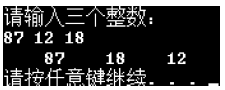


图 7-7 程序 7-4 运行结果

N 个数的时候，经分析可发现共有 N(N－1)/2 种情况，在 N 比较大的时候，情况也比较多，但相对于前一种方法却要少得多。例如在 N＝10 的时候，前一种方法共有 N!＝3628800 种情况，而后一种方法共有 45 种情况。另外，后一种方法始终是单条件。很明显，后一种方法要比前一种方法易于编程实现(情况少，条件简单)。

下面按第二种方法研究多个数情况下的排序算法。

上述第二种方法，在三个数的情况下，其比较是有规律的，即 1 号跟 2、3 号分别比较处理，不符合排序要求时交换内容，这样可找出最大者并放在 1 号位置；然后 2 号再跟 3 号比，找出次大者放在 2 号位置，剩下的就是最小的一个，处于 3 号位置。

将此规律一般化，在 N 个数的情况下应该是这样的：

1 号跟 2,3,…,N 逐个比较处理，就可找出最大者，放到 1 号位置；

2 号跟 3,4,…,N 逐个比较处理，就可找出次大者，放到 2 号位置；

…

N－1 号跟 N 比，找出两者中较大者，放到 N－1 号位置；

剩下的 N 号即为最小的一个，就在 N 号位置。

下面用一个实例加以说明，如表 7-1 所示。

表 7-1 排序比较状态表

趟数	a1	a2	a3	a4	a5	a6	a7	a8	说明
	1	3	5	7	8	6	4	2	初始状态
1	8	1	3	5	7	6	4	2	a1 与 a2～a8 比
2	8	7	1	3	5	6	4	2	a2 与 a3～a8 比
3	8	7	6	1	3	5	4	2	a3 与 a4～a8 比
4	8	7	6	5	1	3	4	2	a4 与 a5～a8 比
5	8	7	6	5	4	1	3	2	a5 与 a6～a8 比
6	8	7	6	5	4	3	1	2	a6 与 a7～a8 比
7	8	7	6	5	4	3	2	1	a7 与 a8 比

N 个数时，共比较 N－1 趟，第一趟 a1 与 a2～aN 相比，共比较 N－1 次；第二趟 a2 与 a3～aN 比，共比较 N－2 次；……；第 j 趟 aj 与 aj＋1～aN 比，共比较 N－j 次；……；最后

一趟(即第 N－1 趟)aN－1 与 aN 比,共比较 1 次,参加比较的两个数若不符合排序要求则进行交换。对于降序,正确的大小顺序应该是后面的数比前面的小,若后面的数比前面的大,就说明不符合排序要求,需要交换。可看出,这种方法最终可实现排序。

对于此排序过程,其基本算法流程如图 7-8 所示方式进行。

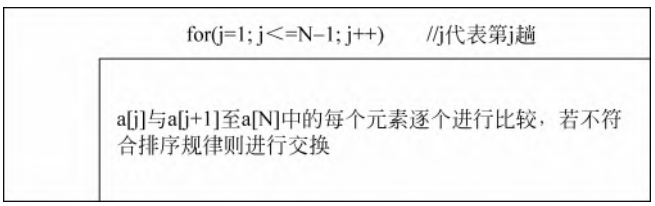


图 7-8　N 个数的排序流程图

算法进一步细化,排序如图 7-9 所示方式进行。

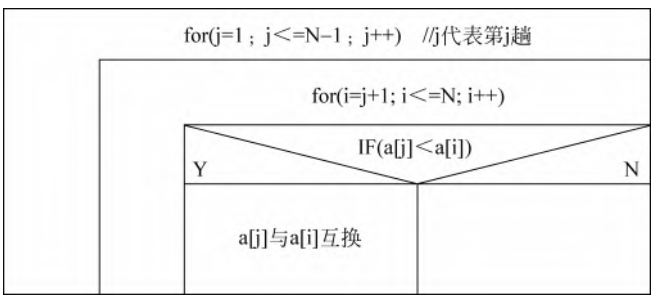


图 7-9　N 个数的排序优化后的流程图

图 7-9 中的算法即为一种典型的交换排序算法,详细代码如程序 7-5 所示。

程序 7-5　交换排序算法

```
#include<stdio.h>
#include<stdlib.h>
#define N 8                          //参加排序的数据总数
int main()
{
      int a[N+1],i,j;
    //定义的数组中包含 N+1 个元素,0 号元素不存放有效数据
    //有效数据从 1 号开始存放,以符合人们日常习惯
    printf("请输入 %d 个数: \n",N);
    for(i=1;i<=N;i++)              //输入原始数据
        scanf("%d",&a[i]);
    printf("\n 排序前: \n");       //按原序输出
    for(i=1;i<=N;i++)
```

```
        printf("%8d",a[i]);
    //下面的二重循环实现排序
    for(j=1;j<=N-1;j++)
        for(i=j+1;i<=N;i++)
            if(a[j]<a[i])
            {
                a[0]=a[j];      //利用空闲的0号元素实现交换
                a[j]=a[i];
                a[i]=a[0];
            }
    printf("\n排序后:\n");      //按排好序的结果输出
    for(i=1;i<=N;i++)
        printf("%8d",a[i]);
    printf("\n");
    system("pause");
return 0;
}
```

编译成功后,运行程序 7-5,结果如图 7-10 所示。

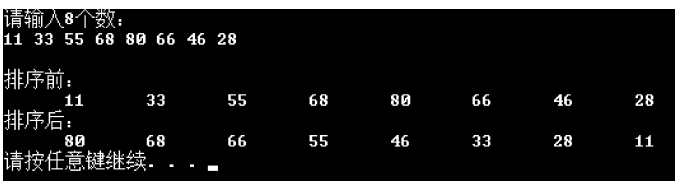


图 7-10　程序 7-5 运行结果

上述程序仅仅是为了演示算法,以 8 个数为例,数据量很小。实际应用中,数据量小的时候完全没必要使用计算机,只有在数据量大的时候,使用计算机进行排序以提高效率,才能发挥出计算机的优势。

开发过程中,有时测试软件的性能需要大批量的数据。在上面的排序程序中,通过键盘输入大批量数据是件非常费时费力的事情。此时,可利用 C 语言的随机函数 rand()产生一组无序数据以供测试之需。

函数 rand()的说明如下:

所在头文件:stdlib.h;

函数格式:int rand();

函数声明:返回 0～RAND_MAX 之间的随机整数值,十六位系统中 RAND_MAX 的最大值是 32767。范围 0～RAND_MAX 内每个数字被选中的机率是相同的。

以 100 个数为例来测试程序 7-5,详细代码如程序 7-6 所示。

程序 7-6 交换排序算法——随机函数生成数据

```
#include<stdio.h>
#include<stdlib.h>
#define N 100                    //参加排序的数据总数
int main()
{
int a[N+1],i,j;
    //定义的数组中包含 N+1 个元素,0 号元素不存放有效数据
    //有效数据从 1 号开始存放,以符合人们日常习惯
    for(i=1;i<=N;i++)            //利用随机函数产生测试用的原始数据
        a[i]=rand();
    printf("\n 排序前:\n");      //按原序输出
    for(i=1;i<=N;i++)
        printf("%8d",a[i]);
    //下面采用交换法循环实现排序
    for(j=1;j<=N-1;j++)
        for(i=j+1;i<=N;i++)
            if(a[j]<a[i])
            {
                a[0]=a[j];       //利用空闲的 0 号元素实现交换
                a[j]=a[i];
                a[i]=a[0];
            }
    printf("\n 排序后:\n");      //按排好序的结果输出
    for(i=1;i<=N;i++)
      printf("%8d",a[i]);
      printf("\n");
    system("pause");
return 0;
}
```

编译成功后,运行程序 7-6,结果如图 7-11 所示。

反复多次运行程序 7-6 就会发现,每次所产生的始终是同一组数。这是什么原因呢?

其实,函数 rand()并不能实现真正的随机,只是一种伪随机,其产生随机数的原理如下:

(1) x0=初始值;

(2) x1=f(x0);

(3) x0=x1。

下次调用时再从(2)开始。

从其产生原理可看出,只要初始值不变,每次产生的一系列数就必然是固定不变的。只有每次改变随机函数的初始值,才能得到不同的随机序列。

```
排序前：
   41   18467    6334   26500   19169   15724   11478   29358   26962   24464
 5705   28145   23281   16827    9961     491    2995   11942    4827    5436
32391   14604    3902     153     292   12382   17421   18716   19718   19895
 5447   21726   14771   11538    1869   19912   25667   26299   17035    9894
28703   23811   31322   30333   17673    4664   15141    7711   28253    6868
25547   27644   32662   32757   20037   12859    8723    9741   27529     778
12316    3035   22190    1842     288   30106    9040    8942   19264   22648
27446   23805   15890    6729   24370   15350   15006   31101   24393    3548
19629   12623   24084   19954   18756   11840    4966    7376   13931   26308
16944   32439   24626   11323    5537   21538   16118    2082   22929   16541

排序后：
32757   32662   32439   32391   31322   31101   30333   30106   29358   28703
28253   28145   27644   27529   27446   26962   26500   26308   26299   25667
25547   24626   24464   24393   24370   24084   23811   23805   23281   22929
22648   22190   21726   21538   20037   19954   19912   19895   19718   19629
19264   19169   18756   18716   18467   17673   17421   17035   16944   16827
16541   16118   15890   15724   15350   15141   15006   14771   14604   13931
12859   12623   12382   12316   11942   11840   11538   11478   11323    9961
 9894    9741    9040    8942    8723    7711    7376    6868    6729    6334
 5705    5537    5447    5436    4966    4827    4664    3902    3548    3035
 2995    2082    1869    1842     778     491     292     288     153      41
```

图 7-11　程序 7-6 运行结果

C 语言中给随机序列赋初值用 srand()函数实现。

如何提供不同的初始值呢？

一种办法是通过键盘输入，但这种办法一方面会添麻烦(程序中会多出一个输入环节)。另一方面如果有两次输入相同，仍会产生相同的随机序列。

C 语言中有一个函数 time(NULL)，其功能是返回自 1970 年 1 月 1 日 0 点到目前为止所经过的秒数，其值是一个整数，因为时间在不断变化，故而这个函数的返回值也始终在变。可将这个返回值作为随机函数序列的初始值，这样就可在不同时刻获得不同的随机序列。

据此，可将程序 7-6 进行修改，详细代码如程序 7-7 所示。

程序 7-7　交换排序算法——随机函数生成数据之 time()

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>                    //有关时间函数的头文件
#define N 100                       //参加排序的数据总数
int main()
{
  int a[N+1],i,j;
    //定义的数组中包含 N+1 个元素,0 号元素不存放有效数据
    //有效数据从 1 号开始存放以符合人们日常习惯
    srand(time(NULL));              //利用秒数值初始化随机序列
    for(i=1;i<=N;i++)               //利用随机函数产生测试用原始数据
        a[i]=rand();
    printf("\n 排序前：\n");        //按原序输出
    for(i=1;i<=N;i++)
        printf("%8d",a[i]);
```

```
    //下面的二重循环实现排序
    for(j=1;j<=N-1;j++)
        for(i=j+1;i<=N;i++)
            if(a[j]<a[i])
            {
                a[0]=a[j];      //利用空闲的0号元素实现交换
                a[j]=a[i];
                a[i]=a[0];
            }
    printf("\n排序后:\n");      //按排好序的结果输出
    for(i=1;i<=N;i++)
        printf("%8d",a[i]);
    printf("\n");
 system("pause");
return 0;
}
```

编译成功后,运行程序时就会发现,每次所产生的数据都不同。

【实例分析 7-5】

给定两个不超过 200 位的正整数 a、b,求 a+b 之和。

注意,所输入的整数 a、b 的位数并不限于 32 位或 64 位以内的整数,但 a、b 的位数不超过 200 位。

解题思路分析:首先要解决的就是存储 200 位整数的问题。显然,任何 C/C++固有类型的变量都无法保存它。最直观的方法是可以用一个字符串来保存它。字符串本质上就是一个字符数组,因此为了编程更方便,我们也可以用数组 unsigned an[200]来保存一个 200 位的整数,让 an[0]存放个位数,an[1]存放十位数,an[2]存放百位数……

那么如何实现两个大整数相加呢?方法很简单,就是模拟小学生列竖式做加法,从个位开始逐位相加,超过或达到 10 则进位。也就是说,用 unsigned an1[201]保存第一个数,用 unsigned an2[200]表示第二个数,然后逐位相加,相加的结果直接存放在 an1 中。要注意处理进位。另外,an1 数组长度定为 201,是因为两个 200 位整数相加,结果可能会有 201 位。

实际编程时,不一定要费心思去把数组大小定得正好合适,稍微开大点也无所谓,以免不小心没有算准这个"正好合适"的数值,而导致数组小了,产生越界错误。

详细代码见程序 7-8 所示:

程序 7-8 A+B-大整数加法

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_LEN 200
```

```
int an1[MAX_LEN + 10];
int an2[MAX_LEN + 10];
char szLine1[MAX_LEN + 10];
char szLine2[MAX_LEN + 10];
int main()
{
    scanf(" % s",szLine1);
    scanf(" % s",szLine2);
    int i,j;
    memset( an1,0,sizeof(an1));
    memset( an2,0,sizeof(an2));
    int nLen1 = strlen( szLine1);
    for(j = 0,i = nLen1 - 1;i >= 0 ; i -- )
        an1[j++] = szLine1[i] - '0';
    int nLen2 = strlen(szLine2);
    for(j = 0,i = nLen2 - 1;i >= 0; i -- )
        an2[j++] = szLine2[i] - '0';
    for(i = 0;i < MAX_LEN; i ++)
    {  an1[i] += an2[i];    //逐位相加
        if(an1[i] >= 10)
        { //看是否要进位
            an1[i] -= 10;
            an1[i + 1] ++;  //进位
        }
    }
    for(i = MAX_LEN; (i >= 0) && (an1[i] == 0); i -- );
    if(i >= 0)
        for(; i >= 0; i -- )
            printf(" % d",an1[i]);
    else
        printf("0");
    printf("\n");
    system("pause");
    return 0;
}
```

编译成功后运行程序 7-8,结果如图 7-12 所示：

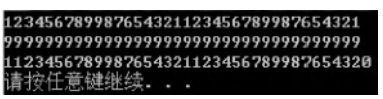


图 7-12　程序 7-8 运行结果

7.2 二维数组

7.2.1 二维数组的定义

当数组中每个元素带有两个下标时，称这样的数组为**二维数组**。在逻辑上可以把二维数组看成是一个具有行和列的表格或矩阵。

下面介绍在C语言中，二维数组的定义格式。

格式：

```
类型名  数组名[常量表达式1][常量表达式2];
```

例如：

```
int a[2][3]
```

二维数组a[2][3]的逻辑结构如表7-2所示。

表7-2 a[2][3]的逻辑结构

	第0列	第1列	第2列
第0行	a[0][0]	a[0][1]	a[0][2]
第1行	a[1][0]	a[1][1]	a[1][2]

定义二维数组应注意以下几点：

(1) 二维数组说明符中必须有用两个方括号括起来的常量表达式，常量表达式的值只能是正整数。可以把“常量表达式1”看成是矩阵的行数，把“常量表达式2”看成是矩阵的列数。

(2) 二维数组的元素在内存中占一系列连续的存储单元。数组元素在内存中的排列顺序是先存放第0行的元素，再存放第1行的元素，以此类推。称这种存放顺序为“按行存放”。

(3) 可以把一个二维数组看成是一个一维数组，每个数组元素又是包含有若干个元素的一维数组。

7.2.2 二维数组元素的引用

二维数组的元素也称为双下标变量，其表示的形式为：

```
数组名[下标][下标]
```

其中下标应为整型常量或整型表达式。例如a[3][4]表示a数组包括三行四列的元素。

下标变量和数组说明在形式中有些相似，但这两者具有完全不同的含义。数组说明的方括号中给出的是某一维的长度，即可取下标的最大值；而数组元素中的下标是该元素在数组中的位置标识。前者只能是常量，后者可以是常量，变量或表达式。

【实例分析7-6】 将一个二维数组行和列的元素互换，存到另一个二维数组中。原理如图7-13所示。

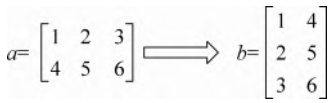


图 7-13　二维数组行和列的元素互换原理图

解题思路：

(1) 定义两个数组，数组 a 为 2 行 3 列，存放指定的 6 个数；数组 b 为 3 行 2 列，开始时未赋值。

(2) 将 a 数组中的元素 a[i][j]存放到 b 数组中的 b[j][i]元素中。

(3) 用嵌套的 for 循环完成(2)。

详细代码如程序 7-9 所示。

程序 7-9　二维数组行和列的元素互换

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int a[2][3]={{1,2,3},{4,5,6}};
 int b[3][2],i,j;
 printf("二维数组转换前:\n");
  for (i=0;i<=1;i++)
 {
  for (j=0;j<=2;j++)
    {
     printf("%5d",a[i][j]);
    }
  printf("\n");
 }
 for (i=0;i<=1;i++)
 {
  for (j=0;j<=2;j++)
    {
     b[j][i]=a[i][j];
    }
}
printf("二维数组转换后:\n");
for (j=0;j<=2;j++)
{
 for (i=0;i<=1;i++)
    {
      printf("%5d",b[j][i]);
    }
```

```
    printf("\n");
   }
   system("pause");
  return 0;
  }
```

编译成功后，运行程序 7-8，结果如图 7-14 所示。

【实例分析 7-7】 有一个 3×4 的矩阵，求出最大的那个元素的值，以及其所在的行号和列号。

解题思路：

（1）先把 a[0][0]的值赋给变量 max。

（2）max 用来存放当前已知的最大值。

（3）a[0][1]与 max 比较，如果 a[0][1]＞max，表示 a[0][1]是已经比过数据中的最大值，把它的值赋给 max，取代 max 的原值。

（4）以后依此处理，最后 max 就是最大的值。

流程图如图 7-15 所示，详细代码如程序 7-10 所示。

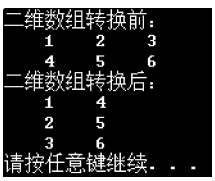


图 7-14　程序 7-9 运行结果

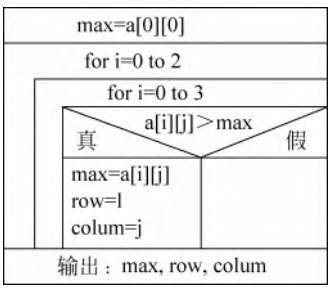


图 7-15　求最大值流程图

程序 7-10　二维数组求最大值

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,j,row=0,colum=0,max;
    int a[3][4]={{1,2,3,4},
                 {6,18,8,7},
                 {5,10,-5,2}};
    for(i=0;i<=2;i++)
     {
      for(j=0;j<=3;j++)
      {
       printf("%5d",a[i][j]);
      }
      printf("\n");
```

```
    }
    max = a[0][0];
    for (i = 0;i <= 2;i++)
        for (j = 0;j <= 3;j++)
            if (a[i][j]> max)
            {   max = a[i][j];
                row = i;
                colum = j;
            }
    printf(" 最大值 = %d\n 所在行 = %d\n 所在列 = %d\n",max,row,colum);
    system("pause");
return 0;
}
```

编译成功后，运行程序 7-10，结果如图 7-16 所示。

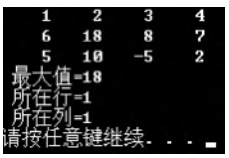


图 7-16　程序 7-10 运行结果

7.2.3　二维数组的初始化赋值

二维数组初始化是在类型说明时给各下标变量赋以初值。二维数组可按行分段赋值，也可按行连续赋值。在{}中给出各数组元素的初值，各初值之间用逗号分开。把{}中的初值依次赋给各数组元素。

有如下几种初始化方式：

(1) 分行进行初始化。

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

在{}内部再用{}把各行分开，第一对{}中的初值 1，2，3 是第 0 行的 3 个元素的初值。第二对{}中的初值 4，5，6 是第 1 行的 3 个元素的初值。相当于执行如下语句：

```
int a[2][3];
a[0][0] = 1; a[0][1] = 2;
a[0][2] = 3; a[1][0] = 4;
a[1][1] = 5; a[1][2] = 6;
```

初始化的数据个数不能超过数组元素的个数，否则出错。

(2) 不分行的初始化。

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

把{}中的数据依次赋给 a 数组中的各元素(按行赋值),即

```
a[0][0] = 1; a[0][1] = 2;
a[0][2] = 3; a[1][0] = 4;
a[1][1] = 5; a[1][2] = 6;
```

(3) 为部分数组元素初始化。

```
int a[3][4] = {{1},{5},{9}};
a[0][0] = 1,a[0][1] = 0,a[0][2] = 0,a[0][3] = 0
a[1][0] = 5,a[1][1] = 0,a[1][2] = 0,a[1][3] = 0
a[2][0] = 9,a[2][1] = 0,a[2][2] = 0,a[2][3] = 0
```

(4) 可以省略第一维的定义,但不能省略第二维的定义。

系统根据初始化的数据个数和第 2 维的长度可以确定第一维的长度。

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

a 数组的第一维的定义被省略,初始化数据共 6 个,第二维的长度为 3,即每行 3 个数,所以 a 数组的第一维是 2。

一般,省略第一维的定义时,第一维的大小按下面规则确定。

初值个数能被第二维整除,所得的商就是第一维的大小;若不能整除,则第一维的大小为商再加 1。

例如:

```
int a[ ][3] = { 1,2,3,4};
```

等价于

```
int a[2][3] = { 1,2,3,4};
```

若分行初始化,也可以省略第一维的定义。下列的数组定义中有两对{},表示 a 数组有两行。

```
int a[ ][3] = {{1,2},{4}};
```

对于二维数组初始化赋值还有以下说明:

数组是一种构造类型的数据。

二维数组可以看作是由一维数组的嵌套而构成的。设一维数组的每个元素都又是一个数组,就组成了二维数组。当然,前提是各元素类型必须相同。根据这样的分析,一个二维数组也可以分解为多个一维数组。C 语言允许这种分解。

例如二维数组 a[3][4],可分解为三个一维数组,其数组名分别为 a[0],a[1],a[2]。

对这三个一维数组不需另作说明即可使用。这三个一维数组都有 4 个元素,例如一维数组 a[0]的元素为 a[0][0],a[0][1],a[0][2],a[0][3]。必须强调的是,a[0],a[1],a[2]不能当作下标变量使用,它们是数组名,不是一个单纯的下标变量。

7.2.4 实例分析

【实例分析 7-8】 打印如图 7-17 格式的乘法口诀表。

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12  4*4=16
1*5=5  2*5=10  3*5=15  4*5=20  5*5=25
1*6=6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

图 7-17 九九乘法表

详细代码如程序 7-11 所示

程序 7-11 九九乘法表

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int i,j;
  for(i=1;i<=9;i++)
    {
        for(j=1;j<=i;j++)    //这里是 j<= i;;
        printf(" %d× %d= %d",j,i,i*j);
        printf("\n");
    }
   system("pause");
return 0;
}
```

编译成功后,运行程序 7-11,结果如图 7-18 所示。

```
1×1=1
1×2=2 2×2=4
1×3=3 2×3=6 3×3=9
1×4=4 2×4=8 3×4=12 4×4=16
1×5=5 2×5=10 3×5=15 4×5=20 5×5=25
1×6=6 2×6=12 3×6=18 4×6=24 5×6=30 6×6=36
1×7=7 2×7=14 3×7=21 4×7=28 5×7=35 6×7=42 7×7=49
1×8=8 2×8=16 3×8=24 4×8=32 5×8=40 6×8=48 7×8=56 8×8=64
1×9=9 2×9=18 3×9=27 4×9=36 5×9=45 6×9=54 7×9=63 8×9=72 9×9=81
请按任意键继续. . .
```

图 7-18 程序 7-11 运行结果

【实例分析 7-9】 打印如样例输出所示的菱形。

题目描述：

从键盘输入一个整数 n(1≤n≤9)，打印出指定的菱形。

输入：

正整数 n(1≤n≤9)。

输出：

第一行前面有 n-1 个空格，第二行有 n-2 个空格，以此类推。

样例输入：

5

样例输出：

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

详细代码如程序 7-12 所示。

程序 7-12 打印菱形

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main(){
int i,j,n;                    //声明 i 为要打印的行数,j 是控制输出打印空格和星号,n 是菱形的高
        printf("请输入菱形的高(奇数):\n");
        scanf("%d",&n);                          //接受输入整数
        for(i=1;i<=n/2+1;i++)
//先打印上半部分,如果奇数输入的是 5,那么上面就会显示 3 行,以此类推
        {
                for(j=1;j<=n-i;j++)      //打印空格
                {
                    printf(" ");
                }
                for(j=1;j<=2*i-1;j++)   //打印星号
                {
                        printf("*");
                }
                printf("\n");
```

```
        }
      for(i=n/2;i>=1;i--)
//n已经明确了,打印下半部分,如果n为5,那么下半部分显示两行,以此类推
      {
            for(j=1;j<=n-i;j++)      //打印空格
            {
                printf(" ");
            }
            for(j=1;j<=2*i-1;j++)    //打印星号
            {
                printf("*");
            }
            printf("\n");
      }
   system("pause");
return 0;
}
```

编译成功后,运行程序 7-12,结果如图 7-19 所示。

图 7-19 程序 7-12 运行结果

【实例分析 7-10】 蛇形填数,在 n×n 的方阵里填入 1,2,...,n×n(n≤8),要求填成蛇形。

例如 n=4 时的方阵为

10	11	12	1
9	16	13	2
8	15	14	3
7	6	5	4

解题思路:

本题较上一题目复杂性有所提升,不再是用一个循环列就可以搞定的题目。二者在方法上有一定区别。二者都是要求打印出符合一定条件的特殊图形。这种类型的题目可以分为两类。

第一类是有一定**对称性的几何图形**,例如打印倒乘法表或者菱形等。这种题目一般思路就是找出图形的特点(对称性等)与循环变量(行号,列号)之间的关系。

可以假设行用 i 表示,列用 j 表示。目的就是找出 i,j 与图形之间的对应关系。按图形形状的不同,复杂性不同。但是都可以看做是在寻找一种或多种“静态关系”。

第二类是**有一定规律性的图形**,比如蛇形填数,走棋盘等。这种题目的一般思路就是找出题目中对图形的限制条件(不能出界,按照一定规则填充等)。

用各种循环和 if 语句将这些"规则"变成程序语句。同样,根据“规则”不同,复杂性也不同。但是都可以看做是在寻找一种或多种“动态关系”。

本题显然是属于第二类。因此寻找的就是“动态关系”。这里的动态关系就是“蛇形”。

通过观察,可以看出规则:在规定的方阵 n×n 里,在不越界及不走重复位置的前提下,填充元素遵循右下左上的规定,即向右走到顶后向下走到顶,再向左走到顶后向上走

到顶。

现在将规则变成程序语句。首先是判断怎样不越界，由于是 n×n 的方阵，使用二维数组最大元素是 a[n－1][n－1]。那么判断是否行越界，是不是用语句 if(y ＜＝ n－1)就行了呢？思路上是没错，不过具体实现时会遇到麻烦，因为当 y ＜＝ n－1 的时候是要不停地进行 y＋＋向右填充的。那么到 y ＝ n－1 的时候仍会向右走一格，y ＝ n 时才会停下来。显然这不是正确的状态，还要处理回退。因此这种判断方法是不完善的。

在这里就要提到很多时候都要用到的判断方法“预判”，即提前一格判断下一格是否越界，如果下一格越界就不再移动。这样就能很好地控制填充的走向，即 y＋1 ＜ n。

在很多情况下，都需要这种预判，不仅对程序的安全性是一种保障，而且避免了一定要走错，才能判断出错误这种尴尬情形。要稍微注意，由于是预判，后面的变量要记住＋1。

详细代码如程序 7-13 所示。

程序 7-13　蛇形填数

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXN 10
int a[MAXN][MAXN];
int main()
{
    int n,x,y,val = 0;
    scanf("%d",&n);
    memset(a,0,sizeof(a));                    //清空数组
    val = a[x = 0][y = n - 1] = 1;            //从右上角开始
    while (val < n * n)                       //圈数,意思是<,如果写成<= 将死循环
    {
        while (x + 1 < n && !a[x + 1][y])   a[++x][y] = ++val;          //右方向下
        while (y - 1 >= 0 && !a[x][y - 1])  a[x][--y] = ++val;          //下方向左
        while (x - 1 >= 0 && !a[x - 1][y])  a[--x][y] = ++val;          //左方向上
        while (y + 1 < n && !a[x][y + 1])   a[x][++y] = ++val;          //上方向右
     }
    for (x = 0; x < n; ++x)
    {
        for (y = 0; y < n; ++y)
        {
            printf("%3d",a[x][y]);
        }
        printf("\n");
    }
  system("pause");
return 0;
}
```

编译成功后，运行程序 7-13，结果如图 7-20 所示。

【实例分析 7-11】 打印水仙花数。水仙花数是指一个三位数，其各位数字立方和等于该数本身。例如 370＝3^3＋7^3＋0^3. 这就说明 370 是一个水仙花数。

其中，“^”表示乘方，5^3 表示 5 的 3 次方，也就是立方。

请用程序找出 100～999 中的水仙花数。

分析：对 100～999 之间的每个数进行处理判断，分别取出这个数的个位数、十位数、百位数；判断这个数是否等于其个位数、十位数、百位数的立方和；如果是，说明这个数是水仙花数，就输出此数，其流程图如图 7-21 所示。

图 7-20　程序 7-13 运行结果

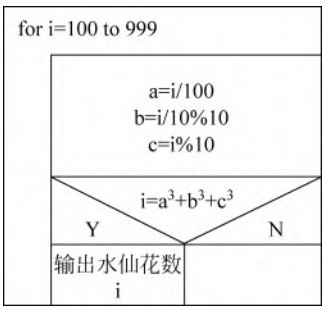


图 7-21　水仙花数程序流程图

详细代码如程序 7-14 所示。

程序 7-14　打印水仙花数-100～999

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main(){
 int i,a,b,c;
 for(i=100;i<=999;i++)
 {
  a=i/100;                                          /*取出百位数*/
  b=i/10%10;                                        /*取出十位数*/
  c=i%10;                                           /*取出个位数*/
  if(i==a*a*a+b*b*b+c*c*c)
  printf("%d    ",i);
 }
 system("pause");
return 0;
}
```

编译成功后，运行程序 7-14，结果如图 7-22 所示。

```
153    370    371    407    请按任意键继续. . .
```

图 7-22 程序 7-14 运行结果

【拓展训练 7-2】 实际上，对 N 的每个取值，可能有多个数字满足条件。

程序的任务是：求 N＝21 时，所有满足条件的花朵数。

注意：这个整数有 21 位，它的各个位数字的 21 次方之和正好等于这个数本身。

如果满足条件的数字不只有一个，请从小到大输出所有符合条件的数字，每个数字占一行。因为这个数字很大，请注意解法时间上的可行性。

要求程序在 3 分钟内运行完毕。

7.3 字符数组

7.3.1 字符数组的定义

用来存放字符量的数组称为**字符数组**。形式与前面介绍的数值数组相同。例如：

```
char c[10];
```

由于字符型和整型通用，也可以定义为 int c[10]，但这时每个数组元素占 2 个字节的内存单元。

字符数组也可以是二维或多维数组。例如：

```
char c[5][10];
```

即为二维字符数组。

7.3.2 字符数组的初始化

字符数组允许在定义时作初始化赋值。例如：

```
char c[10]={'c','','p','r','o','g','r','a','m'};
```

赋值后 c[0]的值为'c'，c[1]的值为' '，c[2]的值为'p'，c[3]的值为'r'，c[4]的值为'0'，c[5]的值为'g'，c[6]的值为'r'，c[7]的值为'a'，c[8]的值为'm'。其中，c[9]未赋值，其值由系统自动赋值为 0。当对全体元素赋初值时也可以省去长度说明。例如：

```
char c[]={'c','','p','r','o','g','r','a','m'};
```

这时 C 数组的长度自动定为 9。

7.3.3 字符数组的引用

字符数组和普通数组一样，也是通过下标引用。

【实例分析 7-12】 输出字符数组中的元素，详细代码如程序 7-15 所示。

程序 7-15 输出字符数组中的元素

```
#include<stdio.h>
#include<stdlib.h>
int main() {
 int i,j;
 char a[][5]={{'B','A','S','I','C',},{'d','B','A','S','E'}};
 for(i=0;i<=1;i++){
  for(j=0;j<=4;j++)
   printf("%c",a[i][j]);
 printf("\n");
 }
 system("pause");
return 0;
}
```

编译成功后，运行程序 7-15，结果如图 7-23 所示。

程序 7-15 中的二维字符数组由于在初始化时全部元素都赋以初值，因此一维下标的长度可以不说明。

图 7-23 程序 7-15 运行结果

7.3.4 字符串和字符串结束标志

在 C 语言中没有专门的字符串变量，通常用一个字符数组存放一个字符串。前面介绍字符串常量时，已说明字符串总是以'\0'作为串的结束符。因此当把一个字符串存入一个数组时，也把结束符'\0'存入数组，并以此作为该字符串是否结束的标志。有了'\0'标志后，就不必再用字符数组的长度判断字符串的长度了。

C 语言允许用字符串的方式对数组作初始化赋值。例如：

```
char c[]={'c',' ','p','r','o','g','r','a','m'};
```

可写为

```
char c[]={"C program"};
```

或去掉{}，写为

```
char c[]="C program";
```

用字符串方式赋值比用字符逐个赋值要多占一个字节，用于存放字符串结束标志'\0'。上面的数组 c 在内存中的实际存放情况为

C		p	r	o	g	r	a	m	\0

'\0'是由 C 编译系统自动加上的。由于采用了'\0'标志,所以在用字符串赋初值时一般无须指定数组的长度,而由系统自行处理。

7.3.5 字符数组的输入与输出

在采用字符串方式后,字符数组的输入与输出将变得简单方便。除了上述用字符串赋初值的办法外,还可用 printf()函数和 scanf()函数一次性输出输入一个字符数组中的字符串,而不必使用循环语句逐个地输入与输出每个字符。

【实例分析 7-13】 使用 printf()输出整个字符数组,详细代码如程序 7-16 所示。

程序 7-16 使用 printf()输出整个字符数组-1

```
#include<stdio.h>
#include<stdlib.h>
int main() {
char c[] = "BASIC\ndBASE";
printf("%s\n",c);
    system("pause");
    return 0;
}
```

编译成功后,运行程序 7-16,结果如图 7-23 所示。

注意在程序 7-16 的 printf()函数中,使用的格式字符串为"%s",表示输出的是一个字符串。而在输出列表中给出的数组名则不能写为 printf("%s",c[])。

【实例分析 7-14】 使用 scanf()从控制台输入一个字符串,然后使用 printf()将其输出,详细代码如程序 7-17 所示。

程序 7-17 使用 printf()输出整个字符数组-2

```
#include<stdio.h>
#include<stdlib.h>
int main() {
 char st[15];
 printf("请输入字符串: ");
 scanf("%s",st);
 printf("您输入的字符串是: %s\n",st);
 system("pause");
return 0;
}
```

编译成功后，运行程序 7-17，结果如图 7-24 所示。

图 7-24　程序 7-17 运行结果

程序 7-17 中由于定义数组长度为 15，因此输入的字符串长度必须小于 15，以留出一个字节用于存放字符串结束标志 '\0'。应该说明的是，对一个字符数组，如果不作初始化赋值，则必须说明数组长度。还应该注意的是，当用 scanf()函数输入字符串时，字符串中不能含有空格，否则将以空格作为串的结束符。

当输入的字符串中含有空格时，运行情况为

```
input string: this is a book
```

输出为

```
your string: this
```

从输出结果可以看出空格以后的字符都未能输出。为了避免这种情况，可多设几个字符数组分段存放含空格的串。可改写如程序 7-18 所示。

【实例分析 7-15】 使用 printf()输出整个字符数组，详细代码如程序 7-18 所示：

程序 7-18　使用 printf()输出整个字符数组-3

```
#include<stdio.h>
#include<stdlib.h>
int main() {
 char st1[6],st2[6],st3[6],st4[6];
 printf("请输入字符串:\n");
 scanf("%s%s%s%s",st1,st2,st3,st4);
 printf("您输入的字符串是: %s %s %s %s\n",st1,st2,st3,st4);
 system("pause");
return 0;
}
```

编译成功后，运行程序 7-18，结果如图 7-25 所示。

图 7-25　程序 7-18 运行结果

程序 7-18 分别设了 4 个数组，输入的一行字符的空格分段分别装入 4 个数组。然后分别输出这 4 个数组中的字符串。在前面介绍过，函数 scanf()的各输入项必须以地址方式出现，例如 &a、&b 等。但在程序 7-17 和 7-18 中却是以数组名方式出现的，这是为什么呢？

这是由于在 C 语言中规定，数组名就代表了该数组的首地址。整个数组是以首地址开头的一块连续的内存单元。例如字符数组 char c[10]，在内存可表示如图 7-26 所示。

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]	C[9]

图 7-26 字符数组 char c[10]在内存的表示

假设数组 c 的首地址为 2000，也就是说 c[0]单元地址为 2000。数组名 c 就代表这个首地址，因此在 c 前面不能再加地址运算符 &。例如写作：

```
scanf("%s",&c);
```

就是错误的。在执行函数 printf("%s",c)时，按数组名 c 找到首地址，然后逐个输出数组中各个字符直到遇到字符串终止标志 '\0' 为止。

7.4 动态数组

7.4.1 为什么引进动态数组

在编写程序过程中，很多地方都用到了数组。但是对于数组的大小却是没有固定的，也就是说可以更改数组大小，其大小是可以变化的。并不像初学时那样，告诉一个范围，就必须取最大值以满足要求。那样就会浪费很多不必要的内存单元，到底应该怎样定义一个动态数组呢？

在前面数组知识讲解过程中，曾介绍过数组的长度是预先定义好的，在整个程序中固定不变。C 语言中不允许动态数组类型。

在定义数组的时候不确定数组的长度。

```
int a[10];                                    //正确
int a[n];                                     //错误
```

例如：

```
int n;
scanf("%d",&n);
int a[n];
```

用变量表示长度，想对数组的大小作动态说明，这是错误的。[ ]里面必须是一个常量，这种数组称为**静态数组**，也就是说静态数组的长度在编译时刻就确定了。

【实例分析 7-16】 数组大小直接赋值，详细代码如程序 7-19 所示。

程序 7-19 数组大小直接赋值

```
#include <stdio.h>
#include <stdlib.h>
int main(){
int n,i;
```

```
int shzu[n];
scanf("%d",&n);
for(i=0;i<10;i++)
        shzu[i]=i;                                          //数组赋值
    for(i=0;i<10;i++)
     printf("%d",i);                                        //输出结果
system("pause");
return 0;
}
```

程序 7-19 在 Dev C++下编译可以通过,但无法运行!其静态数组的大小赋值部分如图 7-27 所示的加粗部分。

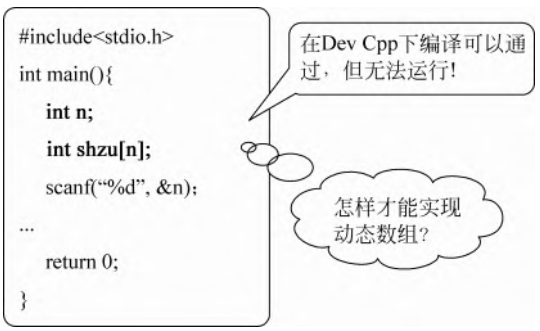


图 7-27 数组大小赋值部分

在实际的编程中,往往会发生这种情况,即所需的内存空间取决于实际输入的数据,但无法预先确定。对于这种问题,用数组的办法很难解决。为了解决上述问题,C 语言提供了一些内存管理函数,这些内存管理函数可以按需要动态地分配内存空间,也可把不再使用的空间回收待用,为有效地利用内存资源提供了方法。其他文献中所提到的"动态数组",指的就是利用内存的申请和释放函数,在程序的运行过程中,根据实际需要指定数组的大小。常用的内存管理函数有以下三个:

1. 分配内存空间函数 malloc()

调用形式:

```
(类型说明符*) malloc (size)
```

功能:在内存的动态存储区中分配一块长度为"size"字节的连续区域。函数的返回值为该区域的首地址。"类型说明符"表示把该区域用于何种数据类型。(类型说明符*)表示把返回值强制转换为该类型指针。"size"是一个无符号数。例如,pc=(char*)malloc(100);表示分配 100 个字节的内存空间,并强制转换为字符数组类型,函数的返回值为指向该字符数组的指针,把该指针赋予指针变量 pc。

2. 分配内存空间函数 calloc()

calloc 也用于分配内存空间。

调用形式：

```
(类型说明符 * )calloc(n,size)
```

功能：在内存动态存储区中分配 n 块长度为“size”字节的连续区域。函数的返回值为该区域的首地址。(类型说明符 *)用于强制类型转换。

calloc()函数与 malloc()函数的区别仅在于一次可以分配 n 块区域。例如 ps=(struet stu *)calloc(2,sizeof (struct stu))；其中的 sizeof(struct stu)是求 stu 的结构长度。因此该语句是按 stu 的长度分配 2 块连续区域，强制转换为 stu 类型，并把其首地址赋予指针变量 ps。

3. 释放内存空间函数 free()

调用形式：

```
free(void * ptr);
```

功能：释放 ptr 所指向的一块内存空间，ptr 是一个任意类型的指针变量，它指向被释放区域的首地址。被释放区应是由 malloc()或 calloc()函数所分配的区域。

关于函数 malloc()/free()的使用如下：

(1) 头文件 #include <malloc. h>。

(2) malloc()函数的参数为所需申请内存的大小，以字节为单位。

(3) malloc()函数返回一个 void * 类型的地址，必须通过强制类型转换，才能赋值给特定的指针变量。

```
int * pint = (int * ) malloc( ... );
```

用 malloc()函数生成各种类型的动态数组，最好使用“sizeof(类型名) * 动态数组长度”形式确定分配内存的大小。

```
int * pint = (int * ) malloc( sizeof(int) * 100 );
```

(4) 分配的内存不再使用时一定要释放。

```
free(pint);
```

7.4.2 动态数组的创建

1. 动态一维数组

一维数组 a[10]，数组的名字 a 是数组 a[0]的地址。动态创建以 a 为名的长度为 n 的一维数组，a 其实就是一个指针。

指针的概念暂且不讲，这里只要知道就行了。

所用到的函数：malloc()，free()。

所用到的头文件＃incluce＜stdlib.h＞。

【实例分析 7-17】 动态一维数组的创建，详细代码如程序 7-20 所示。

程序 7-20 动态一维数组

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
int main(){
int n;
int i;
scanf("%d",&n);
int *shzu = (int *) malloc(sizeof(int) * n);
for(i=0;i<n;i++)
      shzu[i]=i;                                          //数组赋值
    for(i=0;i<n;i++)
     printf("%d",shzu[i]);                                //输出结果
system("pause");
return 0;
}
```

程序 7-19 与 7-20 比较，如图 7-28 所示。在图 7-28 中，左边为代码，右边为动态数组成功创建的主要代码(方框内)部分。

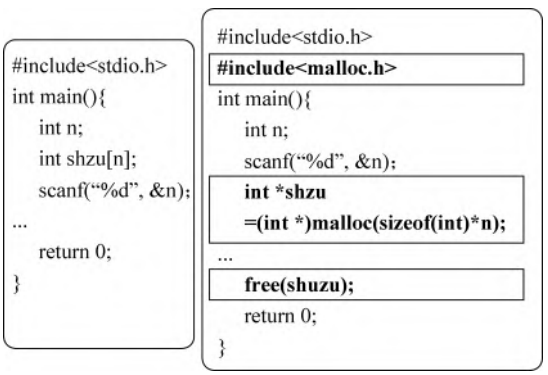


图 7-28 程序 7-19 与 7-20 比较

2. 动态二维数组

动态创建二维数组和动态创建一维数组类似，只不过数组名是二级指针，其主要代码在图 7-29 中以方框标示。

```
#include<stdio.h>
#include<malloc.h>
int main(){
int n;
int **a;
scanf("%d", &n);
a=(int**)malloc(n*sizeof(int*));
for(int i=0; i<n; ++i) a[i]=(int*)malloc(n*sizeof(int))
...
for(int i=0; i<n; ++i)free(a[i]);
    return 0;
}
```

图 7-29 二维动态数组

7.5 测试程序运行时间

怎么测试一段程序的运行时间？C 语言提供了一些函数。

函数 clock()可以用来测试程序运行所花费的时间。

功能：返回处理器调用某个进程或函数所花费的时间。

用法：clock_t clock(void)；

说明：clock_t 其实就是 long，即长整型。该函数返回值是硬件滴答数，要换算成秒或者毫秒，需要除以 CLK_TCK 或者 CLK_TCK CLOCKS_PER_SEC。例如在 Dev-C++中，这两个量的值都是 1000，这表示硬件滴答 1000 下是 1 秒，因此要计算一个进程的时间，用 clock()除以 1000 即可。

一般实现方法如下所示：

```
头文件 time.h
clock_t begin,end;
double  cost;
begin = clock();                                          //开始时间
/* 程序代码 */
end = clock();                                            //结束时间
cost = (double)(end - begin);                             //程序运行时间;
printf("程序运行时间: %lf seconds\n",cost);
int cost =(double)(end - begin);                          //算出来的单位是毫秒
```

【实例分析 7-18】 计算一个 5000 次循环的代码所需要运行的时间。

程序 7-21 计算程序所需要运行的时间

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>
int main()
{
 clock_t begin,end;
 int i;
 double  cost;
 begin = clock();                                          //开始时间
 /* 程序代码 */
 for(i=0;i<10000;i++)
  printf("循环次数:%d\n",i);
  printf("\n\n");
 end = clock();                                            //结束时间
 cost = (double)(end - begin);                             //程序运行时间;
 printf("程序运行时间: %lf seconds\n",cost);
system("pause");
return 0;
}
```

注意：不同的计算机配置运行时间结果可能不一致

7.6 拓展训练

【拓展训练 7-3】

题目描述：

在河南理工大学校园里，如果没有自行车，上课办事会很不方便。但实际上，并非去办任何事情都是骑车快，因为骑车总要找车、开锁、停车、锁车等，这要耽误一些时间。假设找到自行车，开锁并骑上自行车的时间为 27 秒；停车锁车的时间为 23 秒；步行每秒行走 1.2 米，骑车每秒行走 3.0 米，请判断走不同的距离去办事，是骑车快还是走路快。

输入：

第一行为待处理数据的数量 n。

其后每一行整数为一次办事要行走的距离，单位为米。

输出：

对应每个整数，如果骑车快，输出一行"Bike"；如果走路快，输出一行"Walk"；如果一样快，输出一行"All"。

样例输入：

4

50

90

120

180

样例输出：

Walk

Walk

Bike

Bike

【拓展训练 7-4】

题目描述：

一个整型数组里除了两个数字之外，其他的数字都出现了两次。请编写程序找出这两个只出现一次的数字。

输入：

每个测试案例包括两行：第一行包含一个整数 n，表示数组大小。2≤n≤10^6。第二行包含 n 个整数，表示数组元素，元素均为 int。

输出：

对应每个测试案例，输出数组中只出现一次的两个数。输出的数字按从小到大的顺序排列。

样例输入：

82

4 3 6 3 2 5 5

样例输出：

4 6

【拓展训练 7-5】

题目描述：

给出一个长度不超过 1000 的字符串，判断它是不是回文(顺读，逆读均相同)的。

输入：

输入包括一行字符串，其长度不超过 1000。

输出：

可能有多组测试数据，对于每组数据，如果是回文字符串则输出"Yes!"，否则输出"No!"。

样例输入：

hellollehhelloworld

样例输出：

Yes

No

【拓展训练 7-6】

题目描述：

毛毛每天都会熬夜写代码，然后很晚才睡觉，但是每天早晨六点多必须要刷卡出宿舍，

这就导致毛毛必须在某些课上睡一会才能保证充沛的体力，当然某些重要的课是不能睡掉的，而某些课是可以睡的，例如《中国传统文化》，但是睡觉是不能被老师发现的，否则他会以让你重修来威胁你。已知老师会在电子表上显示的时间为回文（例如 15:51）的时候来检查有没有人在睡觉，所以必须要在那个时间之前醒来。现在，给出毛毛开始睡觉的时间，帮她计算出下一个回文时间。

输入：

输入包含多组测试数据。对于每组测试数据，输入只有一行为一个字符串，字符串格式为“HH:MM”，HH 和 MM 都为两位数字（00≤HH≤23，00≤MM≤59）。

输出：

对于每组测试数据，输出只有一行为下一个回文时间。

样例输入：

12:21

23:59

样例输出：

13:31

00:00

第 8 章 自定义函数

函数充分而生动地体现了分而治之和相互协作的理念。它可以将一个大的程序设计任务分解为若干个小的任务，这样便于实现、协作及重用，有效地避免了做什么都要从头开始。同时，大量经过反复测试和实践检验的库函数更能提高程序的开发效率、质量，有效地降低了开发成本。

本章主要包括以下内容：

(1) 函数定义的形式；

(2) 函数参数的形式；

(3) 函数返回值；

(4) 函数调用的形式；

(5) 函数声明。

8.1 为什么要引入函数

日常家庭房屋的建筑过程中，会将一套房子分成厨房、客厅、餐厅、卧室及卫生间等不同的房间(模块)，主要目的是为了承接不同的功能，以便使用起来更方便。程序用于模拟客观世界，在编写程序的过程中，也借鉴模块化的思想进行设计和开发。

8.1.1 模块化程序设计思想

在程序设计过程中，如果程序的功能比较多，规模比较大，为了有效地完成任务，把所有代码都写在 main()函数中，会使主函数变得庞杂和头绪不清，阅读、调试和维护将会变得困难。有时程序中要多次实现同一功能，就需要多次重复编写实现此功能的程序代码，这使程序冗长拖沓。明智的做法是，把所要完成的任务精心分割成若干相对独立，但仍有联系的任务模块，这样的任务模块还可以继续细分成更小的模块。直至那些小模块任务变得相对单纯，对外的数据交换相对简单，容易编写、检测、阅读和维护。

C 源程序是由函数组成的，虽然在前面各章的程序中都只有一个 main()函数，但实际程序一般分为若干个函数模块，由一个主函数和若干个其他函数构成，主函数调用其他函

数，其他函数也可以相互调用。同一个函数可以被一个或多个函数调用多次。因此，C 程序的执行总是从 main()函数开始，完成对其他函数的调用后再返回到 main()函数，最后由 main()函数结束整个程序。一个 C 源程序必须有，且只能有一个主函数 main()。

C 语言中 main()函数是主函数，可以调用其他函数，但不允许被其他函数调用。

程序用于模拟客观世界，函数抽象了现实生活中能相对独立地进行工作的人或组织，函数间的相互协作正好映射了现实生活中人或组织间的相互协作。另外，函数还体现了封装的思想。它有效地将函数内部的具体实现封装起来，对外只提供可见的接口（传入的形式参数与返回的函数值）。这样，调用函数时就不用关心该函数内部具体的实现细节，而只需关注其接口即可调用它来辅助完成所需功能。另外，利用函数还可以大大降低整个程序总的代码量。

特别提示 在 C 语言中，所有的函数定义，包括主函数 main()在内，都是平行的。也就是说，在一个函数的函数体内，不能再定义另一个函数，即不能嵌套定义。但是函数之间允许相互调用，也允许嵌套调用。习惯上把调用者称为主调函数。函数还可以自己调用自己，称为递归调用。

8.1.2 函数分类

从函数定义的角度看，函数可分为库函数和用户定义函数两种。

1. 库函数

由 C 系统提供，用户无须定义，也不必在程序中作类型说明，只需在程序前包含有该函数原型的头文件即可在程序中直接调用。在前面各章中反复用到的 printf()、scanf()等函数均属此类，不同的库函数有不同的头文件。

常见的库函数：

1）输入输出函数

用于完成输入输出功能。

2）字符串函数

用于字符串操作和处理。

3）数学函数

用于数学函数计算。

4）日期和时间函数

用于日期，时间转换操作。

5）字符类型分类函数

用于对字符按 ASCII 码分类：大小写字母，数字，控制字符，分隔符等。

6）转换函数

用于字符或字符串的转换；字符量和各类数字量（整型，实型等）之间的转换；大、小写之间的转换。

7）目录路径函数

用于文件目录和路径操作。

8）其他函数

用于其他各种功能。

以上各类函数不仅数量多，而且有的还需要硬件知识才会使用，因此要想全部掌握需要一个较长的学习过程。应首先掌握一些最基本、最常用的函数，再逐步深入。由于篇幅关系，本书只介绍很少一部分库函数，其余部分读者可根据需要查阅有关手册。

2. 自定义函数

由用户按需要自己写的函数。对于用户自定义函数，不仅要在程序中定义函数本身，而且在主调函数模块中还必须对被调函数进行类型说明，然后才能使用。

本章主要介绍用户自定义的函数。

8.1.3　实例分析

【实例分析 8-1】 C 源程序是由函数组成的，先看一个输出为“我喜欢编程”的程序，详细代码如程序 8-1 所示。

程序 8-1　我喜欢编程

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    printf("我喜欢编程\n");
    system("pause");
return 0;
}
```

在程序 8-1 中，main()函数是一切程序的主函数，函数体里面只有一个输出语句，而这个输出语句也是调用了一个 printf 库函数，并且在程序前包含有该函数原型的头文件(stdio.h)进行了声明(#include<stdio.h>)。

【实例分析 8-2】 一个子函数只实现一个特定的功能，将打印功能由子函数来实现，详细代码如程序 8-2 所示。

程序 8-2　自定义函数——我喜欢编程

```
第 1 行    #include<stdio.h>
第 2 行    #include<stdlib.h>
第 3 行    void print_love(){
第 4 行      printf("我喜欢编程!\n");
```

```
第 5 行          }
第 6 行          int main()
第 7 行          {
第 8 行              print_love ();
第 9 行              system("pause");
第 10 行         return 0;
第 11 行         }
```

在程序 8-2 中，除了 main()函数外还有一个自己定义的函数，函数名是 print_love，整个程序的编译是从上到下。

程序 8-2 的执行顺序是先执行第 6 行的 main()函数，执行到第 8 行调用 print_love()，此时调用进行到第 3 行，执行 3、4、5 行，print_love()函数执行到第 5 行结束后，返回到第 8 行函数调用点，继续往下执行。

8.2 函数定义

8.2.1 函数定义形式

函数定义的一般形式：

```
[类型说明符] 函数名(形式参数表)
{
    函数体;
}
```

说明：

类型说明符：指定函数值的类型。若该项缺省，表示函数值为 int 型。若函数没有返回值，应写作 void。

函数名：即是标识符，用于标识函数，并用其调用函数。

形式参数表：一般来说，计算函数需要多少原始数据，函数的形参表中就有多少个形参，每个形参存放一个数据。

函数体：是一段程序，它实现函数的功能。

自定义函数的常见形式有以下三种：

1. 无参函数的定义形式

```
类型标识符  函数名()
            {
              函数体;
            }
```

例如：print_love()

其中，类型标识符和函数名为函数头。类型标识符指明了函数的类型，函数的类型实际上是函数返回值的类型。该类型标识符符合变量的命名规则。函数名是由用户定义的标识符，函数名后有一个空括号，其中无参数，但括号不可少。{}中的内容称为函数体。在函数体中也有类型说明，这是对函数体内部所用到的变量的类型说明。在很多情况下，都不要求无参函数有返回值，此时函数类型符写为 void。

程序 8-2 的 print_love()是一个无参函数，当被其他函数调用时，输出“我喜欢编程!”字符串。

2．有参函数的定义形式

```
类型标识符　函数名(形式参数表列)
                {
                  函数体;
                }
```

有参函数比无参函数多了两个内容，其一是形式参数表，其二是形式参数类型说明。在形参表中给出的参数称为形式参数，它们可以是各种类型的变量，各参数之间用逗号间隔。在进行函数调用时，主调函数将赋予这些形式参数实际的值。形参既然是变量，当然必须给以类型说明。

【实例分析 8-3】 输入两个整数，计算它们的和并输出运算结果，即将程序 1-3 修改成自定义函数的形式。

程序 8-3　程序 1-3——A＋B 问题

```
#include<stdio.h>
#include<stdlib.h>
int add(int x,int y)
{
    int s;
    s=x+y;
    return s;
}
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",add(a,b));
    system("pause");
return 0;
}
```

3．空函数的定义形式

```
类型标识符　函数名(){ }
```

例如：

```
dummy() {    }
```

特点：调用后什么也不做。

建立程序结构先用空函数占一个位置，以后逐步扩充，在需要时补充功能。主要好处在于可使程序结构清楚，可读性好，以后扩充新功能方便，对程序结构影响不大。

8.2.2 函数参数

函数的参数主要分为形式参数和实际参数。

形式参数：在定义函数时，函数名后面括弧中的变量名，简称形参。

实际参数：在调用函数时，函数名后面括弧中的表达式，简称实参。

也可以理解为在主函数里的是实参，在子函数里的是形参。因为在真正执行的时候是把实参传递给形参来执行的，所以实参才是实际的参数。

以程序 8-3 为例，形参与实参如图 8-1 所示。

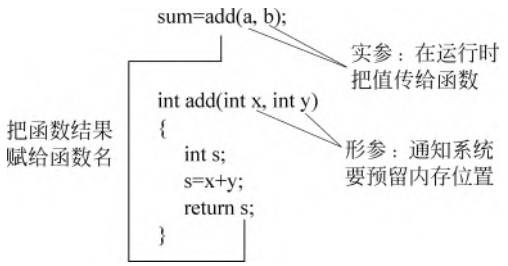


图 8-1 形参与实参说明图

1. 形式参数

形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。实参出现在主调函数中，进入被调函数后，实参变量也不能使用。形参和实参是作数据传送。发生函数调用时，主调函数把实参的值传送给被调函数的形参，从而实现主调函数向被调函数的数据传送。

函数的形参和实参具有以下特点：

(1) 形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效，函数调用结束返回主调函数后，不能再使用该形参变量。

(2) 实参可以是常量、变量、表达式、函数等。无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。因此应用预先赋值，输入等办法使实参获得确定值。

(3) 实参和形参在数量、类型和顺序上应严格一致，否则会发生“类型不匹配”的错误。

(4) 函数调用中发生的数据传送是单向的，即单向传递，只由实参传递给形参，反之不

可。函数调用结束后，只有形参单元被释放，实参单元中的值不变。

【实例分析 8-4】 实参与形参的单向传递，详细代码如程序 8-4 所示。

程序 8-4　实参与形参传递——单向传递-1

```
#include<stdio.h>
#include<stdlib.h>
int s(int n)
{
    int i;
    for(i=n-1;i>=1;i--)
      n=n+i;
    printf("n= %d\n",n);
}
int main()
{
    int n;
    printf("input number\n");
    scanf("%d",&n);
    s(n);
    printf("n= %d\n",n);
    system("pause");
return 0;
}
```

本程序中定义了一个函数 s()，该函数的功能是求∑ni 的值。在主函数中输入 n 值，并作为实参，在调用时传送给 s()函数的形参量 n(注意，本例的形参变量和实参变量的标识符都为 n，但这是两个不同的量，各自的作用域不同)。在主函数中用 printf 语句输出一次 n 值，这个 n 值是实参 n 的值。在函数 s()中也用 printf 语句输出了一次 n 值，这个 n 值是形参，最后取得的 n 值为 0。从运行情况看，输入 n 值为 100，即实参 n 的值为 100。把此值传给函数 s 时，形参 n 的初值也为 100，在执行函数过程中，形参 n 的值变为 5050。返回主函数之后，输出实参 n 的值仍为 100。可见实参的值不随形参的变化而变化。

【实例分析 8-5】 实参与形参传递——单向传递-两数交换，详细代码如程序 8-5 所示。

程序 8-5　实参与形参传递——单向传递-两数交换

```
#include<stdio.h>
#include<stdlib.h>
void swap(int x,int y)
 {
  int temp;
  temp=x;
  x=y;
```

```
    y = temp;
}

int main()
{
   int a = 5,b = 9;
   if(a < b)swap(a,b);
   printf("a = %d\n,b = %d\n",a,b);
   system("pause");
return 0;
}
```

输出结果：a=5,b=9

程序 8-5 最后并不能实现 a,b 值的交换！在开始调用 swap()函数时,a 的值传给 x,b 的值传给 y。执行完 swap()时,x 和 y 的值互换了,但是在 main()函数中 a 和 b 并未互换。也就是说形参值的改变无法传递给实参。

类比解释：

我电脑中有一份加了密码的 word 文档,同事 A 说："嗨,给我一份,参考一下！"

程序 8-4 相当于：我输入密码,打开文档,打印了一份给 A。

我给 A 的是文档本身,A 对给他的那份文档无论怎样修改都不会影响文档原件的内容。

【拓展训练 8-1】 为了使在自定义函数中改变了的变量值能被 main()函数所用,即程序 8-4 中,改变 x,y 的值,a,b 的值也能够改变,如何实现?

2. 函数实参

函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回值作为实参进行传送,因此要求该函数必须有返回值(函数的返回值见 8.2.3)。例如,printf("%d\n",add(a,b));即是把 add 调用的返回值又作为 printf()函数的实参来使用。

在函数调用中还应该注意的一个问题是求值顺序。所谓求值顺序是指对实参表中各量是自左至右使用,还是自右至左使用。对此,各系统的规定不一定相同。在 3.1.1 介绍 printf()函数时已提到过,不再赘述。

8.2.3 函数的返回值

1. 函数返回值

通过函数调用使主调函数得到一个确定的值,称为函数的返回值。

函数的返回值是指函数被调用之后,执行函数体中的程序段所取得的并返回给主调函数的值。如调用程序 8-3 中的 add()函数返回的 s 值等。

函数的返回值语句是 return 语句。

一般形式：

```
return (函数返回值);
```

或者为

```
return 函数返回值;
```

功能：函数体语句执行结束后，使函数保存计算结果并回到程序原来的位置继续计算。

对函数返回值有以下说明：

(1) 函数的值只能通过 return 语句返回主调函数。

return 语句的一般形式为

```
return 表达式;
```

或者为

```
return (表达式);
```

例如：

```
z = x > y? x:y;
 return(z);
```

该语句的功能是计算表达式的值，并返回给主调函数。

(2) 一个函数中可以有多个 return 语句，但是函数一次只能执行其中的一个。当执行到某个 return 语句时，则终止函数执行，并带回函数值。

```
int max(int a,int b)
  {
  if(a > b) return a;
   else return b;
  }
```

(3) 函数值的类型和函数定义中的函数类型应保持一致。如果两者不一致，则以函数类型为准，自动进行类型转换。

(4) 如函数值为整型，在函数定义时可以省去类型说明。

(5) 若函数体内没有 return 语句，就一直将函数执行完，再返回调用函数，带回一个不确定的值。

(6) return 后面可以无“返回值”(即 return;)，则该 return 语句只起到终止函数执行，返回主调函数的作用。

2. 函数值的类型

函数定义时指定函数的类型(即函数值的类型)，应该与 return 语句的类型一致。

说明：

(1) 凡不加类型说明的函数，一律自动按整型处理。

(2) 如果函数类型和 return 语句的类型不一致，以函数类型为准。对数值型数据，可以自动进行类型转换。既函数类型决定返回值的类型。

(3) 如果函数不返回值，可以将函数定义为“无类型”void 或称“空类型”。

不返回函数值的函数，可以明确定义为“空类型”，类型说明符为“void”。详细代码见程序 8-4 中函数 s()，不向主函数返函数值，因此可定义为

```
void s(int n)
{
  …
}
```

一旦函数被定义为空类型，就不能在主调函数中使用被调函数的函数值。例如，在定义 s 为空类型后，在主函数中写语句 sum=s(n);就是错误的。为了使程序有良好的可读性并减少出错，凡不要求返回值的函数都应定义为空类型。函数声明在主调函数中调用某函数之前应对该被调函数进行说明，这与使用变量之前要先进行变量说明是一样的。在主调函数中对被调函数作说明的目的是使编译系统知道被调函数返回值的类型，以便在主调函数中按此种类型对返回值作相应的处理。

8.3 函数调用

8.3.1 函数调用形式

函数调用有以下几种形式：

1. 函数语句

把函数调用作为一个语句。

一般形式：

```
函数名(实际参数表);
```

使用情况：这种方式常用于调用一个可以忽略返回值或没有返回值的函数，只要求函数完成一定的操作。

例如：

```
printstar();
```

2. 函数表达式

函数调用出现在一个表达式中。

一般形式：

```
变量名 = 函数表达式
```

使用情况：这种表达式称为函数表达式。这时要求函数带回一个确定的值以参加表达式的运算。

例如：

```
c = 3 + max(a,b);
```

说明：

(1) 首先被调用函数必须是已存在的函数，例如用户自定义函数或库函数。

(2) 如果使用库函数，需要在文件的开头用#include命令将需要的库函数的头文件包含到文件中。

3. 函数参数

函数调用作为另一函数调用时的实参，例如：

```
m = max(a,max(b,c));
```

其中，max(b,c)是一次函数调用，它的值作为max另一次调用的实参。

8.3.2　函数声明

函数声明是指函数的定义原则上必须在函数调用前完成，例如print_love()。

对被调函数的声明也有两种格式：

```
类型说明符　被调函数名(类型　形参,类型　形参…);
```

或者为

```
类型说明符　被调函数名(类型,类型…);
```

即将函数定义中的函数体花括号{}全部去掉，加上分号；。

在C程序中，一个函数的定义可以放在任意位置，既可放在主函数main()之前，也可放在main()之后。

当函数调用在前，实现(即给出具体的函数定义)在后时，需要使用函数声明，以告知编译器当前使用的函数存在，编译器才会在调用该函数时进行对照检查(例如函数名是否正确，实参与形参的类型和个数是否一致)。如果调用之前没有给出函数实现，也没有函数声明将无法通过编译。

在一个函数中调用另一个函数需要具备如下条件：

(1) 被调用函数必须是已经定义的函数(是库函数或用户自己定义的函数)。

(2) 如果使用库函数，应该在本文件开头加相应的#include指令。

(3) 如果使用自己定义的函数，而该函数的位置在调用它的函数后面，应该进行声明。

特别提醒：函数必须在main函数前完成定义，如果不是，就必须进行函数的声明。

【实例分析 8-6】　函数声明举例，详细代码如程序8-6所示。

程序 8-6　函数声明举例——对比

```
/* 函数定义在主函数之前,不需要声明 */
#include <stdio.h>
#include <stdlib.h>
```

```
/* 函数定义在主函数之后,需要进行函数声明 */

#include <stdio.h>
#include <stdlib.h>
```

```
void print_love(){
  printf("Hello World!");
}

int main()
{
    print_love ();
    system("pause");
return 0;
}
```

```
int main()
{
void print_love();  /* 当子函数放置在主函数后面时,须声明调用的子函数 */
print_love ();      //函数调用
    system("pause");
return 0;
}

void print_love(){
  printf("Hello World!");
}
```

解题思路：如果函数返回值的类型与指定的函数类型不同，按照赋值规则处理。

【实例分析 8-7】 用 add()自定义函数在主函数 main()之前实现两个数的加法。

解题思路：首先要定义 add()函数，它为 float 型，应有两个参数，也为 float 型。特别要注意的是对 add()函数进行声明。

分别编写 add()函数和 main()函数，它们组成一个源程序文件。

main()函数的位置在 add()函数之前，在 main()函数中对 add()函数进行声明。详细代码如程序 8-7 所示。

程序 8-7 求和——main()函数的位置在 add()函数之前，在 main()函数中对 add()函数进行声明

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    float add(float x,float y);     /* 当子函数放置在主函数后面时,须声明调用的子函数 */
    float a,b,c;
    printf("Please enter a and b:");
    scanf("%f,%f",&a,&b);
    c=add(a,b);                     //调用 add()函数
    printf("sum is %f\n",c);
    system("pause");
return 0;
}
float add(float x,float y)          //定义 add()函数
{
    float z;
    z=x+y;
return(z);
}
```

函数原型的一般形式有两种，例如：

```
float add(float x,float y);
float add(float,float);
```

原型说明可以放在文件的开头，这时所有函数都可以使用此函数。

C 语言中规定在以下几种情况时，可以省去主调函数中对被调函数的函数声明。

(1) 如果被调函数的返回值是整型或字符型时，可以不对被调函数作说明，而直接调用。这时系统将自动对被调函数返回值按整型处理。程序 8-3 的主函数中未对函数 s()作说明而直接调用即属此种情形。

(2) 当被调函数的函数定义出现在主调函数之前时，在主调函数中也可以不对被调函数再作说明而直接调用。例如程序 8-3 中，函数 add()的定义放在 main()函数之前，因此可在 main()函数中省去对 add()函数的函数声明 int add(int a,int b)。

(3) 如在所有函数定义之前，在函数外预先说明了各个函数的类型，则在以后的各主调函数中，可不再对被调函数作说明。例如：

```
char str(int a);
float f(float b);
main()
{
…
}
char str(int a)
{
…
}
float f(float b)
{
…
}
```

以上第 1,2 行对 str()函数和 f()函数预先作了说明。因此，在以后各函数中无须对 str()和 f()函数再作说明就可直接调用。

(4) 对库函数的调用不需要再作说明，但必须把该函数的头文件用 include 命令包含在源文件前部。数组作为函数参数数组可以作为函数的参数使用，进行数据传送。

8.3.3　函数声明和函数定义的区别

函数使用的三大步：声明、定义和调用。

函数声明的作用是把函数的名字、类型及形参的类型、个数和顺序通知编译系统，以便在调用该函数时系统按此声明进行对照检查。

函数声明三要素：类型、名称和参数。

函数定义是指对函数功能的确立，包括指定函数名、函数值类型、形参及其类型、函数体等，它是一个完整的、独立的函数单位。

第 9 章 结 构 体

结构体是由不同数据类型组织在一起而构成的一种数据类型，因而一个结构体有多个数据项，每个数据项的类型可以不相同。

本章主要包括以下内容：

（1）结构体类型的定义；

（2）结构变量的定义；

（3）结构体变量的初始化；

（4）结构体变量的赋值；

（5）结构体变量的输入；

（6）结构体变量的输出。

9.1 引子

结构体到底是什么？

我们来思考一个问题，如果要保存一条学生信息，该如何去做呢？

该学生信息包括以下内容：学号、姓名、性别、年龄，两门课（语文、数学）的成绩，如表 9-1 所示。

表 9-1 学生信息表

学号	姓名	性别	年龄	语文成绩	数学成绩
0502	毛毛	F	19	86	98

稍微回忆以前学过的知识，大概能想到用数组去做，因为学号、姓名、成绩都是同类元素的集合，当然用数组了，如果用单个变量真的很麻烦。但是再进一步思考发现，学生的信息还是有不少东西的，比如学号、姓名、各科成绩、电话、家庭住址，如果单单用数组，是不是得好多好多数组呢，而且这样管理起来也非常不方便。其实在高级语言中，有一种类型，就是对基本类型进行重定义，把多个数据类型重新定义成一个新类型。

结构体成员变量访问的思考。

再思考一个问题，结构体是多种不同的数据类型的集合，所以每个元素的大小都是不一

样的，那么如何定义结构体？如何访问结构体中的成员变量呢？要是数组就好办，因为是同类，就以前学习的数组寻址公式一下子就访问到了。

9.2 结构体基本概念

9.2.1 结构体类型的定义

结构体类型的定义格式为

```
struct  结构体类型名
{  类型   数据类型成员名 1;
   类型   数据类型成员名 2;
   类型   数据类型成员名 3;
     …
   类型   数据类型成员名 n;
};
```

注意：结构体定义的花括号{}，最后一定要有分号。

结构体定义的关键字是 struct，表示一个信息结构，后面跟着的是结构体类型名，定义成一个新类型，得起个名字呀！就好比，现在定义了 1 个人，人由哪些构成呢，有名字、性别、年龄及身高等，并且不止定义 1 个人。

定义人：

```
struct Person
{
int num[2];                          //定义两个人,一男一女
char name[64];  /*定长的名字,如果这里给的类型是 char *就可以不定长了.但是用 char *寻址
要寻两次,用定长寻址一次就到啦!*是指针的概念,在此不解释*/
char sex;                            //性别
int age;                             //年龄
float height;                        //身高
};
```

定义人比较广泛，接下来以实际的例子来说明结构体的相关知识点。

描写一个学生的基本情况，涉及学号、姓名、性别、两门课的成绩，分别用 int num；char name[8]；char sex；float score[2]表示，需定义的结构体类型为

```
struct student
{
  int num ;
  char name[8];
  char sex;
```

```
  float score[2];
};
```

注意：(1) 此定义仅仅是结构体类型的定义，它说明了结构体类型的构成情况，C 语言并没有为之分配存储空间。

(2) 结构体中的每个数据成员称为分量或域，它们并不是变量，在实际应用中还需定义结构变量。

(3) struct student 是一个类型名，它和基本数据类型 int、float、char 等是一样的，都可以用来定义变量。

9.2.2 结构体变量的定义

当结构体类型定义完成后，就可以定义结构体变量，定义结构体变量有两种方法，分别为结构体类型与结构体变量同时定义和分开定义。

(1) 结构体类型与结构体变量同时定义

```
struct student
{
  int num ;
  char name[8];
  char sex;
  float score[2];
} stu;
```

(2) 结构体类型与结构体变量分开定义

分开定义是指先定义结构体类型，再定义结构体变量。

```
struct    结构体类型名    结构体变量表;
```

例如：

```
struct student stu;
```

stu 是结构体类型 student 的实例或对象，称为结构体类型变量或结构体变量。

例如：

```
struct student                    /* 声明一个结构体类型 */
 {
      int num;
      char name[20];
      char sex;
      int age;
      float score;
```

```
    struct date birthday;          /* birthday 是 struct date 类型 */
    char addr[30];
}student1,student2;
```

先声明一个 struct date 类型，它代表“日期”，包括 3 个成员：month(月)、day(日)、year(年)。然后在声明 struct student 类型时，将成员 birthday 指定为 struct date 类型，定义如下：

```
struct birthday
{
  int Month ;
  int day;
  int year;
  } birth1;
```

结构体嵌套示意如图 9-1 所示。

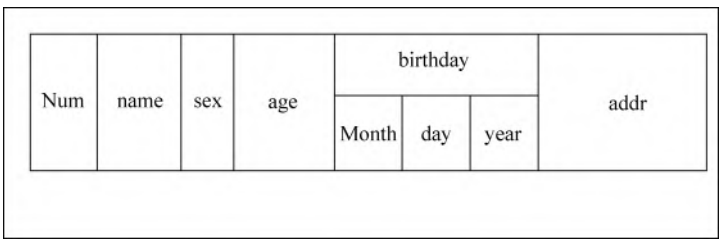


图 9-1　结构体嵌套

9.2.3　结构体变量占据的内存空间

定义了结构体变量 stu 之后，C 编译程序自动为结构体变量的所有成员分配足够的内存，如图 9-2 所示，结构体变量所占的存储空间是结构体类型各成员所占空间之和。

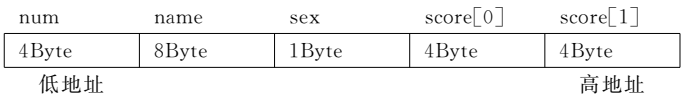


图 9-2　结构体变量 stu 占用内存的情况

在实际应用中，可用语句

```
sizeof(struct student);
```

测试结构体变量占用内存空间的大小。

9.2.4　结构体变量对结构体成员的引用

通过圆点(.)操作符可访问结构体中的成员。访问结构体成员的一般形式为

```
结构体变量名.成员名;
```

表示结构体变量对具体成员的引用，以下代码把 2001 赋给结构体变量 stu1 的成员 num：

```
stu1.num = 2001;
```

在屏幕上显示 stu 的成员 num 所含的学号值，应写为

```
printf("%d",stu.num);
```

同理，从键盘读入的语句是

```
scanf("%d",&stu.num);
```

9.2.5 结构体变量的赋值

C 语言不允许结构体变量整体进行输入和输出，只能对结构体变量的成员进行输入和输出。因此给结构体变量的赋值方法有三种：初始化，赋值和从键盘读入。

(1) 初始化

结构体变量可在声明时直接进行初始化。初始化数据应放在大括号中，并根据成员变量的声明顺序排列，同时数据之间的类型应一致。例如：

```
struct student stu1 = {2001,"毛毛", 'M',86,92};
```

或者为

```
struct student
{
  int num ;
  char name[8];
  char sex;
  float score[2];
}stu1 = {2001,"毛毛", 'M',86,92};
```

【实例分析 9-1】 定义结构类型及其变量，在声明时直接进行初始化，最后在终端中输出结构体变量各成员的值。

分析：初始化结构体变量 stu1，在初始化时直接给各成员变量赋初值，在主程序中用函数 printf()直接输出结构体各成员变量的值。

程序 9-1 结构体举例

```
#include<stdio.h>
#include<stdlib.h>
struct  student
{
  int num ;
  char name[8];
```

```
  char sex;
  float score[2];
} stu1 = {2001,"毛毛", 'M',86,92};

int main()
{
   printf(" %d\t %s\t %c\t %f\t %f\n",stu1.num,stu1.name,stu1.sex,stu1.score[0],stu1.
score[1]);
   system("pause");
return 0;
}
```

编译成功后，运行程序 9-1，结果如图 9-3 所示。

图 9-3　程序 9-1 运行结果

（2）对结构体变量的数据成员进行逐个赋值。

【实例分析 9-2】 定义一个结构体类型及其结构体变量，逐个给结构体变量赋值，最后输出结构体变量的值。

首先进行变量的存储，并在程序运行时逐个给成员变量赋值，最后在终端中输出毛毛同学的个人信息。

分析：定义结构体类型 student 及结构体变量 stu1，在程序中逐个给成员变量赋值，最后用函数 printf()直接输出结构体各成员变量的值，详细代码如程序 9-2 所示。

程序 9-2　结构体举例——逐个赋值

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct  student
{
  int num ;
  char name[8];
  char sex;
  folat score[2];
}stu1;
int main()
{
 stu1.num = 2010;
 strcpy(stu1.name,"毛毛");          // 不可写成 stu1. name = "毛毛";
 stu1.sex = 'M';
 stu1.score[0] = 86;
```

```
 stu1.score[1] = 92;
 printf(" % d\t % s\t % c\t % f\t % f\n",stu1.num,stu1.name,stu1.sex,stu1.score[0],stu1.score
[1]);
system("pause");
return 0;
}
```

编译成功后，运行程序 9-2，结果如图 9-4 所示。

图 9-4 程序 9-2 运行结果

(3) 可用单赋值语句把一个结构体变量的全部内容赋给另一个同类结构体变量，而不必逐个成员地多次赋值。

【实例分析 9-3】 利用结构体变量存储，并在初始化时给成员变量赋值，把所有成员变量的值传递给另一个结构体变量，利用第二个结构体变量，在终端中输出毛毛同学的个人信息。

分析：初始化结构体变量 stu1，在初始化时直接给各成员变量赋初值，在主程序中定义一个结构相同的结构体变量 stu2，用单赋值语句把结构体变量 stu1 的全部内容赋给第二个同类结构体变量 stu2，最后用函数 printf()直接输出结构体变量 stu2 各成员变量的值，详细代码如程序 9-3 所示。

程序 9-3 结构体举例——赋值

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct  student
{
  int num ;
  char name[8];
  char sex;
  float score[2];
}stu1 = {2001,"毛毛", 'M',86,98};
int main()
{
  struct student stu2;
  stu2 = stu1;
  printf(" % d\t % s\t % c\t % f\t % f\n",stu2.num,stu2.name,stu2.sex,stu2.score[0],stu2.
score[1]);
system("pause");
return 0;
}
```

编译成功后，运行程序 9-3，结果如图 9-5 所示。

```
2010    毛毛    M       86      98
请按任意键继续. . .
```

图 9-5　程序 9-3 运行结果

(4) 从键盘逐个读入结构体数据成员。

【实例分析 9-4】 利用结构体变量存储，从键盘中逐个读入结构体数据成员，给成员变量赋值，在终端中输出毛毛同学的个人信息。

分析：初始化结构体变量 stu1，在初始化时并不赋初值，在主程序中等待键盘输入，依次把键盘的输入赋值给结构体变量 stu1 的各成员变量，最后用函数 printf()直接输出结构体变量 stu1 各成员变量的值，详细代码如程序 9-4 所示。

程序 9-4　结构体举例——逐个读入结构体数据成员

```
#include<stdlib.h>
#include<string.h>
struct  student
{
  int num ;
  char name[8];
  int score[2];
}stu1;

int main()
{
  scanf("%d%s%d%d",&stu1.num,&stu1.name,&stu1.score[0],&stu1.score[1]);
  printf("%d\t%s\t%d\t%d\n",stu1.num,stu1.name,stu1.score[0],stu1.score[1]);
system("pause");
return 0;
}
```

编译成功后，运行程序 9-4，结果如图 9-6 所示。

```
2001   maomao 98 99
2001    maomao  98      99
请按任意键继续. . .
```

图 9-6　程序 9-4 运行结果

9.3　结构体类型的数组

结构体的最常见用法是结构体数组(Array of Structures)。例如，描述一个班级的学生。用结构体类型中不同类型的成员变量描述学生的具体属性，用数组类型描述拥有相同

属性的一个班级的学生情况，以便使用循环对数组元素进行统一处理，优化算法。

9.3.1 结构体数组变量的定义

定义结构体数组变量和定义结构体变量的方法相仿，说明其为数组即可。

(1) 先定义结构体类型，再定义结构体数组。例如：

```
struct   student
  {
    int num ;
      char name[8];
      char sex;
      float score[2];
  } ;
struct  student  st[6] ;
```

(2) 定义结构体类型的同时定义结构体数组。例如：

```
struct   student
{
  int num ;
  char name[8];
  char sex;
  float score[2];
} st[6] ;
```

9.3.2 结构体数组的引用

结构体数组的引用如下：

(1) 引用某个数组元素的成员。例如：

```
puts( st[0]. name ) ;
printf(" % d, % d",st[1]. age,st[1]. s1 ) ;
```

(2) 数组元素之间可以整体赋值，也可以将一个元素赋给一个相同类型的结构体变量。例如：

```
struct  student  st[3] = { {"Mary",21,78,86},{"Alex",… } } ;
struct  student   x ;
st[2] =   st[0] ;
x   =   st[1] ;
```

st[2] = st[0] 和 x = st[1] 都是结构体变量的整体赋值形式。

(3) 输入和输出操作只能对数组元素的成员进行，和结构体变量方法相同。

9.3.3 结构体数组的初始化

结构体数组的初始化与其他类型的数组类似。将每个数组元素的数据用花括号{}括起

来，一般形式是在定义数组的后面加上“＝{初值表列}；”。

例如：

```
struct student
{
  int num ;
  char name[8];
  char sex;
  float score[2];
}stu[2] = {{4001,"刘", 'M',86.5,97.3},{4002,"张", 'F',78.4,86.5}};
```

说明：

(1) 如果对于数组中的元素全部赋值，长度可省略，例如：

```
struct student st[ ] = { {4001,"刘", 'M',86.5,97.3},
                         {4002,"张", 'F',78.4,86.5},
                         {4003,"王", 'F',74.6,89.8} };
```

(2) 对数组中的部分元素赋初值，例如：

```
struct student st[3] = { {4001,"刘", 'M',86.5,97.3}};
struct student st[ ] = { {4001,"刘", 'M',86.5,97.3},
                         {0},
                         {4003,"王", 'F',74.6,89.8} };
```

(3) 数组中的内层括号可省略，例如：

```
struct student st[3] = {4001,"刘", 'M',86.5,97.3,4002,"张", 'F',78.4,86.5,4003,"王", 'F',74.6,
89.8};
```

【实例分析 9-5】 对候选人得票的统计程序。假设有 3 个候选人，每次输入一个得票的候选人的名字，要求最后输出各人得票结果。详细代码如程序 9-5 所示。

程序 9-5　结构体数组举例——候选人数组的后面加上“＝{初值表列}”

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct person
{
 char name[20];
 int count;
 } leader[3] = {"Li",0,"Zhang",0,"Wang",0} ;
int main()
{
 int i,j;
 char leader_name[20];
```

```
 for(i=1;i<=10;i++)
  {
   scanf("%s",leader_name);
   for(j=0;j<3;j++)
   if(strcmp(leader_name,leader[j].name)==0)
   leader[j].count++;
  }
  printf("\n");
  for(i=0;i<3;i++)
  printf("%5s:%d\n",leader[i].name,leader[i].count);
system("pause");
return 0;
}
```

编译成功后，运行程序 9-5，结果如图 9-7 所示。

图 9-7　程序 9-5 运行结果

程序 9-5 定义一个全局的结构体数组 leader，它有 3 个元素，每一个元素包含两个成员 name(姓名)和 count(票数)。在定义此数组时初始化，使 3 位候选人的票数都先置零。在主函数中定义字符数组 leader-name，它代表候选人的姓名，在 10 次循环中每次先输入一个候选人的姓名，然后把它与 3 个候选人姓名相比，看和哪一个候选人的名字相同。在输入和统计结束之后，将 3 人的名字和得票数输出。

【实例分析 9-6】 逐一赋值，求 3 个学生各自成绩的平均分，详细代码如程序 9-6 所示。

程序 9-6　结构体数组举例——求 3 个学生各自成绩的平均分

```
#include<stdio.h>
#include<stdlib.h>
////先定义结构体
struct student
{ char num[6];                    //学号
  char name[15];                  // 姓名
  int score[2];                   //两门课成绩
  float avr;                      //平均分
};

int main()
{
  int i,j,sum;
  struct student stu[3];          //结构体数组,3 个学生
  for(i=0;i<3;i++)
  {
```

```
    printf("输入学生学号:");
    scanf("%s",&stu[i].num);      //输入学生学号
    printf("输入学生姓名:");
    scanf("%s",&stu[i].name);     //输入学生姓名
    sum=0;
    for(j=0;j<2;j++)
    {
        printf("成绩%d  ",j+1);
    scanf("%d",&stu[i].score[j]);//输入学生两门课的成绩
        sum+=stu[i].score[j];
    }
stu[i].avr=sum/2.0;               //求学生平均分
}
for(i=0;i<3;i++){                 //输出各学生的信息
  printf("%s    %s    %d    %d    %f\n",stu[i].num,stu[i].name,stu[i].score[0],stu
[i].score[1],stu[i].avr);
    }
system("pause");
return 0;
}
```

编译成功后,运行程序 9-6,结果如图 9-8 所示。

```
输入学生学号:201501
输入学生姓名:maomao
成绩1  98
成绩2  96
输入学生学号:201502
输入学生姓名:qiuqiu
成绩1  88
成绩2  98
输入学生学号:201503
输入学生姓名:junjun
成绩1  96
成绩2  99
201501maomao    maomao   98   96   97.000000
201502qiuqiu    qiuqiu   88   98   93.000000
201503junjun    junjun   96   99   97.500000
请按任意键继续. . .
```

图 9-8　程序 9-6 运行结果

【实例分析 9-7】 输入 5 位同学的一组信息,包括学号、姓名、数学成绩、计算机成绩,求得每位同学的平均分和总分,然后按照总分从高到低排序。详细代码如程序 9-7 所示。

程序 9-7　结构体数组举例

```
#include<stdio.h>
#include<stdlib.h>
struct mes
{   int sno;
char sname[20];
```

```
float grade1;
float grade2;
float sum;
float avg;
}student [5];                          //定义结构体变量数组
int main()
{
  int i,j,k;
  struct  mes temp;
  printf("请输入五位学生的信息\n");
  printf("学号\t 姓名\t 数学\t 计算机\n");
  for(i = 0;i < 5;i++)
  {
scanf(" % d\t % s\t % f\t % f",&student[i]. sno,student[i]. sname,&student[i]. grade1,&student
[i]. grade2);student[i]. sum = student[i]. grade1 + student[i]. grade2;student[i]. avg = student
[i]. sum/2;
}                                      //输入每位学生间隔的信息时按 Tab 键
  for(i = 0;i < 4;i++)
  {
  k = i;
  for(j = i + 1;j < 5;j++)
    if(student[k]. sum < student[j]. sum)
    k = j;
    temp = student[k];
    student[k] = student[i];
    student[i] = temp;
  }
  printf("学生成绩的排序结果为:\n");
  for(i = 0;i < 5;i++)
  {
  printf("学号: % d,姓名: % s,数学成绩: % 3.1f,计算机成绩: % 3.1f: % 3.1f,总分: % 3.1f\n",
student[i]. sno, student[i]. sname, student[i]. grade1, student[i]. grade2, student[i]. avg,
student[i]. sum);
  }                                    //显示 5 位同学的信息
  system("pause");
return 0;
}
```

编译成功后,运行程序 9-7,结果如图 9-9 所示。

```
请输入五位学生的信息
学号    姓名    数学    计算机
201501  wang    90        98
201502  jian    88        90
201503  meng    86        80
201504  ling    66        90
201505  feng    80        88
学生成绩的排序结果为:
学号:201501,姓名:wang,数学成绩:90.0,计算机成绩:98.0:94.0,总分:188.0
学号:201502,姓名:jian,数学成绩:88.0,计算机成绩:90.0:89.0,总分:178.0
学号:201505,姓名:feng,数学成绩:80.0,计算机成绩:88.0:84.0,总分:168.0
学号:201503,姓名:meng,数学成绩:86.0,计算机成绩:80.0:83.0,总分:166.0
学号:201504,姓名:ling,数学成绩:66.0,计算机成绩:90.0:78.0,总分:156.0
请按任意键继续. . .
```

图 9-9　程序 9-7 运行结果

【实例分析 9-8】 定义一个结构体变量(包括年、月、日)。编写一个函数 days(),计算该日期在本年中是第几天(注意闰年问题)。由主函数将年月日传递给 days()函数,计算之后,将结果传回到主函数输出。详细代码如程序 9-8 所示。

程序 9-8 结构体数组举例——日期计算

```
#include <stdio.h>
#include <stdlib.h>
struct date
{
    int year;
    int month;
    int day;
};
/*判断闰年*/
int leap_year(int a)
{
    if(a%400==0||(a%4==0&&a%100!=0))
       return 1;
    else
       return 0;
}
/*计算输入的日期是本年度的第几天*/
int cal_day(struct date a)
{
    int sum=0,b[]={31,28,31,30,31,30,31,31,30,31,30,31};
    for(int i=0;i<a.month-1;i++)
        sum+=b[i];
    if(a.month>2)
        sum=sum+a.day+leap_year(a.year);
    else
        sum=sum+a.day;
    return sum;
}

int main()
{
    struct date a;
    int n;
    printf("请输入年月日(年月日中间用空格隔开)\n");
    scanf("%d%d%d",&a.year,&a.month,&a.day);
    n=cal_day(a);
    printf("该日是本年中的第%d天\n",n);
    system("pause");
```

```
return 0;
}
```

编译成功后，运行程序 9-8，结果如图 9-10 所示。

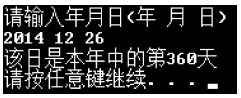


图 9-10　程序 9-8 运行结果

附录 A

Dev C++安装说明

C 和 C++都是编译语言，不能直接运行 C 或者 C++源代码。要想运行用 C 语言或者 C++语言编写的程序，必须使用编译器将 C 或者 C++源文件编译成可执行文件。从源代码到可执行文件要经历三个步骤：预处理、编译，链接。

常用的编译器有：Turbo C、Turbo C++（这两个太旧，不推荐使用）、GCC、MicroSoft Visual C++6.0、Dev C++等。

Dev C++是一个 Windows 环境下 C/C++的集成开发环境（IDE），它是一款自由软件，遵守 GPL 许可协议分发源代码。它集合了 MinGW 等众多自由软件，并且可以取得最新版本的各种工具支持，而这一切都是来自全球的狂热者所做的工作。Dev C++使用 MingW64/TDM-GCC 编译器，遵循 C++11 标准，同时兼容 C++98 标准。开发环境包括多页面窗口、工程编辑器及调试器等，在工程编辑器中集合了编辑器、编译器、链接程序和执行程序，提供高亮度语法显示以减少编辑错误，还有完善的调试功能，Dev C++是 NOI（全国青少年信息学奥林匹克竞赛）、NOIP（全国青少年信息学奥林匹克联赛）和 ACM/ICPC 等比赛的指定工具，适应初学者与编程高手的不同需求，是学习 C 或 C++的首选开发工具。

（1）双击 Dev C++ 4.9.9.2.exe ，一路点击“同意”或“下一步”按钮即可（可参考步骤（2））。

（2）步骤分解（英文版本或者初次安装软件者使用），如图 A-1 至图 A-8 所示。

图 A-1


Dev-C++ 5 beta 9 release (4.9.9.2)
License Agreement
Please review the license terms before installing Dev-C++ 5 beta 9 release (4.9.9.2).
Press Page Down to see the rest of the agreement.
Bloodshed Dev-C++ is distributed under the GNU General Public License.
Be sure to read it before using Dev-C++.
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Dev-C++ 5 beta 9 release (4.9.9.2).
Nullsoft Install System v2.0
I Agree
Cancel

图 A-2

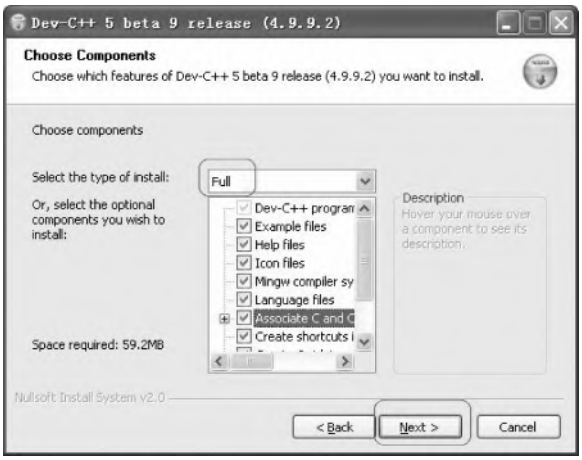

Dev-C++ 5 beta 9 release (4.9.9.2)
Choose Components
Choose which features of Dev-C++ 5 beta 9 release (4.9.9.2) you want to install.
Choose components
Select the type of install:
Full
Or, select the optional components you wish to install:
Dev-C++ program
Example files
Help files
Icon files
Mingw compiler sy
Language files
Associate C and C
Create shortcuts i
Description
Hover your mouse over a component to see its description.
Space required: 59.2MB
Nullsoft Install System v2.0
< Back
Next >
Cancel

图 A-3

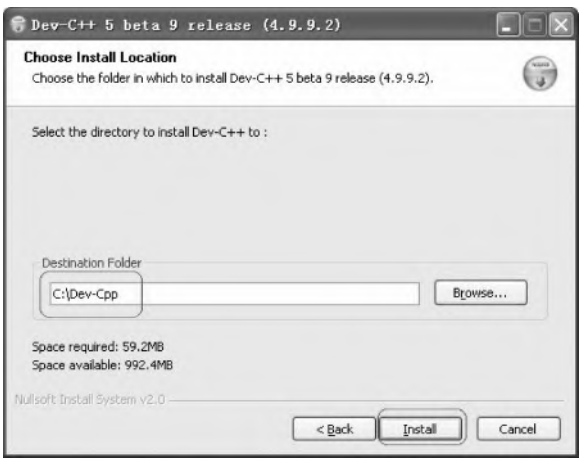

Dev-C++ 5 beta 9 release (4.9.9.2)
Choose Install Location
Choose the folder in which to install Dev-C++ 5 beta 9 release (4.9.9.2).
Select the directory to install Dev-C++ to :
Destination Folder
C:\Dev-Cpp
Browse...
Space required: 59.2MB
Space available: 992.4MB
Nullsoft Install System v2.0
< Back
Install
Cancel

图 A-4


Dev-C++ 5 beta 9 release (4.9.9.2)
Completing the Dev-C++ 5 beta 9 release (4.9.9.2) Setup Wizard
Dev-C++ 5 beta 9 release (4.9.9.2) has been installed on your computer.
Click Finish to close this wizard.
Run Dev-C++ 5 beta 9 release (4.9.9.2)
< Back
Finish
Cancel

图　A-5

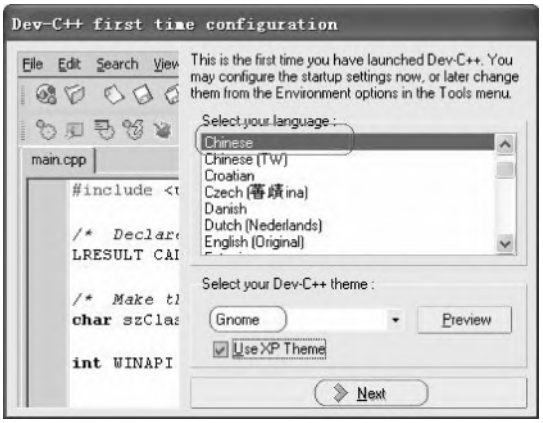

Dev-C++ first time configuration
File Edit Search View
main.cpp
This is the first time you have launched Dev-C++. You may configure the startup settings now, or later change them from the Environment options in the Tools menu.
Select your language :
Chinese
Chinese (TW)
Croatian
Czech
Danish
Dutch (Nederlands)
English (Original)
Select your Dev-C++ theme :
Gnome
Preview
Use XP Theme
Next

图　A-6

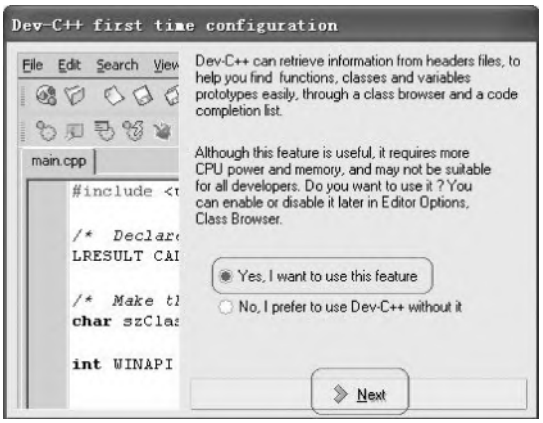

Dev-C++ first time configuration
File Edit Search View
main.cpp
Dev-C++ can retrieve information from headers files, to help you find functions, classes and variables prototypes easily, through a class browser and a code completion list.
Although this feature is useful, it requires more CPU power and memory, and may not be suitable for all developers. Do you want to use it ? You can enable or disable it later in Editor Options, Class Browser.
Yes, I want to use this feature
No, I prefer to use Dev-C++ without it
Next

图　A-7

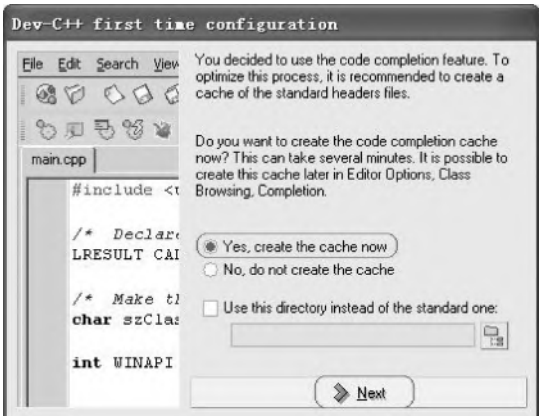


图 A-8

注：不同版本安装方式略有不同。

附录 B

DEV C++使用说明

Dev C++是一个可视化集成开发环境，可以用此软件实现 C/C++程序的编辑、预处理、编译、链接、运行和调试。附录 B 中介绍了 Dev C++常用的一些基本操作。

1. 启动 Dev C++

方法一：

（1）单击任务栏中的“开始”按钮，然后单击“程序”菜单项，选中“程序”下的子菜单项“Bloodshed Dev C++”，显示该项下的子菜单。

（2）单击“Dev-C ++”菜单项，即启动 Dev-C ++集成开发工具。如图 B-1 所示。

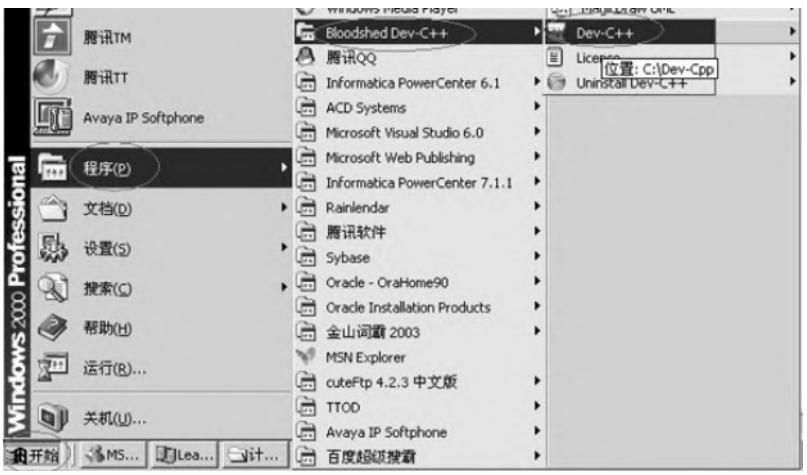


图 B-1　启动界面

方法二：

直接双击桌面上的 Dev C++的图标。

2. 新建、保存、编写源程序

（1）从主菜单选择 File→New→Source File（“文件”→“新建”→“源代码”）命令即可，如图 B-2 所示。

如果界面是中文的，可以根据下面操作将界面改为英文。单击主菜单“工具”→“环境”

(a) 英文界面　　(b) 中文界面

图 B-2　新建源文件

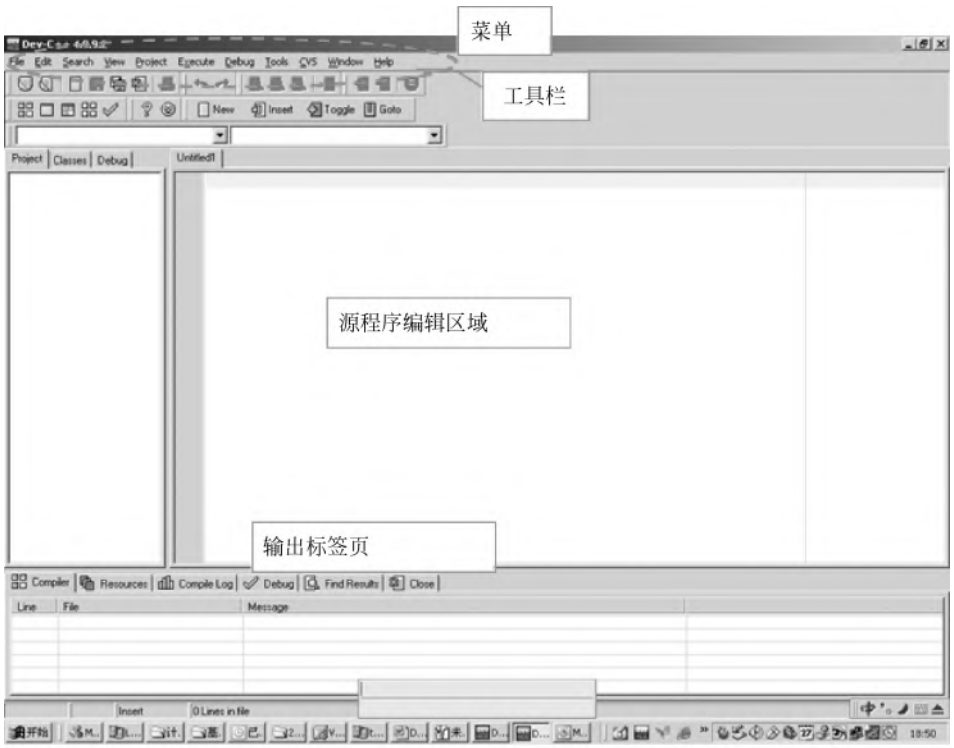
选项，在弹出的对话框中单击“界面”选项卡，在“语言”下拉列表中选择“English”即可，如图 B-3 所示。此时界面上的菜单、工具条等全部以英文命名。

图 B-3　更换语言界面

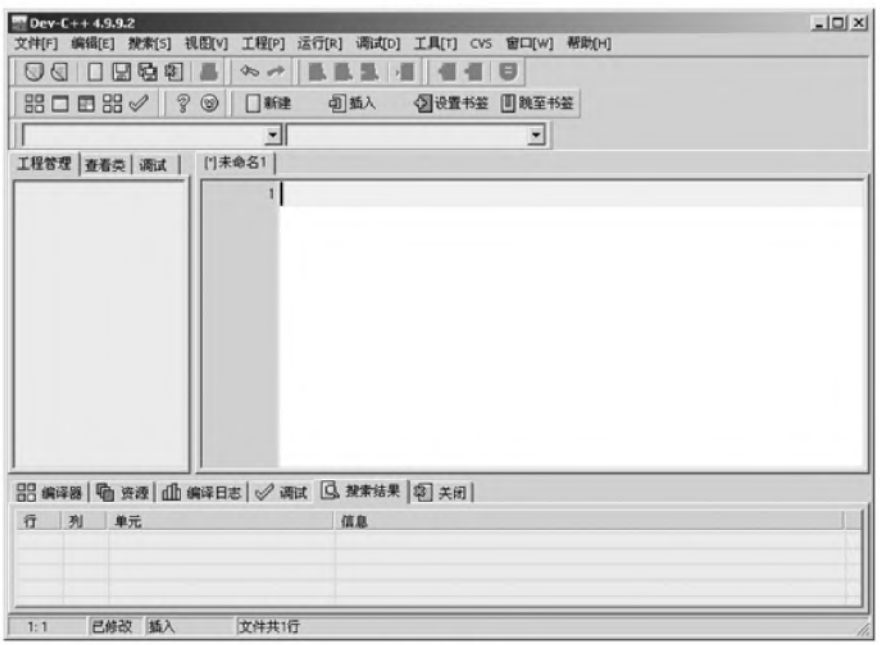
(2) 此时屏幕右下侧出现一片白色区域，可以在此输入程序。如图 B-4 所示。

(3) 保存源程序到硬盘，养成一个好习惯在创建了一个新程序后，还未输入代码之前，先将该程序保存到硬盘某个目录下，然后在程序的编辑过程中经常性地进行保存，以防止机器突然断电或者死机。要保存程序，只需从主菜单选择 File→Save(“文件”→“保存”)命令就可以将文件保存到指定的硬盘目录。如图 B-5 所示。

此时会弹出一个对话框，如图 B-6 所示。在此需要指定文件要存放的目录(此处为 D:\temp)，文件名称(此处为 test)及保存类型。在保存类型处选择 C source files(*.c)，意思是保存的是一个 C 文件。在单击右下角的“保存”按钮后，在 temp 目录下出现一个名为 test.c 的源文件。

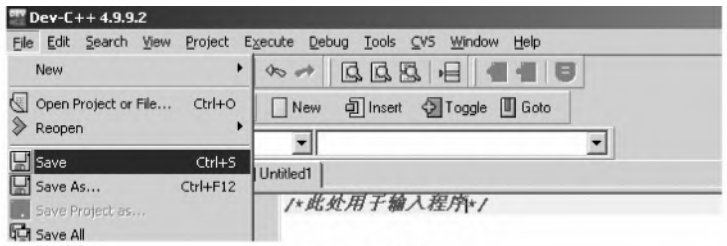


(a) 英文界面



(b) 中文界面

图 B-4　输入程序界面



(a) 英文界面

(b) 中文界面

图 B-5　保存

图 B-6　保存 C 语言文件

(4) 在程序编辑区域编辑程序，如图 B-7 所示。在输入程序的过程中要随时对程序进行保存(使用菜单 File→Save，或者用快捷键 Ctrl+s)，此时会将程序重新保存到之前指定的目录下，例如 D:\temp。如果想将程序保存到其他路径下，可以选择 File→Save As…，如图 B-8 所示，可以重新指定程序的名称和保存路径。

图 B-7　随时对程序进行保存

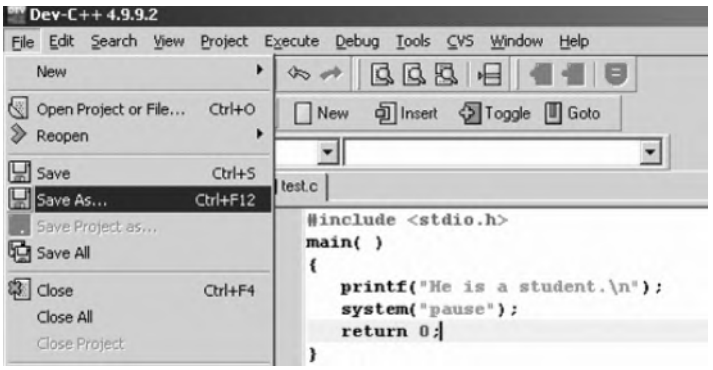


图 B-8　程序另存为

注意：① 必须在英文输入环境下编辑程序(如果当前能在程序编辑区输入中文，说明是在中文输入环境下。此时必须切换到英文输入环境下)。

② 在 Dev C++环境中，为了查看程序运行结果，需要在 main()函数的 return 语句前加上 system("PAUSE")或者 system("pause")；，这样程序运行到该语句时，结果显示屏幕将会停留。否则结果显示屏幕将会一闪而过。

(5) 编译环境的设置(可选)

图 B-9 中间偏左的地方有一个黑色竖条，是用来显示行数的，设置方法为 Tools→Editor Options→Display，如图 B-10 所示。

选中 Line Numbers(行数)，然后单击 OK(确定)按钮即可。也可对编辑区域的字体大小、颜色等进行设置。

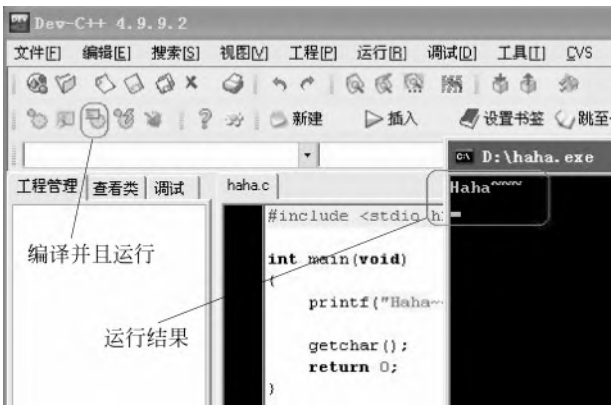


图 B-9 编译、运行示意图

图 B-10 设置行数

3. 预处理、编译、链接程序

从主菜单选择 Excute→compile 命令(也可编译当前文件)或使用快捷键 Ctrl+F9,可以一次性完成程序的预处理、编译和链接。如果程序中存在词法、语法等错误,则编译过程失败,编译器将会在屏幕右下角的"Compile Log"标签页中显示错误信息,如图 B-11 所示,并且将源程序相应的错误行标成红色底色,如图 B-12 所示(由于删除了 printf 语句后面的分号,编译时报错,提示 system 语句前面的语句有语法错误(syntax error))。

"Compile Log"标签页中显示的错误信息是寻找错误原因的重要信息来源,要学会看这些错误信息,并且每一次碰到错误并且最终解决了时,要记录错误信息及相应的解决方法。

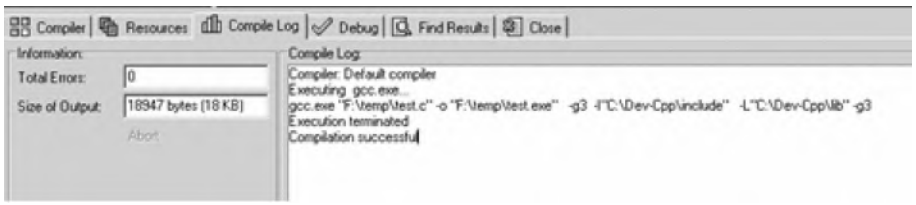


图 B-11　编译日志

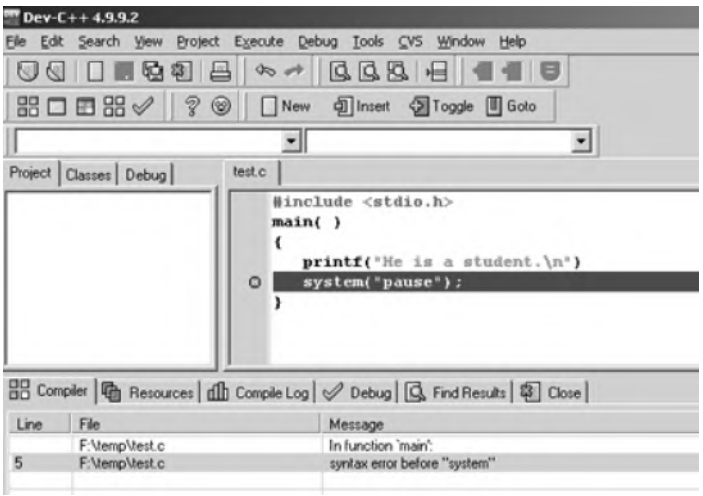


图 B-12　错误提示

以后看到类似的错误提示信息，就知道是源程序哪里有问题，从而提高程序调试效率。

排除了程序中存在的词法、语法等错误后，编译成功。此时源文件所在目录下会出现一个同名的.exe 可执行文件(如 test.exe)。双击这个文件，即可编译成功，并运行程序。

4. 编译成功后运行程序

对程序进行预处理、编译和链接后，有两种方法编译成功，并运行程序。

(1) 双击生成的.exe 文件；

(2) 直接在 Dev C++环境下从主菜单选 Excute→Run 命令或使用快捷键 Ctrl+F10 编译成功后，运行程序。如图 B-13 所示。

5. 调试程序

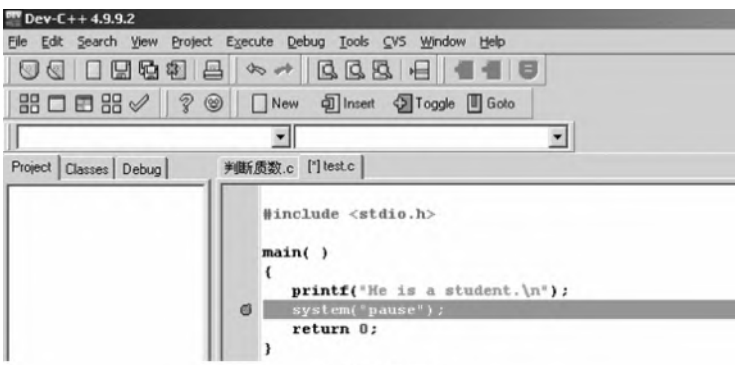
通过预处理、编译和链接的程序仅仅是该程序中没有词法和语法等错误，而无法发现程序深层次的问题(例如算法不对导致结果不正确)。当程序运行出错时，需要找出错误原因。仔细读程序来寻找错误固然是一种方法，但有时光靠读程序已经解决不了问题，此时需要借助于程序调试(Debug)手段。这是一种有效的排错手段，每一位同学都需要掌握。

(a) 英文界面　　(b) 中文界面

图 B-13　运行图

1）设置程序断点

调试的基本思想是让程序运行到可能有错误的代码前，然后停下来，在控制下逐条语句地运行，通过在运行过程中查看相关变量的值，来判断错误产生原因。如果想让程序运行到某一行前能暂停下来，就需要将该行设成断点。在代码所在行行首单击，该行将被加亮，默认的加亮颜色是红色。如图 B-14 所示，将 system("pause")语句设成断点，则程序运行完 printf 语句后，将会暂停。可以在程序中根据需要设置多个断点。



(a) 英文界面

(b) 中文界面

图 B-14　设置程序断点

如果想取消断点，在此单击代码行首即可。

2）编译成功后运行程序

设置断点后，程序运行进入 debug(调试)状态。要想编译成功后运行程序，就不能使用主菜单 Execute→Run 命令，而是需要用主菜单 Debug→Debug 命令或者按快捷键 F8，如图 B-15 所示。

(a) 英文界面　　(b) 中文界面

图 B-15　编译成功后运行程序

程序将运行到第一个断点处，此时断点处亮色由红色变成蓝色，表示接下去将运行蓝色底色的代码，如图 B-16 所示。

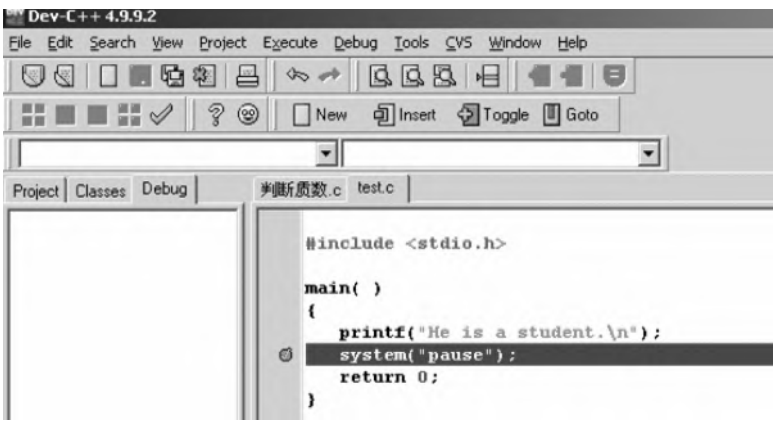


图 B-16　运行到断点处

注意：有时会发现即使设置了断点，单击主菜单 Debug→Debug 命令，程序还是不在断点处停留。解决方法：取消断点，重新编译程序，然后再次设置断点，单击主菜单 Debug→Debug 命令即可。

3）单步执行程序

要想运行蓝色底色的代码，可以使用图 B-17 所示的 Next Step(F7)、Step Into(Shift＋F7)、Continue(Ctrl＋F7)，Run to Cursor(Shift＋F4)等命令。在学习函数之前，一般用的是 Next Step 和 Continue。学习函数后，还会用到 Step Into。

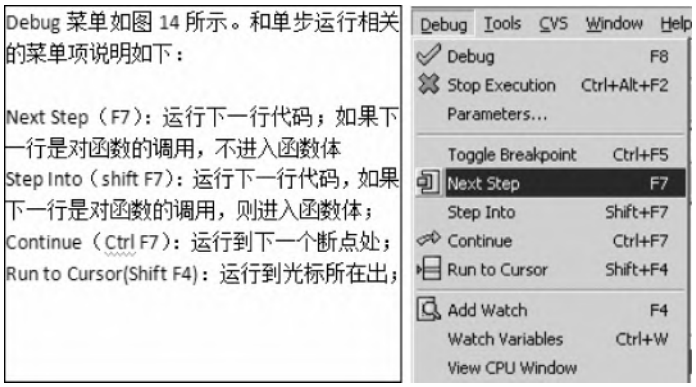


图 B-17 “Debug”菜单

4）设置 watch 窗口

在调试程序时，可能要看程序运行过程中变量的值，以检测程序对变量的处理是否正确，可以在调试时，通过调试菜单下的添加变量(Add Watch)窗口来增加变量 watch，新增的变量将会显示在最左边 Explore 的 Debug 页中，如图 B-18 所示。如果左边 Explore 中的当前页不是 Debug 页，可以单击 Debug 标签使之成为当前页。

6. 打开一个已经存在的程序

单击主菜单的 File→Open Project or File 命令，如图 B-19 所示，在弹出的对话框中指定文件所在的路径，选择要打开的文件即可。

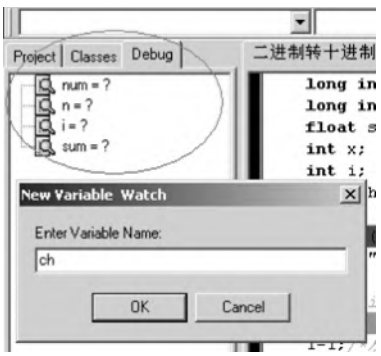


图 B-18 增加变量 watch

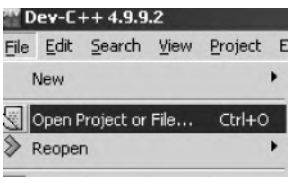


图 B-19 打开一个已经存在的程序

7. 提高程序书写风格的一些操作

1）整段缩进

运用适当的缩进，可以提高代码的可读性。选中要缩进的代码段，单击主菜单下的 Edit→Indent 命令，如图 B-20 所示，即可将整段代码右移 N 个字符。

图 B-20　缩进的代码段

每一次缩进要移动的字符数可以定制，方法：单击主菜单下的 Tools→Edit Options 命令，在弹出对话框中的 General 标签页中进行设置，将 Tab Size 设置成希望的数字，建议设成 3，如图 B-21 所示。该对话框提供了定制 Dev C++的界面编辑风格的功能。

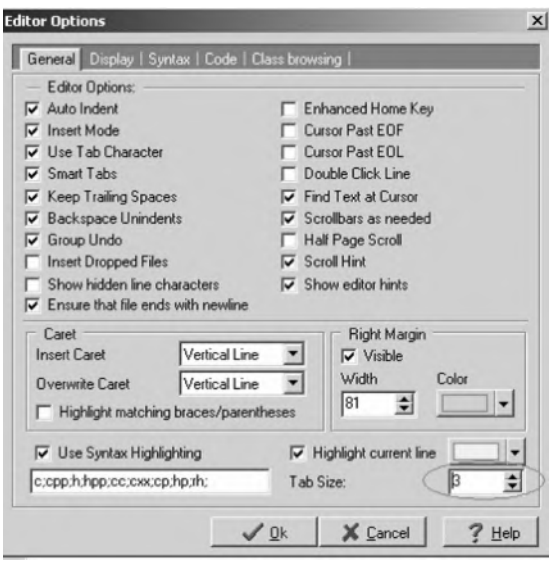


图 B-21　缩进设置

也可以单击 Edit→Unindent 命令使整段代码往左移。

2）插入程序说明

在源程序里简要说明程序的功能，是一个良好的习惯，可以单击 Edit→Insert→Comment Header 命令，从而在程序编辑区光标处插入一段注释，如图 B-22 所示。

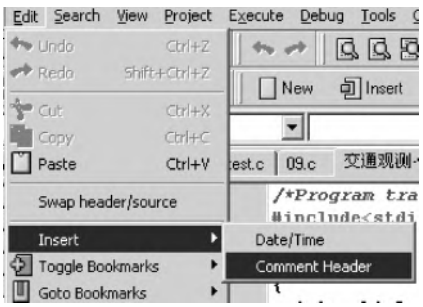


图 B-22　插入一段注释

插入后的效果如图 B-23 所示。

```
/*
  Name:
  Copyright:
  Author:
  Date: 27-12-14 05:13
  Description:
*/
#include<stdio.h>
#include<stdlib.h>
struct Date
{
```

图 B-23　插入注释效果

附录 C 常见错误信息中英语句索引

Ambiguous operators need parentheses：不明确的运算需要用括号括起来
Ambiguous symbol ´xxx´：不明确的符号
Argument list syntax error：参数表语法错误
Array bounds missing] in function main：缺少数组界限符"]"
Array bounds missing：丢失数组界限符
Array size too large：数组尺寸太大
Bad character in paramenters：参数中有不适当的字符
Bad file name format in include directive：包含命令中文件名格式不正确
Bad ifdef directive synatax：编译预处理 ifdef 有语法错误
Bad undef directive syntax：编译预处理 undef 有语法错误
Bit field too large：位字段太长
Call of non-function：调用未定义的函数
Call to function with no prototype：调用函数时没有函数的说明
Cannot modify a const object：不允许修改常量对象
Case outside of switch：case 出现在 switch 语法外
Case syntax error：Case 语法错误
Code has no effect：代码不可述不可能执行到
Compound statement missing{：分程序漏掉"{"
Conflicting type modifiers：不明确的类型说明符
Constant expression required：要求常量表达式
Constant out of range in comparison：在比较中常量超出范围
Conversion may lose significant digits：转换时会丢失有意义的数字
Conversion of near pointer not allowed：不允许转换近指针
Could not find file ´xxx´：找不到 xxx 文件
Declaration missing ;：说明缺少"；"
Declaration syntax error：说明中出现语法错误

Default outside of switch：Default 出现在 switch 语句之外
Define directive needs an identifier：定义编译预处理指令 Define 需要标识符
Division by zero：用零作除数
Do statement must have while：Do-while 语句中缺少 while 部分
Enum syntax error：枚举类型语法错误
Enumeration constant syntax error：枚举常数语法错误
Error directive：xxx：错误的编译预处理命令
Error writing output file：写输出文件错误
Expression syntax error：表达式语法错误
Extra parameter in call：调用时出现多余参数
File name too long：文件名太长
Function call missing)：函数调用缺少右括号
Fuction definition out of place：函数定义位置错误
Fuction should return a value：函数必需返回一个值
Goto statement missing label：Goto 语句没有标号
Hexadecimal or octal constant too large：十六进制或八进制常数太大
Illegal character ´x´：非法字符 x
Illegal initialization：非法的初始化
Illegal octal digit：非法的八进制数字
Illegal pointer subtraction：非法的指针相减
Illegal structure operation：非法的结构体操作
Illegal use of floating point：非法的浮点运算
Illegal use of pointer：指针使用非法
Improper use of a typedefsymbol：类型定义符号使用不恰当
In-line assembly not allowed：不允许使用行间汇编
Incompatible storage class：存储类别不相容
Incompatible type conversion：不相容的类型转换
Incorrect number format：错误的数据格式
Incorrect use of default：default 使用不当
Invalid indirection：无效的间接运算
Invalid pointer addition：指针相加无效
Irreducible expression tree：无法执行的表达式运算
Lvalue required：需要逻辑值 0 或非 0 值
Macro argument syntax error：宏参数语法错误
Macro expansion too long：宏的扩展太长
Mismatched number of parameters in definition：定义中参数个数不匹配

Misplaced break：此处不应出现 break 语句
Misplaced continue：此处不应出现 continue 语句
Misplaced decimal point：此处不应出现小数点
Misplaced elif directive：不应编译预处理 elif
Misplaced else：此处不应出现 else
Misplaced else directive：此处不应出现编译预处理 else
Misplaced endif directive：此处不应出现编译预处理 endif
Must be addressable：必须是可以编址的
Must take address of memory location：必须存储定位的地址
No declaration for function ´xxx´：没有函数 xxx 的说明
No stack：缺少堆栈
No type information：没有类型信息
Non-portable pointer assignment：对不可移动的指针(地址常数)赋值
Non-portable pointer comparison：不可移动的指针(地址常数)比较
Non-portable pointer conversion：不可移动的指针(地址常数)转换
Not a valid expression format type：不合法的表达式格式
Not an allowed type：不允许使用的类型
Numeric constant too large：数值常数太大
Out of memory：内存不够用
Parameter ´xxx´is never used：参数 xxx 没有用到
Pointer required on left side of－＞：符号－＞的左边必须是指针
Possible use of ´xxx´before definition：在定义之前就使用了 xxx(警告)
Possibly incorrect assignment：赋值可能不正确
Redeclaration of ´xxx´：重复定义了 xxx
Redefinition of ´xxx´is not identical：xxx 的两次定义不一致
Register allocation failure：寄存器定址失败
Repeat count needs an lvalue：重复计数需要逻辑值
Size of structure or array not known：结构体或数组大小不确定
Statement missing ;：语句后缺少"；"
Structure or union syntax error X：结构体或联合体语法错误
Structure size too large：结构体尺寸太大
Sub scripting missing]：下标缺少右方括号
Superfluous & with function or array：函数或数组中有多余的"&"
Suspicious pointer conversion：可疑的指针转换
Symbol limit exceeded：符号超限
Too few parameters in call：函数调用时的实参少于函数的参数

Too many default cases Default：太多(switch 语句中的一个)
Too many error or warning messages：错误或警告信息太多
Too many type in declaration：说明中类型太多
Too much auto memory in function：函数用到的自动存储太多
Too much global data defined in file：文件中全局数据太多
Two consecutive dots：两个连续的句点
Type mismatch in parameter xxx：数 xxx 类型不匹配
Type mismatch in redeclaration of ´xxx´：xxx 重定义的类型不匹配
Unable to create output file ´xxx´：无法建立输出文件 xxx
Unable to open include file ´xxx´：无法打开被包含的文件 xxx
Unable to open input file ´xxx´：无法打开输入文件 xxx
Undefined label ´xxx´：没有定义的标号 xxx
Undefined structure ´xxx´：没有定义的结构 xxx
Undefined symbol ´xxx´：没有定义的符号 xxx
Unexpected end of file in comment started on line xxx：从 xxx 行开始的注解尚未结束，文件不能结束
Unexpected end of file in conditional started on line xxx：从 xxx 开始的条件语句尚未结束，文件不能结束
Unknown assemble instruction：未知的汇编结构
Unknown option：未知的操作
Unknown preprocessor directive：´xxx´：不认识的预处理命令 xxx
Unreachable code：不可到达的代码
Unterminated string or character constant：字符串缺少引号
User break：用户强行中断了程序
Void functions may not return a value：Void 类型的函数不应有返回值
Wrong number of arguments：调用函数的参数数目错
´xxx´not an argument：xxx 不是参数
´xxx´not part of structure：xxx 不是结构体的一部分
xxx statement missing (：xxx 语句缺少左括号
xxx statement missing)：xxx 语句缺少右括号
xxx statement missing ;：xxx 缺少分号
´xxx´declared but never used：声明了 xxx 但没有使用
´xxx´is assigned a value which is never used：给 xxx 赋了值但没有使用
Zero length structure：结构体的长度为零

注意：(1) 不同编译环境的功能或提示可能有差异。
(2) 部分说明为“经验性”的，仅供参考。

附录 D 常用头文件及包含的函数

常见的包含库类别与头文件的对应关系如表库类别和头文件所示。

表　库类别和头文件

序　　号	库　类　别	头　文　件
1	输入与输出函数	stdio. h
2	数学函数	math. h
3	字符串处理函数	string. h
4	时间处理函数	time. h
5	字符处理函数	ctype. h
6	实用工具函数	stdlib. h

1. 头文件 stdio.h

输入与输出函数：该分类用于处理包括文件、控制台等各种输入与输出设备，各种函数以“流”的方式实现。

remove：删除文件

rename：修改文件名称

tmpfile：生成临时文件名称

tmpnam：得到临时文件路径

fclose：文件访问，关闭文件

fflush：刷新缓冲区

fopen：打开文件

fprintf：格式输出

fscanf：格式输入

printf：格式输出(控制台)

scanf：格式输入(控制台)

sprintf：格式输出到缓冲区

sscanf：从缓冲区中按格式输入

vfprintf：格式化输出

vprintf：格式化输出
vsprintf：格式化输出
fgetc：输入一个字符
fgets：字符串输入
fputc：字符输出
fputs：字符串输出
getc：字符输入(控制台)
getchar：字符输入(控制台)
gets：字符串输入(控制台)
putc：字符输出(控制台)
putchar：字符输出(控制台)
puts：字符串输出(控制台)
ungetc：字符输出到流的头部
fread：直接流读操作
fwrite：直接流写操作
clearerr：错误清除
feof：文件结尾判断
ferror：文件错误检测
perror：得到错误提示字符串

2. 头文件 math.h

数学函数：该分类给出了各种数学计算函数，必须提醒的是，ANSIC 标准中的数据格式并不符合 IEEE754 标准，但一些 C 语言编译器却遵循 IEEE754 标准(例如 frinklin C51)。

acos：反余弦
asin：反正弦
atan：反正切
atan2：反正切 2
cos：余弦
sin：正弦
tan：正切
cosh：双曲余弦
sinh：双曲正弦
tanh：双曲正切
exp：指数函数
frexp：指数分解函数
fdexp：乘积指数函数
log：自然对数

log10：以 10 为底的对数
modf：浮点数分解函数
pow：幂函数
sqrt：平方根函数
ceil：求下限接近整数
fabs：绝对值
floor：求上限接近整数
fmod：求余数

3. 头文件 string.h

字符串处理函数：该分类的函数用于对字符串进行合并、比较等操作。
memcpy：字符串复制、块复制(目的和源存储区不可重叠)
memmove：块复制(目的和源存储区可重叠)
strcpy：串复制
strncpy：按长度的串复制
strcat：字符串连接函数、串连接
strncat：按长度连接字符串
memcmp：串比较函数，块比较
strcmp：字符串比较
strcoll：字符串比较(用于非英文字符)
strncmp：按长度对字符串比较
strxfrm：字符串转换
memchr：字符与字符串查找，字符查找
strchr：字符查找
strcspn：字符串查找
strpbrk：字符串查找
strspn：字符串查找
strstr：字符串查找
strtok：字符串分解
memset：杂类函数，字符串设置
strerror：错误字符串映射
strlen：求字符串长度

4. 头文件 time.h

日期和时间函数：该类别给出时间和日期处理函数。
clock：得到处理器时间
difftime：得到时间差
mktime：设置时间

time：得到时间

asctime：得到以 ASCII 码表示的时间

ctime：得到字符串表示的时间

strftime：得到指定格式的时间

5. 头文件 ctype.h

字符处理函数：该类别函数用于对单个字符进行处理，包括字符的类别测试和字符的大小写转换。

isalnum：是否字母和数字

isalpha：是否字母

iscntrl：是否控制字符

isdigit：是否数字

isgraph：是否可显示字符(除空格外)

isprint：是否可显示字符(包括空格)

ispunct：是否既不是空格，也不是字母和数字的可显示字符

isspace：是否空格

isupper：是否大写字母

isxdigit：是否十六进制数字(0—9，A—F)字符

toupper：字符大小写转换函数，转换为大写字母

tolower：转换为小写字母

6. 头文件 stdlib.h

实用工具函数：该分类给出了一些无法按以上类别分类，但又是编程所必须的函数。

atoi：字符串转换为整数

atol：字符串转换为长整数

strtod：字符串转换为浮点数

strtol：字符串转换为长整数

strtoul：字符串转换为无符号长整型

rand：产生随机数

srand：设置随机函数的起动数值

pause：屏幕暂停函数

附录 E C 语言 32 个关键字和 9 种控制语句

1. C 语言的关键字

C 语言的关键字共有 32 个，根据关键字的作用，可分为数据类型关键字、控制语句关键字、存储类型关键字和其他关键字四类。

1) 数据类型关键字(12 个)

(1) char：声明字符型变量或函数。

(2) double：声明双精度变量或函数。

(3) enum：声明枚举类型。

(4) float：声明浮点型变量或函数。

(5) int：声明整型变量或函数。

(6) long：声明长整型变量或函数。

(7) short：声明短整型变量或函数。

(8) signed：声明有符号类型变量或函数。

(9) struct：声明结构体变量或函数。

(10) union：声明共用体(联合)数据类型。

(11) unsigned：声明无符号类型变量或函数。

(12) void：声明函数无返回值或无参数，声明无类型指针(基本上就这 3 个作用)。

2) 控制语句关键字(12 个)

A　循环语句

(1) for：一种循环语句(可意会不可言传)。

(2) do：循环语句的循环体。

(3) while：循环语句的循环条件。

(4) break：跳出当前循环。

(5) continue：结束当前循环，开始下一轮循环。

B　条件语句

(1) if：条件语句。

(2) else：条件语句否定分支(与 if 连用)。

(3) goto：无条件跳转语句。

C　开关语句

(1) switch：用于开关语句。

(2) case：开关语句分支。

(3) default：开关语句中的“其他”分支。

D　返回语句

return：子程序返回语句(可以带参数，也可以不带参数)。

3) 存储类型关键字(4个)

(1) auto：声明自动变量，一般不使用。

(2) extern：声明变量是在其他文件中声明(也可以看做是引用变量)。

(3) register：声明寄存器变量。

(4) static：声明静态变量。

4) 其他关键字(4个)

(1) const：声明只读变量。

(2) sizeof：计算数据类型长度。

(3) typedef：用以给数据类型取别名(当然还有其他作用)。

(4) volatile：说明变量在程序执行中可被隐含地改变。

2. C语言控制语句

C语言的控制语句共有9种。

(1) goto语句：无条件转向。

(2) if语句：判断语句。

(3) while语句：循环语句。

(4) do-while语句：先执行循环体，然后判断循环条件是否成立，之后继续循环。

(5) for语句：循环，可替代while语句；只是用法不同。

(6) break语句：跳出本层的循环，(只跳出包含此语句的循环)。

(7) continue语句：继续，一般放到循环语句里，不再执行它下面的语句，直接跳到判断语句。例如for语句就直接跳到第二个分号处，while语句就直接跳到while()的括号里。

(8) switch语句：多项选择。

(9) return语句：返回。

参 考 文 献

1. 谭浩强. C 程序设计[M]. 4 版. 北京：清华大学出版社，2010.
2. 刘汝佳，陈锋. 算法竞赛入门经典——训练指南(算法艺术与信息学竞赛)[M]. 北京：清华大学出版社，2012.
3. 高克宁等. 程序设计基础(C 语言)[M]. 2 版. 北京：清华大学出版社，2013.
4. 俞经善等. ACM 程序设计竞赛基础教程(计算机科学与技术专业实践系列教材)[M]. 北京：清华大学出版社，2010.
5. 俞勇. ACM 国际大学生程序设计竞赛：算法与实现(ACM 国际大学生程序设计竞赛(ACM-ICPC)系列丛书)[M]. 北京：清华大学出版社，2013.
6. 吴永辉，王建德. 算法设计编程实验：大学程序设计课程与竞赛训练教材[M]. 北京：机械工业出版社，2013.
7. Thomas H. Cormen，Charles E. Leiserson，Ronald L. Rivest，Clifford Stein. 算法导论(原书第 3 版)[M]. 殷建平，徐云，王刚等译. 北京：机械工业出版社，2013.
8. Horton，I. C 语言入门经典(第 5 版)[M]. 杨浩译. 北京：清华大学出版社，2013.
9. 明日科技. C 语言从入门到精通[M]. 2 版. 北京：清华大学出版社，2012.
10. 秋叶拓哉，岩田阳一，北川宜稔. 挑战程序设计竞赛[M]. 2 版. 巫泽俊，庄俊元，李津羽译. 北京：人民邮电出版社，2013.
11. 曾棕根. ACM 程序设计[M]. 2 版. 北京：北京大学出版社，2011.
12. 余立功. ACM/ICPC 算法训练教程[M]. 北京：清华大学出版社，2013.

本书特色

本书可以作为“C语言程序设计”基础教材！此外，本书将ACM竞赛平台OJ（Online Judge）系统用于C语言的学习过程中，旨在帮助读者快速掌握C语言的基础知识和ACM/ICPC的基本答题方法。全书内容深入浅出、通俗易懂，可使读者在短时间内掌握编程要领。书中主要实例采用了ACM/ICPC规定的格式进行描述，所有程序均已在DEV C++上调试通过。

配套资源

本书配套资源下载地址为清华大学出版社网站（www.tup.com.cn）本书页面，内容包括书中的全部实例的源代码文件。

清华大学出版社数字出版网站
WQBook 书文局泉
www.wqbook.com

上架指导：计算机/程序设计/C语言
ISBN 978-7-302-40116-2
9 787302 401162 >
定价：35.00元