第 1 章　C 语言基础

掌握程序设计的前提是掌握程序设计语言，在众多程序设计语言中，C 语言以其灵活性和实用性受到了广大计算机应用人员的喜爱。C 语言是一种既得到美国国家标准化协会(ANSI)标准化又得到工业界广泛支持的计算机语言。几乎任何一种机型、任何一种操作系统都支持 C 语言开发。C 语言在巩固其原有应用领域的同时，又在拓展新的应用领域。C 语言支持大型数据库开发和 Internet 应用，一旦掌握了 C 语言，就可以较为轻松地学习其他任何一种程序设计语言，并为后续的面向对象程序设计、Windows 程序设计、Java 程序设计等程序设计语言的学习打下基础。

1.1　程序与程序设计语言

1.1.1　计算机与程序

1946 年，世界上第一台电子计算机问世。随着计算机科学及其应用的迅猛发展，计算机被广泛地应用于人类生产、生活的各个领域，推动了社会的进步与发展。特别是随着国际互联网(Internet)日益深入千家万户，传统的信息收集、传输及交换方式正在发生革命性的改变，计算机已将人类带入了一个新的时代——信息时代。

计算机是一种能快速、高效地对各种信息进行存储和处理的电子设备。它按照人们事先编写的程序对输入的原始数据进行加工处理、存储或传送，以得到预期的输出信息，并利用这些信息来提高社会效率，改善人们的生活质量。

计算机是怎样工作的呢？它又是如何懂得人类语言的呢？计算机是“聪明傻瓜”，不告诉它怎样干它什么也不会干，告诉它怎样干它会干得很好。使计算机明白一个个的命令，它就会按照你的命令去干。这种给计算机的命令，称为指令。指令是计算机要执行的一种基本操作命令，是对计算机进行程序控制的最小单位。指令由操作码和操作数构成，它们分别表示何种操作和存储地址。一连串计算机指令的集合，叫做程序。

所谓程序，就是要计算机完成某一任务所规定的一系列动作或步骤。没有程序和程序设计，计算机将不能做任何事情，即程序(软件)是计算机的必要组成部分。

1.1.2　计算机语言

要使计算机能够很好地为人类服务，人类与计算机之间必须通过一种语言来相互沟通和交流。这种既能够表达人类的思想，又能够被计算机所识别、接受的语言就叫做计算机语言。

计算机程序设计语言的发展，经历了从机器语言、汇编语言、高级语言到面向对象的程序设计语言的历程。

1. 机器语言

机器语言被称为第一代计算机语言。机器语言是最底层的计算机语言，是用二进制代码指令表达的计算机语言。它能被计算机硬件直接识别并执行，由操作码和操作数组成，如指令 00111110，00000111。机器语言程序编写的难度较大且不容易移植，即针对一种计算机编写的机器语言程序不能在另一种计算机上运行。

2. 汇编语言

汇编语言是用助记符代替操作码，用地址符代替操作数的一种面向机器的低级语言，一条汇编指令对应一条机器指令。例如，用“ADD”代表加法，“MOV”代表数据传递等。这样，人们很容易理解程序在做什么，纠错及维护就变得方便了。这种程序设计语言也称为编译语言，即第二代计算机语言。由于汇编语言采用了助记符，它比机器语言易于修改、编写、阅读，但用汇编语言编写的程序（称为汇编语言源程序）机器不能直接执行，必须使用编译程序把它翻译成机器语言即目标程序后，才能被机器理解和执行，这个编译的过程称为汇编。

汇编语言同样依赖于计算机硬件，移植性不好，但效率仍十分高。针对计算机特定硬件而编制的汇编语言程序，能准确发挥计算机硬件的功能和特长，程序精练而质量高。所以，它至今仍是一种常用且强有力的软件开发工具。

汇编语言和机器语言都是面向机器的语言，称为低级语言。

3. 高级语言

从最初与计算机交流的痛苦经历中，人们意识到，应该设计一种语言，这种语言接近于数学语言或人类的自然语言，同时又不依赖于计算机硬件，编写出的程序能在所有机器上通用。经过不断努力，1954 年，第一个完全脱离计算机硬件的高级语言——FORTRAN 语言问世了。50 多年来，共有几百种高级语言出现，其中，有重要意义的有几十种。

高级语言是直接面向过程的程序设计语言，它与具体的计算机硬件无关。用高级语言编写的源程序可以直接运行在不同机型上，因而具有通用性。但是，计算机不能直接识别和运行高级语言程序，必须经过“翻译”。所谓“翻译”，就是由一种特殊程序把源程序转换成目标程序，这种特殊程序被称为语言处理程序。高级语言的翻译方式有两种：一种是“编译方式”，另一种是“解释方式”。编译方式是通过编译程序将整个高级语言源程序翻译成目标程序，再经过连接程序生成可执行程序（.EXE）；解释方式是通过解释程序边解释边执行，不产生目标程序。

最常用的高级语言有 BASIC，C，PASCAL，FORTRAN 等。

20 世纪 60 年代中后期，软件越来越多，规模越来越大，而软件的生产基本上是各自为战，缺乏科学规范的系统规划与测试和评估标准。其后果是大批耗费巨资建立起来的软件系统，由于错误而无法使用，从而带来巨大损失。软件给人的感觉越来越不可靠，几乎没有不出错的软件。这一切，极大地震动了计算机界，史称“软件危机”。人们认识到：大型程序的编制不同于写小程序，它应该是一项新的技术，应该像处理工程一样处理软件研制的全过程。程序的设计应易于保证正确性，也便于验证正确性。1969 年，结构化程序设计方法被提出，1970 年，第一个结构化程序设计语言——PASCAL 语言出现，标志着结构化程序设计时期的开始。

20 世纪 80 年代初开始，在软件设计思想上，又产生了一次革命，其成果就是面向对象的程序设计。

高级语言的下一个发展目标是面向应用，即只需要告诉程序要干什么，程序就能自动生成算法，自动进行处理，这就是非过程化的程序语言。

4. 面向对象的程序设计语言

面向对象的程序设计语言，一般具有可视化、网络化、多媒体等功能。目前比较流行的有 Visual Basic，Visual C++，Java，Delphi，PowerBuilder 等。

1.1.3　高级语言程序的开发过程

程序是为解决某一问题而编写的语句的集合，编写程序的过程就是程序设计。概括地说，程序设计就是分析问题、设计算法、编写程序、调试程序的过程。在最初的程序设计步骤中，把解决问题的过程看成是数据被加工的过程，基于这种方法的程序设计称为面向过程的程序设计。

C 语言是面向过程的结构化程序设计语言，它适合用自顶向下的方法进行软件开发。该方法采用逐步求精的设计过程，主要有 6 个步骤：确定问题、分析问题、设计算法、实现算法、程序的测试与调试和程序的维护。本书的案例分析都采用该方法。

1. 确定问题

确定问题就是要明白程序要解决的问题。一般情况下，一个具体的问题要涉及诸多方面，这是问题的复杂性所在。为了便于解决这个问题，往往需要忽略一些次要的方面，而把注意力集中在问题的实质上。

2. 分析问题

分析问题就是要确定问题中的数据以及数据之间的联系。例如，求解两个数之和的问题，需要输入的数据是两个数，要输出的结果是两个数的和，如何由两个数计算出它们的和，则是算法要解决的问题。

3. 设计算法

设计算法就是要用一种符号或语言来描述解决问题的具体步骤和方法。一般采用自顶向下、逐步求精的设计原则和方法。先从最原始的问题入手，按照功能把问题逐步分解为若干子问题，子问题还可以再分解为若干下一级子问题，直到不能再分解为止。

4. 实现算法

实现算法就是要用计算机语言来实现具体的算法。在这个环节中，把描述算法的每个步骤转变为一种程序设计语言中的一条或多条语句，最后把算法转变为计算机程序。

5. 程序的测试与调试

程序的测试是在程序通过编译、不存在语法错误和连接错误的前提下，找出程序中可能存在的错误并对其进行改正，因此，应该测试程序的运行情况。输入不同的数据可以检测出程序在不同情况下运行的状况，看程序是否完成预期的功能，是否能够顺利通过和实现程序的设计目的。若有错误必须对程序进行修改，直到正确为止。

6. 程序的维护

程序在交付使用之后对其进行的修改称为程序的维护。程序维护的目的是更正程序设计中的漏洞，升级和更新程序。

1.2 C 语言概述

C 语言是国际上广泛流行的、很有发展前途的计算机高级语言，它集高级语言和低级语言的功能于一体。它适合作系统描述语言，既可用来编写系统软件，也可用来开发应用软件。本节介绍 C 语言的发展历程，C 语言的标准及特点。

1.2.1 C 语言的发展

以前的系统软件主要是用汇编语言编写的。由于汇编语言依赖于计算机硬件，程序的可读性和可移植性都比较差。为了提高程序的可读性和可移植性，最好改用高级语言，但一般高级语言难以实现汇编语言的某些功能。人们设想能否找到一种既具有一般高级语言特性，又具有低级语言特性的语言。C 语言就在这种情况下应运而生了。

在 20 世纪 60 年代，BCPL 语言是计算机软件人员在开发系统软件时，作为记述语言使用的一种程序语言。1970 年，美国贝尔实验室的 Ken Thompson 在软件开发工作中，继承和发展了 BCPL 语言的特点，进而提出了“B 语言”。当时最新型的小型计算机，美国 DEC 公司的 PDP-7 型机中的 UNIX 操作系统就是使用 B 语言记述和开发的。但 B 语言过于简单，功能有限。1972 年至 1973 年间，美国贝尔实验室的 Dennis M. Ritchie 在 B 语言的基础上设计出了 C 语言。

早期的 C 语言主要用于 UNIX 系统。后来，C 语言进行了多次改进，但主要还是在贝尔实验室内部使用。直到 1975 年 UNIX 第 6 版公布后，C 语言的突出优点才引起人们的普遍注意。由于 C 语言的强大功能和各方面的优点逐渐为人们所认识，到了 20 世纪 80 年代，C 开始进入其他操作系统，并很快在各类大、中、小和微型计算机上得到广泛的使用。现在 C 语言已经风靡全世界，成为世界上应用最广泛的几种计算机语言之一。

1.2.2 C 语言的标准

1978 年，Brian W. Kernighan 和 Dennis M. Ritchit(合称 K&R)合著了影响深远的名著 *The C Programming Language* 一书。这本书中介绍的 C 语言成为广泛使用的 C 语言版本的基础，它被称为标准 C，又称为 K&R 标准。1983 年，美国国家标准化协会根据 C 语言的各种版本对 C 的发展和扩充，制定了一个新的标准，称为 ANSI C。ANSI C 是 C 语言的标准，任何 C 语言的编译器都在 ANSI C 的基础上扩充。

1990 年，国际标准化组织 ISO(International Organization for Standardization)接受 87 ANSI C 为 ISO C 的标准(ISO 9899:1990)。目前流行的 C 编译系统都是以它为基础的，但不同版本的 C 编译系统所实现的语言功能和语法规则略有差别。本书的叙述基本上以 ANSI C 为基础。

在 ANSI 标准化后，C 语言的标准在相当一段时间内都保持不变，尽管C++ 继续在改进(实际上，Normative Amendment 1 在 1995 年已经开发了一个新的 C 语言版本。但是这个版本很少为人所知)。C 语言标准在 20 世纪 90 年代才真正经历了改进，这就是 ISO 9899:1999(1999 年出版)。这个版本就是通常提及的 C99，它于 2000 年 3 月被 ANSI 采用。

1.2.3　C 语言的特点

C 语言是一种通用的结构化语言。它的通用性和无限制性，使得它对许多程序设计者来说都显得更加通俗和有效。无论是系统软件还是应用软件或者数据处理、非数值计算等，都可以很方便地使用 C 语言。

C 语言的主要特点如下：

(1)简洁、紧凑、灵活。C 语言有丰富的数据类型和运算符，数据结构描述能力及表达式表达能力强。语言组成精练、简洁，语法限制不太严格，而且使用方便灵活。程序书写形式自由，几乎允许任何类型之间的转换。

(2)模块化、结构化。C 语言是以函数为模块来组成程序的，函数实现了程序的模块化。C 语言提供了顺序、选择和循环控制结构，从而实现了程序的结构化。

(3)移植性强。C 语言通过预处理命令等方法，允许程序尽可能地把与计算机硬件有关的部分从程序中分离出来，从而便于在不同的硬件环境和操作系统间实现程序的移植。

(4)C 语言允许直接访问物理地址，能进行位(bit)一级的操作，实现汇编语言的大部分功能，可以直接对硬件进行编程操作。因此，C 既具有高级语言的功能，又具有低级语言的许多功能，可用来编写系统软件。

(5)生成目标代码质量高，程序执行效率高。在代码效率方面可以和汇编语言媲美。

1.3　C 程序的基本结构

1.3.1　一个简单的 C 程序

用 C 语言编写程序非常简单，请看下面的例子。

【例 1-1】 一个简单的 C 程序。

参考程序如下：

```
/* Chap1_1.c:C 程序举例 */
#include<stdio.h>
int main()
{
  printf("Welcome to C class! \n"); /* 显示输出字符串“Welcome to C class!” */
  return 0;
}
```

程序的运行结果为：

```
Welcome to C class!
```

其中，int main()指定了程序的主函数，每一个 C 程序都必须有一个 main 函数。函数名是 main，函数体由一对花括号“{}”括起来。在主函数中，每条语句最后都有一个分号。#include<stdio.h>是一条预处理命令，这条命令在编译之前的预处理阶段执行。符号“/*”和“*/”之间的部分为注释部分。

1.3.2 C 程序的组成

由【例 1-1】可以看到，一个 C 程序通常由函数、语句、注释、预处理命令等几个基本部分组成。

1. 函数

C 程序是由函数构成的。一个 C 源程序有且仅有一个 main 函数，也可以有一个 main 函数和若干其他函数，其他函数都是通过 main 函数直接或间接调用来执行的。因此，函数是 C 程序的基本单位。可以说 C 是函数式的语言，程序中的全部工作都是由各个函数分别完成的。编写 C 程序就是编写一个个函数。C 的函数库十分丰富，ANSI C 提供了 100 多个库函数。

函数必须有自己的名称用来加以识别，但是 main 函数是不能更改名称的。因为编译器在编译程序时，会从 main 函数的位置开始编译，也就是说，main 函数是整个程序的入口，如果没有这个函数，就无法完成编译工作。一般函数的结构如下：

```
类型修饰符 函数名(形式参数表)
{
  函数体
}
```

其中，类型修饰符表示函数的返回值类型；函数体必须包括在左花括号“{”和右花括号“}”中，左花括号表示函数的起始位置，右花括号表示函数的结束位置；系统将通过函数名调用该函数。例如，主函数的结构如下：

```
int main()     /* int 表示该函数返回值为整型,形式参数表为空 */
{
  函数体
}
```

说明：在 C 语言程序中，参数部分写为 void 或空，表示该函数没有参数，只执行一个过程。函数的写法有多种，将在后面的章节详细介绍。

2. 语句

C 语言程序代码中，一条语句可以分写在几行，一行也可以写多条语句，每条语句间用分号隔开。一个函数由多条语句构成，每条语句最后都有一个分号(;)标识。在一个函数中，语句分为声明语句和执行语句。

1)声明语句

```
#define NUM 35          /* 声明一个符号常量 NUM */
int x,y;                /* 声明两个整型变量 x 和 y */
```

2)执行语句

执行语句是在程序执行时发出的某种动作。执行语句由操作和操作的内容组成。如：

```
printf("welcome to C class! \n");
printf("%d,%d\n",x,y);
```

3. 注释

程序的注释在程序的编写过程中是非常重要的工作。一个没有注释的程序就像一个没

有使用说明书的软件。

C 的程序注释方式如下：

- //注释的内容
- /* 注释的内容 */

注释以“//”开头，其后是注释文字，可一直延续到该行行尾。“/* 注释的内容 */”是另一种方式，注释文字夹在“/*”和“*/”之间，这样的注释不但可以出现在行尾，也可以出现在一行中的其他位置，还可以跨越多行。

编译系统不理会注释文字，因此注释文字可以是任意的。注释可使程序更容易理解，不会有副作用，因此在编写程序时，随时添加注释是一种良好的习惯。

4. 编译预处理

预处理命令不是 C 语言的一部分，它只是用来扩充 C 程序设计的环境。常用的预处理命令有两种：

1）#include 命令

#include 命令也称文件包含命令，其作用是将指定的文本文件引入到程序该点处，该文本文件包含了许多函数的定义。例如，之前提到的 printf 函数是 C 语言中实现输出的函数，它的定义就在“stdio. h”这个头文件中，所以在该函数之前，必须先引入#include<stdio. h>。

2）#define 命令

#define 命令用来定义一个符号常量。例如：

```
#define PI 3.1415926
```

这里定义了一个符号常量 PI，PI 在程序中用来代替 3.141 592 6。

注意：编译预处理命令还有很多，它们都是以“#”开头，并且不用分号（;）结束，所以不是 C 程序中的语句。

5. 保留字和标识符

1）保留字

上面例题中的 include，int，return 等都是保留字。所谓保留字，就是 C 语言中已有的具有特殊含义的字符符号。这些保留字不能用于其他目的。

2）标识符

所谓标识符，就是程序设计人员自己定义的表达一定含义的字符符号，如函数名、变量名、常量名等。标识符是用户自己定义的，但也必须遵循以下规则：

（1）标识符的第一个字符必须是字母或下划线，后面可以由字母、下划线或数字组成。

（2）C 语言规定不能使用保留字作为标识符，例如，不能将标识符命名为 int，float，main 等。但是标识符中可以包含保留字。

（3）符号遵循见名知义原则，即一看到标识符就清楚它所表达的含义。

注意：在 C 语言标识符中是区分大小写的。例如，变量 SUM 和 sum 代表两个不同的变量。

【例 1-2】 认识 C 程序的组成。

参考程序如下：

```
/* Chap1_2.c：计算两个数的和 */
#include <stdio.h>
int main( )
```

```
{
  int a,b,sum;     /* 声明 3 个变量的类型 */
  a=1;
  b=2;
  sum=a+b;
  printf(" a add b is: %d \n",sum);
  return 0;
}
```

程序的运行结果为：

```
 a add b is:3
```

在本程序中，只有一个主函数 main，函数体由 6 条语句构成，其中，“int a,b,sum;”为声明语句，其他的都是执行语句。“printf()”是格式输出函数，它是 C 语言中的库函数，使用该函数之前，需要在程序的开头包含定义它的头文件，即“#include <stdio.h>”。

1.4 运行 C 程序的步骤和方法

1.4.1 C 程序的编译环境介绍

本书采用的编译环境是中文版 Visual C++ 6.0。

Visual C++ 6.0 是美国微软公司研制开发的C++语言版本，它是一个集C++程序的编辑、编译、调试、运行和在线帮助等功能及可视化软件开发功能于一体的软件开发工具，或称开发环境、开发系统等。本节对其作简单介绍，目的是让读者掌握编辑、编译和运行一个C++控制台应用程序(console application program)的简要过程。

启动 Visual C++ 6.0 的常用方法有：

(1)双击 Windows 操作系统桌面上的 Microsoft Visual C++ 6.0 图标，即可打开 Visual C++ 6.0 集成开发环境，其操作界面如图 1-1 所示。

图 1-1 Visual C++ 6.0 界面

(2)打开“开始”菜单,执行“程序”→Microsoft Visual C++ 6.0→Microsoft Visual C++ 6.0 命令,即打开 Visual C++ 6.0 集成开发环境。

1.4.2　运行一个简单的 C 语言程序

下面简单介绍在 Visual C++ 6.0 中如何运行一个 C 语言程序。主要步骤有创建工程、新建源文件、编辑源文件、运行程序等。

1.创建工程

使用 Visual C++ 6.0 创建工程的具体步骤如下:

(1)启动 Visual C++ 6.0 后,执行“文件”→“新建”命令,弹出“新建”对话框。

(2)在“新建”对话框中,单击“工程”选项卡,在左边的列表框中选择 Win32 Console Application 选项,然后在右侧的“位置”文本框中输入保存的位置(也可单击文本框右侧的按钮,选择保存位置),再在“工程名称”文本框中输入工程名,如 proj11,如图 1-2 所示。

图 1-2　“新建”对话框

(3)单击“确定”按钮,弹出如图 1-3 所示的“Win32 Console Application-步骤 1 共 1 步”对话框,采用默认选项,单击“完成”按钮。

图 1-3　建立控制台应用程序的第 1 步

(4)这时,弹出如图 1-4 所示的“新建工程信息”对话框,这里显示新建工程类型、目录等信息。

图 1-4 “新建工程信息”对话框

(5)单击“确定”按钮。这时已新建一个工程,工程名为 proj11,位置在“E:\XIANSHI” 下。

2. 新建源文件

新建源文件的步骤如下:

(1)执行菜单栏“文件”→“新建”命令,弹出“新建”对话框,切换到“文件”选项卡,如图 1-5所示。

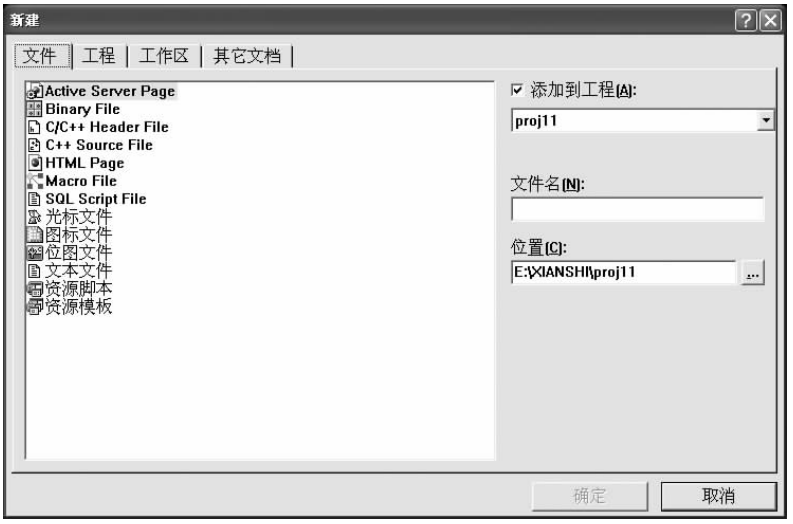


图 1-5 “文件”选项卡

(2)在“文件”选项卡的列表框中,选择C++ Source File 文件类型选项,在“文件名”文本框中,输入文件名“例 1-2. c”,如图 1-6 所示。这时,“确定”按钮由灰色变为可用的黑色,单击此按钮,则可新建一个 C 源文件。

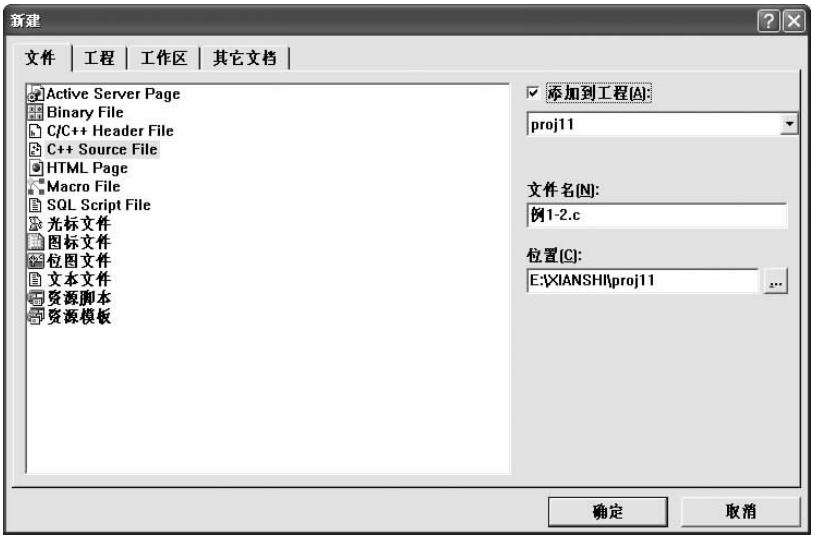


图 1-6　输入文件名

注意:在默认情况下,文件名的扩展名为.cpp,所以此处需要在“文件名”文本框中输入文件的类型为.c。

(3)单击“确定”按钮,关闭该对话框,回到 Visual C++ 6.0 集成开发环境界面。

3. 编辑源文件

在图 1-7 所示的源文件编辑界面中输入【例 1-2】中的源程序代码。

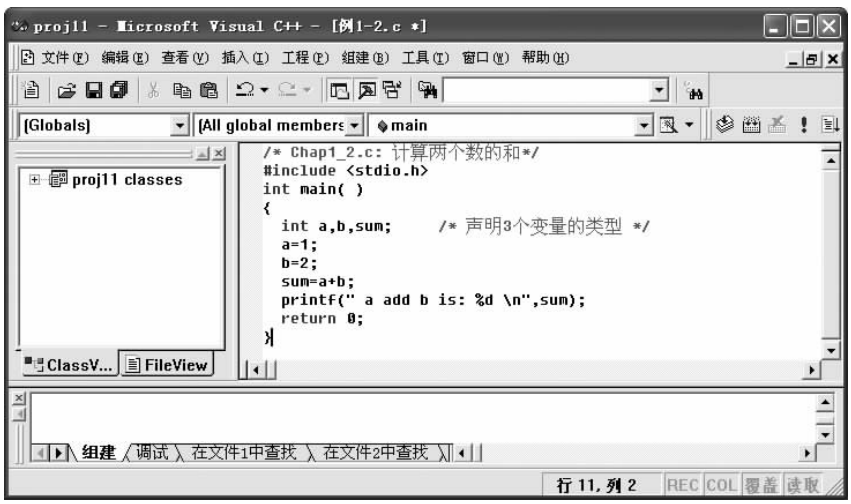


图 1-7　源文件编辑界面

4. 编译、连接、运行程序

1)编译程序

当输入和编辑好一个程序文件后,运行该程序文件之前要先进行编译。执行“组建”→“编译”命令,即可编译在编辑窗口中打开的源程序文件,生成一个扩展名为.obj 的目标文件。通常应首先编译程序主文件,然后再编译其他程序文件。

若在编译过程中检查出语法错误，则在状态输出窗口显示出产生错误的程序行行号和错误原因，以便用户重新回到编辑窗口修改错误。

编译时检查出的错误包含两类：一类为严重错误(error)，又称为致命错误，用户必须修改它，否则不能进一步向下处理；另一类为警告错误(warning)，它不影响进入下一步处理过程，但最好把它修改掉，使得程序在编译后不产生任何错误。

当编译完一个程序文件后，将在状态输出窗口显示出各类错误的个数，若不出现任何错误，则显示出“0 error(s)，0 warning(s)”信息，表示没有错误。

2)连接程序

连接程序文件就是将一个程序中的主目标文件与其他目标文件和相关的库函数文件连接起来形成一个可执行的文件。

具体连接操作是：执行“组建”→“组建”命令即可。若连接过程没有发现任何错误，则表示连接成功，此时在状态输出窗口显示出“0 error(s)，0 warning(s)”信息，若连接过程中发现有错误，则将在状态输出窗口显示出发生错误的文件、所在的行号和出错原因。用户应根据这些信息修改有关程序文件中的错误，然后再重新进行编译和连接。

3)运行程序

运行程序就是运行经程序编译和连接而生成的可执行文件。具体操作是：执行 “组建”→“执行”命令，会弹出如图 1-8 所示提示是否建立文件的对话框，单击“是”按钮，就会出现如图 1-9 所示的程序运行结果界面。根据提示信息，用户按下任一键后将关闭该窗口，重新回到 Visual C++ 6.0 界面。

图 1-8 提示是否建立文件的对话框

图 1-9 程序运行结果界面

需要注意的是，当一个程序运行结束后，要在 Visual C++ 6.0 主窗口（即 Visual C++ 6.0 集成操作界面）中，执行“文件”→“关闭工作空间”命令，关闭该项目的工作区；否则，再新建源程序时，一个程序中会出现两个 main 函数，这是不允许的。

Visual C++ 6.0 主窗口中的每个菜单都含有丰富的菜单项，读者可以在实践中逐渐了解和掌握。

说明：本书的程序均是在 Visual C++ 6.0 环境下编译并运行的。C 程序的编译环境很多，常用的还有 Turbo C++ 3.0，它的使用可参照附录 E。

本 章 小 结

本章首先简要介绍了程序的概念以及计算机程序设计语言的发展、C 语言的发展与特点，之后通过简单而典型的 C 语言程序实例，介绍了 C 程序的组成，最后介绍了在 Visual C++ 6.0 环境下运行 C 程序的基本方法与步骤。

习　题　1

1. 为什么说 C 语言是一种通用的计算机语言？它的主要特点是什么？
2. C 语言源程序由哪几部分组成？其组成结构有什么特点？
3. 简述运行 C 程序的步骤。
4. 编写一个 C 程序，输出以下信息：

```
**********
我是一名大学生！
我喜爱 C 语言！
**********
```

5. 编写一个计算 a＝b＋10 的程序，在程序前加上注释并运行。
6. 上机运行本章【例 1-2】中的程序。

第 2 章　C 语言中的数据

数据处理是计算机的主要功能。数据处理最终是通过计算机程序来实现的，数据是程序必要的组成部分，是程序中被处理的对象。凡数据都必须有类型；变量在使用之前必须先进行类型说明。这是两条必须遵循的规则。本章重点介绍 C 语言程序能够处理的基本数据类型、数据处理的基本步骤以及数据输入输出函数。

2.1　概　　述

2.1.1　数据及数据处理

数据用来描述一个具体事物或问题的静态特征。例如，求出 3 个数的和及平均值。该问题中涉及的数据有 5 个，分别是已知的 3 个数、这 3 个数的和及平均值。输入的数据是 3 个数；输出的数据是这 3 个数的和及平均值。由 3 个已知数到计算并输出和及平均值的过程就是数据处理的过程，即解决问题的过程。

2.1.2　数据处理的步骤

数据处理的基本步骤是：数据的输入、数据处理、数据的输出。

解决前面提到的求 3 个数的和及平均值问题的具体步骤：数据的输入就是 3 个数的输入过程；数据处理就是由 3 个数计算并输出和及平均值的过程；数据的输出就是将计算的结果输出显示到屏幕上的过程。

2.2　数 据 类 型

数据是程序处理的对象。程序设计的过程就是数据加工和处理的过程。C 语言规定，程序中使用的每个数据都必须属于某种数据类型。数据类型是对程序所处理数据的一种“抽象”，通过类型名对数据赋予一些约束，以便进行高效处理和检查。这些约束包括以下几个方面：

(1)取值范围。每种数据类型对应不同的取值范围，即数据类型是数值的一个集合。

(2)存储空间大小。每种数据类型对应不同规格的字节空间。

(3)运算方式。数据类型是一个数据集合以及其上所带某种操作的集合。

C 语言提供了丰富的数据类型，主要有基本数据类型和导出数据类型，具体如图 2-1 所示。

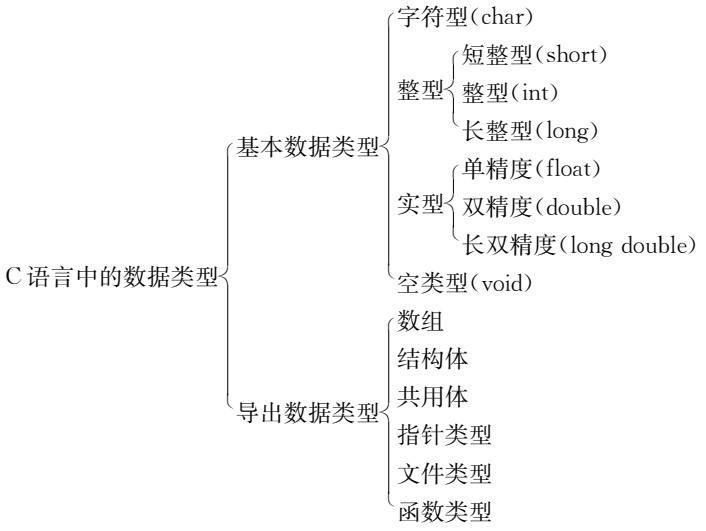


图 2-1　数据类型

这种数据分类是从数据的存在形式上划分的，如果从变化的角度划分，C 语言中的数据还可分为常量和变量两类。

2.3　常　　量

常量是指在程序运行过程中，其值不能被改变的量。常量根据类型的不同，可分为整型常量、实型常量、字符型常量、字符串常量、符号常量等。

1. 整型常量

在 C 语言中，整型常量可用以下 3 种形式表示：

(1)十进制整数。这是最常用的一种形式，如 512，－16。

(2)八进制整数。以数字 0 开头的数是八进制数。例如，0116 表示八进制整数，相当于十进制整数 78；－012 表示八进制整数，相当于十进制整数－10。

(3)十六进制整数。以 0x 开头的数是十六进制数。例如，0xA5 表示十六进制整数，相当于十进制整数 165；－0x123 表示十六进制整数－123，相当于十进制整数－291。

下面是非法的整型常量：

- 0819：非十进制整数，又非八进制整数，因为八进制数只能包含 0～7 的 8 个数字。
- 20fa：非十进制整数，又非十六进制整数。
- 0x36g：非十六进制整数，因为包含非法字符 g。

需要说明的是，在没有任何特殊标志的情况下，可以根据常量所在的范围来确定其类型。例如，在 16 位机器中，当一个常量值在十进制整数范围－32 768～32 767 内，则被看做一个 short int 型整数或 int 型整数。超出了上述范围的整型常量，则被看做长整型。

有时，可以在常量的后面加上字母标识符来表示该常量是长整型或者无符号整型。例如：

- 128L：表示十进制 long int 型整数。
- －12LL：表示十进制 long long int 型整数。

• 0x4efb2L:表示十六进制 long int 型整数。

• 40000U:表示十进制 unsigned int 型整数。

后缀字母标识符可以是大写,也可以是小写。

在 16 位字长的机器中,一旦把一个常量表示为 long int 型,系统便将其存储空间扩充为 4 个字节。从值的大小上看,128L 和 128 之间没有区别,但它们所占用的存储空间不同。

2. 实型常量

实型常量又称为浮点数或实数。C 语言中,实数只有十进制形式,它有两种表示方式:一种是定点数形式,另一种是浮点数形式。

(1)定点数形式,即凡不带指数部分的数均称为定点数。例如,3.141 59,−1.298,10.,2 009.0等。整数都属于定点数。

(2)浮点数形式,又称为指数形式。在计算机内部,实数都以浮点数形式存储。例如,0.003 141 59在计算机中内部表示为 3.141 59e−3;其中,指数部分是 e−3,用 10 的幂表示,3.141 59 是小数部分。又如,−0.001 23 可表示为−1.23e−3。

注意:用浮点数形式表示实数,字母 e(或 E)之前的小数部分中,小数点左边应有且只有一位非零的数字,e 后面的指数必须是整数。例如,e−6,7.2e+2.5 都是不合法的实数。

C 语言将实型数据分为 float,double,long double 三种类型,且默认实型数据是 double 类型的。因此,对于实型常量,C 语言编译器会将它视为 double 类型。如果特别说明某实型常量是 float 类型或 long double 类型,可以加上后缀字母表示。

(1)用 f 或 F 表示 float 类型,如 123.5f,1.234 5e+3F。

(2)用 l 或 L 表示 long double 类型,如 123.5l,1.234 5e+3L。

3. 字符型常量

字符型常量是用一对单引号引起来的单个字符,如'A'、'*'、'8'等都是合法的字符型常量。

C 语言中的一个字符型常量占据一个字节的存储空间,在该字节中存放的并不是字符本身,而是该字符的 ASCII 码值。如字符'A'的 ASCII 码值为 65,'0'的 ASCII 码值为 48。由于字符型常量存储的是一个整数,因此它可以像整数一样参与数值运算。在 C 语言程序中,字符型常量通常用于字符之间的比较。

C 语言还允许一种特殊形式的字符型常量,就是以反斜杠“\”开头的转义字符序列。转义字符序列用来表示 ASCII 码字符集内的控制代码。例如,前面用“\n”表示换行,“\n”实际上是一个字符,它的 ASCII 码值为 10。转义字符常用于表示 ASCII 字符集内的控制字符和某些用于功能定义的字符,如单引号、双引号和反斜杠等。

常用的转义字符见表 2-1。

表 2-1 转义字符表

字符形式	功 能
\n	换行
\t	横向跳格(即跳到下一个输出区)
\v	竖向跳格
\b	退格
\r	回车

续表

字符形式	功　　能
\f	换页
\\	反斜杠字符“\”
\′	单引号字符
\ddd	1～3 位八进制数所代表的字符
\xhh	1～2 位十六进制数所代表的字符

表 2-1 中列出的字符称为“转义字符”，意思是将反斜杠(\)后面的字符转换成另外的意义。如′\n′中的“n”不代表字母 n 而作为“换行”符。

′\ddd′是用 ASCII(八进制数)表示一个字符。例如，′\101′代表 ASCII 码(十进制)为 65 的字符′A′。原因是八进制的 101 转换成十进制就是 65，ASCII 码值为 65 的字符是′A′。

′\xhh′是用 ASCII(十六进制)表示一个字符。例如，′\x42′是用两位十六进制数所代表的字符，十六进制的 42 转换成十进制就是 66，ASCII 码值为 66 的字符是′B′。

4. 字符串常量

字符串常量是用双引号括起来的零个、一个或多个字符序列，如″12345″，″张三″等。双引号仅作为定界符使用，它不属于字符串本身。

C 语言中，字符串在内存中存储时，系统自动在字符串的末尾加 ASCII 码值为 0 的字符，用以表示该字符串的结束。因此，长度为 n 个字节的字符串，实际上占据了 n+1 个字节的存储空间。例如，字符串常量″12345″有 5 个字节，却占据了 6 个字节的存储空间，其存储形式如图 2-2 所示。

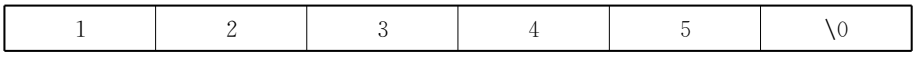


图 2-2　字符串常量

实际上，图 2-2 中的字符应当用对应的 ASCII 码表示，这里为了方便，直接用字符表示。另外，″A″和′A′除了形式上有区别外，在内存中的存储形式也是不同的，如图 2-3 所示。

(a)字符′A′的存储表示　　(b)字符串″A″的存储表示

图 2-3　字符′A′和字符串″A″的存储

5. 符号常量

C 语言中，常量也可以用一个标识符来命名，这就是符号常量。为了与一般变量相区分，符号常量习惯上用大写字母表示。符号常量在使用之前必须先定义，其定义的一般形式如下：

```
#define 符号常量名 常量值
```

例如：

```
#define PI 3.1415926
```

定义了一个符号常量 PI，这样凡在程序中出现 PI，都表示 3.141 592 6。PI 是一个常量，在程序中只能被引用，不能被修改。

用符号常量代替常量，使程序更清晰易懂，同时使程序便于修改，保证了对常量修改的一致性。

注意：定义符号常量的＃define 行不能以分号结束。

【例 2-1】 符号常量的使用。

参考程序如下：

```
/* Chap2_1.c:符号常量的使用 */
#include<stdio.h>
#define NUM 35
int main()
{
  float p,total;
  scanf("%f",&p);
  total=p*NUM;
  printf("%f\n",total);
  return 0;
}
```

程序运行结果如下：

```
6↙
210.000000
```

说明：程序中用＃define 命令行定义 NUM 代表常量 35。此后凡在本程序中出现的 NUM 都代表 35，可以和常量一样地使用。

需要注意的是，符号常量不同于变量，它的值在其作用域内不能改变，也不能被重新赋值。如再赋值给 NUM（如 NUM=45；）是错误的。

2.4 变　　量

变量是在程序的运行过程中其值可以改变的量。一个变量应该有一个名字，应该在内存中占据一定的存储单元，并在该存储单元中存放变量的值。请注意区分变量名和变量值这两个不同的概念，如图 2-4 所示。

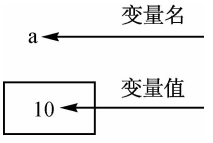


图 2-4　存储单元

变量名也称变量的标识符，它实际上是一个符号地址，在对程序编译连接时由系统给每个变量分配一个内存地址。程序从变量中取值时，实际上是通过变量名找到相应的内存地址，从其存储单元中读取数据。

变量的命名要符合标识符的规定，通常使用小写字母。在选择变量名时应“见名知义”，即选有含义的英文单词（或单词的缩写）作标识符，如 num，count，total 等。除了数值程序外，一般不用单个符号（如 a，b，n 等）作为变量名，以增加程序的可读性。这是结构化程序的一个特征。本书在一些简单的举例中，为方便起见，采用了单个字符作为变量名，请读者在其他程序中不要都如此命名。

2.4.1　变量的定义

在 C 语言中，要求对所有使用的变量作强制定义，也就是“先定义，后使用”，即所有的变量在使用之前都必须加以说明。

变量定义的一般形式如下：

类型标识符 变量名表；

例如：

- int a,b,c; /* 定义了 3 个整型变量 */
- char ch1; /* 定义了一个字符型变量 */
- float sum; /* 定义了一个实型变量 */

在变量的使用过程中，应注意以下两点：

(1)每个变量被声明为一种数据类型，编译时系统会根据确定的类型为变量分配相应的存储单元。

(2)变量名表可以是一个或多个变量名，中间用逗号分隔。

2.4.2　变量的分类

变量可分为整型变量、实型变量和字符型变量。

1. 整型变量

1)整型变量的分类

整型变量的基本类型符为 int。根据数值的范围可以将整型变量分为基本整型、短整型、长整型。根据这 3 种整型的内部表示的最高位的不同又可分为有符号整型和无符号整型。

(1)基本整型以 int 表示，如 98，－125，30 000 等。

(2)短整型以 short int 或 short 表示。

(3)长整型以 long int 或 long 表示。

以上 3 类均表示有符号的整型。为了充分利用变量的数值范围，对以上 3 类都加上修饰符 unsigned，则指定为无符号数；如果加上修饰符 signed，则指定为有符号数。如果既不指定 unsigned，也不指定 signed，则默认为有符号数。

无符号型的整数必须是正整数或零。

2)整型变量的存储

整型变量数据在内存中是以二进制形式存放的。

各种数据类型所占的内存因机器而异。在 PC 机上，各种整型数据所占位数和数的范围如表 2-2 所示。

表 2-2　各种整型数据所占位数和数的范围

数据类型	所占位数	数的范围
int	16	$-2^{15}\sim(2^{15}-1)$
short	16	$-2^{15}\sim(2^{15}-1)$
long	32	$-2^{31}\sim(2^{31}-1)$
unsigned	16	$0\sim(2^{16}-1)$
unsigned short	16	$0\sim(2^{16}-1)$
unsigned long	32	$0\sim(2^{32}-1)$

需要说明的是，在内存中存储整数时，一般以其最高位（即最左边的一位）表示符号，以 0 表示正，以 1 表示负。数值是以补码形式存放的。一个正数的补码就是该数的二进制数。下面将介绍求一个负数的补码的方法。

例如，－10 的补码的求解过程如下：

(1)取该数的绝对值：　　10

(2)以二进制数的形式表示：　　10 的二进制数码位 00000000,00001010

(3)对其按位求反：　　按位求反所得为 11111111,11110101

(4)加 1：　　加 1 即得 11111111,11110110

即：－10 的 16 位存储形式为 11111111,11110110。

所有负数的最高位必然是 1。从该位的状态可以判断一个数的正或负，所以这样的整数称为有符号的整数。在 C 语言中，也可以使用无符号的整数，以它来表示只有正值的数值（如人数，年龄等）。如图 2-5 所示为同样长度的内存单元存储数据时，有符号整数和无符号整数之间取值范围的比较。

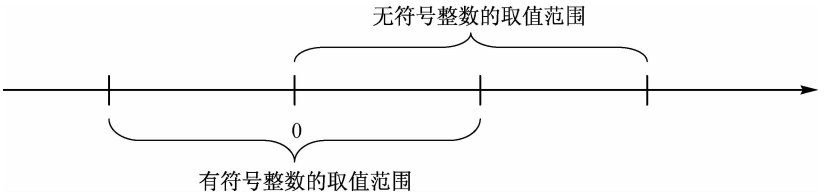


图 2-5　有符号整数和无符号整数之间取值范围的比较

显然，无符号整数的最大值约为有符号整数的最大值的两倍再加 1（因为是补码表示）。例如，一个有符号的两个字节整数的最大值为 32 767，而一个无符号两个字节整数的表示范围为 0～65 535。

在 C 语言中，有符号整数用 signed 修饰，无符号整数用 unsigned 修饰，并且有符号整数的定义可以将符号修饰符缺省。例如：

```
signed int a,b;              /* a 和 b 是有符号整数 */
int c,d;                     /* c 和 d 是有符号整数 */
unsigned int e,f;            /* e 和 f 是无符号整数 */
```

3)整型变量的定义

例如：

```
int num;                     /* 定义整型变量 num */
unsigned int a,b;            /* 定义两个无符号整型变量 a 和 b */
```

4)整型变量举例

【例 2-2】 整型变量的定义与使用。

参考程序如下：

```
/*Chap2_2.c:整型变量的定义与使用*/
#include<stdio.h>
int main()
{
  int a,b,c,d;
```

```
    unsigned u;
    a=12;
    b=-36;
    u=10;
    c=a+u;
    d=b+u;
    printf("a+u= %d,b+u= %d\n",c,d);
    return 0;
}
```

程序运行结果如下：

```
a+u=22,b+u=-26
```

说明：不同种类的整型数据可以进行算术运算。

2. 实型变量

1)实型变量的分类

根据数值的范围可分为单精度(float)、双精度(double)和长双精度(long double)3 种类型。

PC 机上实型数据所占位数及数的范围如表 2-3 所示。

表 2-3　各种实型数据所占位数和数的范围

数据类型	有效数字	所占位数	数的范围
float	6～7	32	-3.4×10^{-38}～3.4×10^{38}
double	15～16	64	-1.7×10^{-308}～1.7×10^{308}
long double	18～19	128	$-1.2\times10^{-4\,932}$～$1.2\times10^{4\,932}$

不同的编译系统，有效数字的位数是有区别的。

注意：在 C 语言中，实数都是有符号的，不可以使用无符号修饰符。

C 语言提供了一个测试某种数据类型所占存储空间大小的运算符“sizeof”，其格式为：

```
sizeof(类型标识符或数据)
```

例如：

```
sizeof(int)
```

当不了解所用编译系统中的某种数据类型的宽度时，可以使用这个运算符进行计算。

2)实型数据的存储

一个实型数据一般在内存中占 4 个字节(32 位)。与整型数据的存储方式不同，实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。实数 3.141 59 在内存中的存放形式可以用图 2-6 表示。

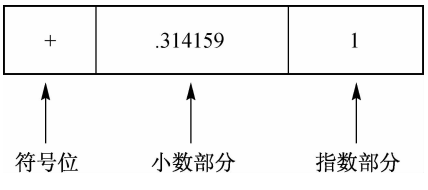


图 2-6　实数 3.141 59 在内存中的存放形式

图 2-6 中是用十进制数来示意的，实际上在计算机中是用二进制数来表示小数部分以及用 2 的幂次来表示指数部分的。

在 4 个字节中，究竟用多少位来表示小数部分，多少位来表示指数部分，ANSI C 标准并无具体规定，由各 C 编译系统自定。不少编译系统以 24 位表示小数部分(包括符号位)，以 8 位表示指数部分(包括指数的符号)。小数部分占的位数愈多，数的有效数字愈多，精度愈高。指数部分占的位数愈多，则能表示的数值范围愈大。

3)实型变量的定义

例如：

```
• float x,y;        /* 定义 x 和 y 为单精度变量 */
• double z;         /* 定义 z 为双精度变量 */
```

3. 字符型变量

1)字符型变量的定义

定义字符型变量的一般形式如下：

```
char ch1,ch2;
```

表示定义了 ch1 和 ch2 为字符型变量，各能存储一个字符型常量。可以用下面语句对 ch1 和 ch2 赋值：

```
ch1='a';ch2='b';
```

说明：一个字符型变量只能存储一个字符型常量。

2)字符型变量的存储

将一个字符型常量放到一个字符型变量中，并不是把该字符本身放到内存单元中去，而是将该字符的 ASCII 码放到存储单元中。

在 C 语言中，字符型数据和整型数据之间可以通用，即一个字符型数据既可以以字符形式输出，也可以以整数形式输出。以字符形式输出时，需要先将存储单元中的 ASCII 码转换成相应的字符，然后输出。以整数形式输出时，直接将 ASCII 码作为整数输出。字符型数据也可以进行算术运算，此时是对它们的 ASCII 码进行算术运算。例如：

```
ch2=ch1+1;
```

2.4.3 变量的初始化

所谓变量的初始化，也就是在定义变量的同时对变量赋予初值。可以给其中的一个赋值，也可以同时给多个变量赋值。例如：

```
• int sum=0;
• char ch1='a',ch2='b';
• int a,b,c=10;
```

需要说明的是，变量的初始化不是在编译阶段完成的(除了静态存储变量和外部变量)，而是在程序运行时才完成的。

【例 2-3】 变量的初始化与使用。

参考程序如下：

```
/*Chap2_3.c:变量的初始化*/
#include<stdio.h>
```

```
int main()
{
  int a,b,c=10;
  char ch1='A',ch2='a';
  printf("%d\n",c);
  printf("%c,%c\n",ch1,ch2);
  return 0;
}
```

程序运行结果如下：

```
10
A,a
```

本程序中，“int a,b,c=10;”表示定义了 3 个整型变量，但只给变量 c 赋初值 10。如果要同时给 3 个变量赋值，可以采用如下形式：

```
int a,b,c;
a=b=c=10;
```

2.5　数据的输出

数据的输出是指将内存中的数据显示到外部输出设备（显示器、打印机、磁盘等）中。

C 语言本身没有输入/输出语句，数据的输入/输出是由函数来实现的。在 C 标准函数库中提供了一些输入/输出函数，它是以标准的输入/输出设备为输入/输出对象的，其中有 printf 函数、scanf 函数、putchar 函数、getchar 函数等。本节主要介绍实现数据输出的 printf 函数和 putchar 函数。

2.5.1　printf 函数

格式输出函数 printf 的作用是向终端输出一个或多个任意类型的数据。

1. printf 函数的一般格式

```
printf(格式控制,输出表列);
```

例如：

```
printf("%d,%c\n",a,ch1);
```

说明：printf 函数在圆括号内包括格式控制和输出表列两部分。

1）格式控制

“格式控制”是用双引号引起来的格式字符串，也称“转换控制字符串”。它包括两种信息：

（1）格式说明符，如“%d”，“%f”等，它的作用是将要输出的数据转换成指定的格式输出。格式说明符总是由“%”字符开始的。

（2）普通字符，即需要原样输出的字符，主要作用是解释、说明。例如，上面 printf 函数示例中一对双引号内的逗号和换行符即为普通字符。

2)输出表列

“输出表列”是需要输出的一个或多个数据,可以是表达式。

在上面的 printf 函数示例中,a 和 ch1 分别是两个不同输出项。多个输出项之间须用逗号来分隔。

【例 2-4】 在屏幕上分别输出 25,3.141 59,A,“张三”这 4 个数据。

参考程序如下:

```
/* Chap2_4.c:printf 函数的使用 */
#include<stdio.h>
int main()
{
  printf("%d\n",25);
  printf("%f\n",3.14159);
  printf("%c,%s\n",'A',"张三");
  return 0;
}
```

程序运行结果如下:

```
25
3.141590
A, 张三
```

在该程序中,printf 函数被称为格式输出函数,即用 printf 函数把数据按照指定的格式输出到显示器上。printf 函数是 C 语言系统提供的函数,在头文件 stdio.h 中定义,所以要用该函数必须包含此文件,格式为:

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

2.格式字符

对于不同类型的数据采用不同的格式字符。主要格式字符及其说明如表 2-4 所示。

表 2-4 printf 函数中常用的格式字符

格式字符	说　明
d	以十进制数形式输出带符号整数
o	以八进制数形式输出带符号整数
x	以十六进制数形式输出带符号整数
f	以小数形式输出单精度、双精度实数
c	输出单个字符
s	输出字符串

【例 2-5】 整型数据的输出。

参考程序如下:

```
/* Chap2_5.c:格式字符 d 的应用 */
#include<stdio.h>
int main()
```

```
{
  int a=12;
  long b=2456988;
  printf("a= %d,a= %6d,a= %-6d,a= %06d\n",a,a,a,a);
  printf("b= %ld\n",b);
  printf(" %d, %o, %x\n",a,a,a);
  printf("\n");
  return 0;
}
```

程序运行结果如下：

```
a=12,a=□□□□12,a=12□□□□,a=000012
b=2456988
12,14,c
```

在该程序中，对“%d”，“%6d”等说明如下：

- %d：表示按整数的实际长度输出。
- %6d：表示该数据的输出占 6 位，如果不足 6 位左边补以空格。
- %-6d：表示以左对齐方式输出，如果不足 6 位右边补以空格。
- %06d：表示该数据的输出占 6 位，如果不足 6 位左边出现空格的地方用 0 来代替。
- %ld：表示输出长整型数据。

【例 2-6】 实型数据的输出。

参考程序如下：

```
/* Chap2_6.c: 格式字符 f 和 e 的应用 */
#include<stdio.h>
int main()
{
  float x=123.456;
  double y=123.45678;
  printf(" %f, %f\n",x,y);     /* %f 的精度缺省值是 6 */
  printf(" %8.2f, %10.3f\n",x,y);
  printf(" %e\n",x);
  return 0;
}
```

程序运行结果如下：

```
123.456001,123.456780
□□123.46,□□□123.457
1.234560e+002
```

对于该程序，说明以下 3 点：

(1)在第 1 个输出结果中，虽然 x 的小数点后有 6 位数字，但因为 x 是单精度数，只有 7 位有效数字，故最后 3 位是无效的数字。

(2)“printf(″%8.2f,%10.3f\n″,x,y);”表示将 x 的值保留 2 位小数，并且该数据在输

出时占 8 位;将 y 的值保留 3 位小数,并且该数据在输出时占 10 位。

(3)“%e”格式表示将实型数据按指数形式输出,这是系统默认的格式,数字部分的小数位数占 6 位,指数占 4 位。数值按规范化指数形式(小数点前必须有而且只有 1 位非零数字)输出。

【例 2-7】 字符数据的输出。

参考程序如下:

```
/* Chap2_7.c:格式字符 c 的应用 */
#include<stdio.h>
int main()
{
  char ch1='a';
  int i=97;
  printf("%c,%d\n",ch1,ch1);
  printf("%3c,%6d\n",ch1,ch1);
  printf("%c,%d\n",i,i);
  return 0;
}
```

程序运行结果如下:

```
a,97
□□a, □□□□97
a,97
```

在程序中,“printf("%3c,%6d\n",ch1,ch1);”的含义是:第 1 个变量 ch1 输出占 3 位,第 2 个变量 ch1 输出占 6 位。

【例 2-8】 字符串的输出。

参考程序如下:

```
/* Chap2_8.c: 格式字符 s 的应用 */
#include<stdio.h>
int main()
{
  printf("%3s\n","CHINA");
  printf("%7.2s\n","CHINA");
  printf("%.4s\n","CHINA");
  printf("%-5.3s\n","CHINA");
  return 0;
}
```

程序运行结果如下:

```
CHINA
□□□□□CH
CHIN
CHI□□
```

对于格式字符“s”的使用，说明以下几点：

(1)%ms：输出指定宽度的字符串，若 m 大于字符串的实际长度，则左端补以空格；否则按实际长度输出。

(2)%－ms：如果 m 大于字符串的实际长度，则输出的字符串在 m 列范围内左对齐，右端补以空格。

(3)%m.ns：输出占 m 列，但只截取字符串左端 n 个字符，这 n 个字符输出在 m 列的右端，左端补以空格。如果 n＞m，则 m 自动等于 n 值，即保证了 n 个字符正常输出。

(4)%－m.ns：m，n 的含义同上，区别在于 n 个字符输出在 m 列的左端，右端补以空格。

2.5.2　putchar 函数

字符输出函数 putchar 的作用是向终端输出一个字符。其一般格式为：

```
putchar(ch);
```

参数：ch 是函数的参数，它可以是字符型变量，也可以是字符型常量。

功能：表示它输出字符型变量的值或字符型常量。

【例 2-9】　字符数据的输出。

参考程序如下：

```
/* Chap2_9.c:putchar 函数的应用 */
#include<stdio.h>
int main()
{
  char ch1='a';
  int i=97;
  putchar(ch1);
  putchar('\n');
  putchar(i);
  putchar('\n');
  return 0;
}
```

程序运行结果如下：

```
a
a
```

说明：putchar 函数是 C 语言中的字符输出函数，使用该函数需要包含定义它的头文件“stdio.h”。putchar('\n')输出一个换行符。putchar 函数中的参数可以是字符型变量，也可以是整型变量。

2.6　数据的输入

数据的输入是通过输入设备(键盘、鼠标等)把数据送入计算机内存中。

2.6.1 scanf 函数

1. scanf 函数的一般格式

scanf(格式控制,地址表列);

参数说明:

- 格式控制:含义与 printf 函数相同,但不能用来显示非格式字符串,即不能显示提示字符串。
- 地址表列:是由若干地址组成的表列。可以是变量的地址,也可以是字符串的首地址。其作用是按指定的格式用键盘把数据输入到变量中。在 C 语言中,变量的地址由取地址运算符“&”加变量名组成,如变量 a 的地址,可写成“&a”。

功能:scanf 函数称为格式输入函数,其作用是从标准输入设备(通常指键盘)读入数据,按照指定的格式把它们送到相应的数据存储地址中。

【例 2-10】 用 scanf 函数输入数据,用 printf 函数输出数据。

参考程序如下:

```
/* Chap2_10.c: 用 scanf 为变量提供数据 */
#include<stdio.h>
int main()
{
  int a;
  float b;
  scanf("%d",&a);
  printf("a=%d\n",a);
  scanf("%f",&b);
  printf("b=%f\n",b);
  printf("b=%.2f\n",b);
  return 0;
}
```

程序运行结果如下:

```
10↙
a=10
3.1415↙
b=3.141500
b=3.14
```

说明:scanf 函数的作用是从键盘为不同类型的变量提供值。scanf 函数是一个标准库函数,它是在头文件 stdio.h 中定义的。

对于初学者来说,应当注意:scanf 函数中的地址表列应当是变量的地址,即由取地址运算符“&”后跟变量组成而不是变量名。这种错误一般在编译时是检查不出来的,但当程序

执行时就会出现问题。

【例 2-11】 scanf 函数的格式说明。

参考程序如下：

```
/* Chap2_11.c：scanf 函数的格式说明*/
#include<stdio.h>
int main()
{
  int a,b,c;
  scanf("%d%d%d",&a,&b,&c);
  printf("%d,%d,%d\n",a,b,c);
  return 0;
}
```

程序运行结果如下：

```
3□4□5↙
3,4,5
```

在该程序中，“scanf("%d%d%d",&a,&b,&c);”的作用是通过键盘输入 3 个数据依次存放在 3 个不同的存储单元中。输入数据时，两个数据之间可以用一个或多个空格间隔，也可以使用 Enter 键、Tab 键。下面输入均为合法：

```
3□□4□□□□5↙
```

或

```
3↙
4↙
5↙
```

如果在“格式控制”字符串中除了格式说明符以外还有其他字符，则在输入数据时应输入与这些字符相同的字符。例如：

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

输入数据时，应采用如下的形式：

```
3,4,5↙
```

注意：在输入的数据之间要用逗号，它和 scanf 函数中的“格式控制”字符串中的逗号是对应的。如果输入数据时不用逗号而用空格或其他字符则是不对的。

2. 格式字符

scanf 函数与 printf 函数中的格式说明符相似，其格式说明符以“%”开头，以一个格式字符结束，中间可以插入附加字符。

scanf 函数中常用的格式字符如表 2-5 所示。scanf 函数中可以用的附加格式字符（修饰符）如表 2-6 所示。

表 2-5 scanf 函数中常用的格式字符

格式字符	说　明
d	用于输入十进制整数
o	用于输入八进制整数
x	用于输入十六进制整数
c	用于输入单个字符
s	用于输入字符串
f	用于输入单精度数据
e	与 f 作用相同

表 2-6 scanf 函数中的附加格式字符

附加格式字符	说　明
l	用于输入长整型数据(可用%ld,%lo,%lx)以及 double 型数据(%lf 或%le)
h	用于输入短整型数据(可用%hd,%ho,%hx)
域宽	用于指定输入数据所占宽度(列数),域宽为正整数
*	表示本输入项在读入后不赋给相应的变量

【例 2-12】 scanf 函数的使用。

参考程序如下：

```
/* Chap2_12.c: scanf 函数中格式字符的应用 */
#include<stdio.h>
int main()
{
  int a,b,c;
  scanf("%3d%2d%3d",&a,&b,&c);
  printf("%d,%d,%d\n",a,b,c);
  scanf("%2d %*3d %2d",&a,&b,&c);
  printf("%d,%d,%d\n",a,b,c);
  return 0;
}
```

程序运行结果如下：

```
12345678↙
123,45,678
12□345□67↙
12,67,678
```

在该程序中,对于语句“scanf("%3d%2d%3d",&a,&b,&c);”输入：

```
12345678↙
```

系统会自动按它截取所需数据,将 123 赋值给 a,45 赋值给 b,678 赋值给 c。

对于语句“scanf("%2d %*3d %2d",&a,&b,&c);”输入：

```
12□345□67↙
```

系统将 12 赋值给 a,“%*3d”表示读入了 3 位整数但不赋值给任何变量,然后再读入 2 位整数 67 赋值给 b。

3.注意事项

(1)scanf 函数中的“地址表列”应当是变量地址序列,而不应是变量名。例如:

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

(2)用“%c”输入字符时,空格字符、转义字符和跳格符(Tab)都作为有效字符输入。例如:

```
scanf("%c  %c  %c",&c1,&c2,&c3);
```

如输入:

a□b□c↙

结果是将字符 a 送给 c1,空格字符送给 c2,字符 b 送给 c3。由于“%c”只要求读入字符,所以字符之间不需要用空格作为分隔。正确的输入如下:

abc↙

(3)输入数据时不能规定精度。例如,以下用法是错误的:

```
scanf("%7.2f",&a);
```

2.6.2　getchar 函数

getchar()函数是字符输入函数,其功能是等待从键盘输入一个字符,返回它的值并在屏幕上自动回显该字符,即从终端输入一个字符。getchar 函数没有参数,其一般形式如下:

```
getchar();
```

函数的值就是从输入设备得到的字符。

【例 2-13】 getchar 函数的功能。

参考程序如下:

```
/* Chap2_13.c: getchar 函数的应用 */
#include<stdio.h>
int main()
{
  char ch;
  ch=getchar();
  putchar(ch);
  printf("\n");
  return 0;
}
```

程序运行结果如下:

```
A↙
A
```

运行时若从键盘输入字符“A”,并按回车键,就会在屏幕上看到输出的字符“A”。

需要注意的是,getchar 函数只能接收一个字符。getchar 函数得到的字符可以赋给一

个字符变量或整型变量，也可以不赋给任何变量而作为表达式的一部分。例如，上例的输入、输出可用下面一行代替：

```
putchar(getchar());
```

2.7 运算符与表达式

C 语言提供了丰富的运算符，如算术运算符、关系运算符、逻辑运算符、位运算符等。用运算符和括号等，将运算对象连接起来的、符合 C 语法规则的式子，称为表达式。

2.7.1 赋值运算

1. 赋值运算符

在 C 语言中，赋值也被认为是一种运算。赋值符号“=”就是赋值运算符，它的作用是将赋值运算符右边表达式的值赋给左边的变量。其形式如下：

变量=表达式；

例如：

```
x=a+8;
```

式中，“=”是赋值运算符，表示将 a+8 的值赋给变量 x。

注意：赋值运算符的左边只能是变量。

2. 赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为赋值表达式。

例如：

```
a=(b=10);
```

括号中的 b=10 是一个赋值表达式，它的值是 10，因此 a=(b=10)，相当于 a=10。

赋值运算符是按“自右向左”的顺序结合，因此，上面的式子可写成：

```
a=b=10;
```

需要注意的是，赋值运算符“=”与数学中的“相等”概念是不同的，它表示把表达式的值送到变量所代表的存储单元中。因此，赋值运算符的左边只能是变量，而不能是常量或表达式，如 5=a+b，a+b=c 等都是不合法的。

【例 2-14】 观察赋值运算符的应用，分析程序的运行结果。

参考程序如下：

```
/* Chap2_14.c：赋值运算符的应用 */
#include<stdio.h>
int main()
{
  int a,b;
  a=10;                    /* 把常量 10 赋值给变量 a */
  b=5;
  printf("%d,%d\n",a,b);
```

```
    a=a+b;                    /* 把表达式 a+b 的值赋值给变量 a */
    b=a-b;
    a=a-b;
    printf("a= %d,b= %d\n",a,b);
    return 0;
}
```

程序运行结果如下：

```
10,5
a=5,b=10
```

2.7.2　算术运算

1.算术运算符

算术运算符包括以下几种：

- +：加法运算符或正值运算符，如 5+3，+10。
- -：减法运算符或负值运算符，如 5-3，-6。
- *：乘法运算符，如 5*3。
- /：除法运算符，如 5/3。
- %：模运算符，又称求余运算符，如 5%3。

需要注意的是，符号“*”表示乘，不能用数学上习惯用的“×”或“·”表示乘。例如，a*b不能写成 ab，a×b，a·b 等形式。

【例 2-15】 求两个数的商和余数。

参考程序如下：

```
/* Chap2_15.c:算术运算符的应用 */
#include<stdio.h>
int main()
{
    int a,b;
    a=7;
    b=4;
    printf("商为：%d\n",a/b);
    printf("余数为：%d\n",a%b);
    return 0;
}
```

程序运行结果如下：

```
商为:1
余数为:3
```

对于该程序，需要说明以下两点：

(1)求余运算符要求参与运算的两个操作数必须为整型。

(2)本程序中 7/4 的结果是 1，而不是 1.75。原因是两个整型数据的商为整型数据；若

是两个操作数为实型，则结果为实型。

2. 算术表达式

算术表达式是由算术运算符和圆括号将操作数连接起来的表达式。操作数包括常量、变量和函数等。最简单的表达式是一个常量或一个变量。作为一般情况，一个表达式可以有多个运算符。例如：

- a+b*c
- -a/(b+c)-10%7*'a'

3. 算术运算符的优先级和结合性

1)运算符的优先级

在求表达式的值时，运算有先后，这种运算的先后次序称为运算符的优先级。例如，乘(*)、除(/)、取余(%)运算符的优先级高于加(+)、减(-)运算符的优先级；正负号运算符的优先级高于乘(*)、除(/)运算符的优先级。

C 语言中算术运算符的优先级由高到低排列如下：

圆括号→正负号(+,-)→乘除余(*,/,%)→加减(+,-)

2)运算符的结合性

C 语言还规定了运算符的结合性，所谓结合性是指当一个运算对象两侧运算符的优先级相同时，进行运算的结合方向。算术运算符的结合方向是“自左至右”，即运算对象先与左边运算符结合。

常用运算符的优先级和结合性，具体见附录 A。

了解了算术运算符的优先级和结合性，可以看出上述表达式-a/(b+c)-10%7*'a'的运算过程：

① 求(b+c)的值。

② 求-a 的值。

③ 求①/②的值。

④ 求 10%7 的值。

⑤ 求④*'a'的值。

⑥ 求③-⑤的值。

3)圆括号可以改变运算的处理顺序

由于圆括号的优先级最高，因此圆括号的运算总是最先进行的。例如：

(a+b)*c

先计算 a+b 的值，然后再计算出乘以 c 的值。

【例 2-16】 分析程序的运行结果，理解运算符的优先级。

参考程序如下：

```
/*Chap2_16.c:运算符的优先级*/
#include<stdio.h>
int main()
{
  float x=2.5,y=4.1,z;
  int a=10;
```

```
    z=x+a%3*(x+y)/4;
    printf("%.2f\n",z);
    return 0;
}
```

程序运行结果如下：

```
4.15
```

本程序中用到的运算符分别为加(+)、取余(%)、乘(*)、圆括号以及除(/)，因为圆括号的优先级最高，所以要先计算 x+y 的结果 6.6，而%，*，/这 3 个运算符的优先级相同，按左结合，所以 a%3 的结果为 1，再乘以 6.6，然后再除以 4，最后加上 2.5，整个表达式的值为 4.15。

【例 2-17】 求 3 个数的平均值。

参考程序如下：

```
/* Chap2_17.c：求 3 个数的平均值 */
#include<stdio.h>
int main()
{
    int a,b,c,sum;
    float average;
    scanf("%d,%d,%d",&a,&b,&c);
    sum=a+b+c;
    average=sum/3;
    printf("平均值：%.2f\n",average);
    return 0;
}
```

程序运行结果如下：

```
12,35,26↙
平均值:24.00
```

本程序中的语句“printf(″平均值：%.2f\n″,average);”的作用是将变量 average 的值按照格式说明符%.2f 的格式控制输出，其中格式说明符%.2f 表示将实数保留两位小数。

2.7.3　自增、自减运算符

自增运算符(++)和自减运算符(--)是 C 语言特有的两个运算符。这两个运算符都只有一个运算对象，它们的作用是将变量的值增 1 或减 1 后，将运算结果再赋给该变量，具体见表 2-7。

表 2-7　自增(自减)运算符

运 算 符	名　　称	运算功能
++	自增运算符	++i,i++
--	自减运算符	--i,i--

自增(自减)运算符可用在变量的前面,如++i;也可以用在变量的后面,如 i++。尽管++i 和 i++都表示变量 i 的值增 1,但它们的功能是有区别的:++i(--i)是在使用变量 i 之前,使变量 i 的值加(减)1;i++(i --)是在使用变量 i 之后,再使变量 i 的值加(减)1。

【例 2-18】 自增运算符的使用。

参考程序如下:

```
/* Chap2_18.c:自增运算符的使用 */
#include<stdio.h>
int main()
{
  int i=3,j=3;
  printf("%d\n",++i);
  printf("%d\n",i);
  printf("%d\n",j++);
  printf("%d\n",j);
  return 0;
}
```

程序运行结果如下:

```
4
4
3
4
```

关于自增运算符和自减运算符的使用,需要注意以下几点:

(1)自增运算符和自减运算符只能用于变量,而不能用于常量或表达式,如 6++或(a+b)++都是错误的。

(2)++和--的结合方向是“自右向左”,也称右结合性。例如,-a++,a 的左面是负号运算符,右面是自增运算符,则按右结合性,即它相当于-(a++)。

(3)自增或自减操作运行速度比其等价的赋值运算快得多,目标代码的效率也更高。因此,在条件允许的情况下,若有 a=a+1 和 a++两个选择,那么首先选择的是 a++。

合理使用自增(减)运算符,对于编写高质量的 C 程序是很有用的,它们常用于数组下标的变化和循环语句中使循环变量自动加(减)1,也用于指针变量,使指针变量指向下一个地址。这些内容将在以后的章节中学习。

2.7.4 复合赋值运算符

C 语言的赋值运算符,除了基本的赋值符“=”外,为了简化程序和提高编译效率,可以在“赋值符”之前加上其他运算符,以构成复合赋值运算符,具体见表 2-8。

表 2-8　复合赋值运算符

运 算 符	名　　称	运算功能
＋＝	加赋值	a＋＝b 等价于 a＝a＋b
－＝	减赋值	a－＝b 等价于 a＝a－b
＊＝	乘赋值	a＊＝b 等价于 a＝a＊b
/＝	除赋值	a/＝b 等价于 a＝a/b
%＝	取余赋值	a%＝b 等价于 a＝a%b

参与复合赋值运算的两个运算量，先进行相应的运算，然后将其运算结果赋给第一个运算量。

下面是复合运算符的应用的例子：

- a＋＝5 等价于 a＝a＋5。
- a＊＝b＋5 等价于 a＝a＊(b＋5)。
- a/＝a＋a 等价于 a＝a/(a＋a)。
- a%＝5 等价于 a＝a%5。

【例 2-19】 复合运算符的使用。

参考程序如下：

```
/*Chap2_19.c:复合运算符的使用*/
#include<stdio.h>
int main()
{
  int a=2,b=1,c=1;
  int d=6,e=9;
  b+=a;printf("b:%2d\n",b);
  c*=a;printf("c:%2d\n",c);
  d/=a;printf("d:%2d\n",d);
  e%=a;printf("e:%d\n",e);
  return 0;
}
```

程序运行结果如下：

```
b:□3
c:□2
d:□3
e:1
```

2.7.5　逗号运算符和逗号表达式

1. 逗号运算符

C 语言还提供了一种特殊的运算符——逗号运算符(,)，用它将两个表达式连接起来。

例如：

3+5,6+9

2. 逗号表达式

逗号表达式是由逗号运算符连接表达式构成的。逗号表达式的一般形式为：

表达式 1,表达式 2

逗号表达式的求解过程是：先求解表达式 1，再求解表达式 2。整个逗号表达式的值是表达式 2 的值。例如，上面逗号表达式“3+5,6+9”的值为 15。

逗号表达式的一般形式可扩展为：

表达式 1,表达式 2,表达式 3,…,表达式 n

它的值为最后一个表达式 n 的值。

逗号运算符是所有运算符中级别最低的。例如，“a=3 * 5,a * 4;”整个式子构成一个逗号表达式，先将 3 * 5 的值赋值给 a，然后求解 a * 4，得到 60，整个逗号表达式的值是 60，而不是 15。注意下面表达式作用是不同的：

- x=(a=3,6 * 3)　　/ * x 的值为整个逗号表达式的值 18 * /
- x=a=3,6 * 3　　/ * x 的值为 3，整个逗号表达式的值 18 * /

逗号表达式中的表达式可以是字符型数据、算术表达式或赋值表达式等。

逗号表达式无非是将若干表达式“串联”起来。在很多情况下，使用逗号表达式只是想从左到右顺序求出各个表达式的值，而并非一定得到整个逗号表达式的值，因此，它又称为“顺序求值运算符”。逗号表达式最常用于循环语句中。

【例 2-20】 运算符的综合运用。

参考程序如下：

```
/* Chap2_20.c:运算符的综合运用 */
#include<stdio.h>
int main()
{
  int a=12,n=5;
  int x,y=7,z=4;
  a*=2+3;
  printf("%d\n",a);
  a=15;
  a%=(n%=3);
  printf("%d\n",a);
  a=12;
  a/=a+a;
  printf("%d\n",a);
  x=(y=y+6,y/z);
  printf("x=%d\n",x);
  printf("y=%d\n",y);
  return 0;
}
```

程序运行结果如下：

```
60
1
0
x=3
y=13
```

在本程序中，要注意运算符的优先级，其中逗号运算符的优先级低于赋值运算符。在“x=(y=y+6,y/z);”语句中，赋值号右侧的表达式是由逗号运算符构成的表达式，先要将 y+6 的值赋值给 y，再计算 y/z 的值，最后把 y/z 的值赋值给变量 x。

2.8　不同数据类型间的转换

如果一个运算符两侧的运算量的类型不同，先要将它们转换成相同的类型，然后才能进行运算。不同类型的数据在进行赋值和混合运算时都需要进行类型转换。这种类型转换有两种方式：一种是隐式转换，一种是显式转换。

2.8.1　数据类型的隐式转换

1. 赋值转换

当赋值运算符两边的类型不同时，C 语言允许通过赋值运算符，使赋值号右边表达式值的类型自动转换成其左边变量的类型。赋值转换具有强制性。

当将实型数据(包括单、双精度)赋给整型变量时，舍弃实数的小数部分。例如，i 是整型变量，执行 i=5.68 的结果是 i 的值为 5。

当将整型数据赋给实型变量时，数值不变，但以实数形式存放到整型变量中。

2. 输出转换

如一个长整型数在格式输出函数 printf()中指定用“%d”格式输出，相当于先将长整型数转换成整型再输出。

2.8.2　数据类型的显式转换

C 语言提供了一种“强制类型转换”运算符，将一种类型的变量强制转换成另一种类型。例如，(int)3.5 中 int 的作用是将实型数据 3.5 转换成整型。

强制类型转换是一种显式转换，其一般形式为：

(类型标识符) 表达式

其作用是把表达式的值转换为类型名指定的类型。

例如：

```
• (double) a                    /* 把 a 转换成 double 类型 */
• (char)(8-3.14*x)              /* 得到字符型数据 */
• (float)(x=99)                 /* 得到单精度数据 */
• k=(int)((int)x+(float)i+j)    /* 得到整型数据 */
```

显式转换实际上是一种单目运算。各种数据类型的标识符都可以作为显式转换运算符,但必须用圆括号把类型标识符括起来,而不能写成 int(3.56)的形式。

需要注意的是,无论是隐式转换还是显式转换,仅仅是对变量或表达式的类型进行临时性的转换,并未改变原来变量或表达式的类型。例如,若 x 原来为实型变量且其值为 3.5,在执行"i=(int)x;"后得到了一个整数 3,并把它赋值给整型变量 i,但 x 仍为实型,其值为 3.5。

【例 2-21】 数据类型的转换。

参考程序如下:

```
/*Chap2_21.c:数据类型的转换*/
#include<stdio.h>
int main()
{
  int i;
  float x=69.87;
  float y;
  i=(int)x;                    /* 数据类型的显式转换*/
  printf("i=%d\n",i);
  printf("x=%f\n",x);
  y=56;                        /*数据类型的隐式转换*/
  printf("y=%f\n",y);
  return 0;
}
```

程序运行结果如下:

```
i=69
x=69.870003
y=56.000000
```

本 章 小 结

本章主要学习了 C 语言中的基本数据类型、输入/输出函数的格式与应用、运算符与表达式以及数据类型的转换。C 语言中的基本数据类型包括整型、实型和字符型。有些数据在程序中保持不变,这种数据称为常量;有些数据在程序中可以改变,这就是变量,变量在使用之前必须进行类型说明。C 语言中没有输入/输出语句,程序的输入/输出功能是由输入/输出函数来实现的。scanf 和 printf 是格式化输入/输出函数,是将数据按照指定的格式输入/输出。getchar 和 putchar 函数是字符输入/输出函数。C 语言的运算符包括的范围很广,本章主要介绍了赋值运算符、算术运算符、自增(自减)运算符、复合赋值运算符、逗号运算符,并简单介绍了赋值表达式、算术表达式和逗号表达式。

习　题　2

1. C 语言中基本数据类型包括哪些?
2. 什么是常量? 什么是变量? 如何定义一个字符型变量?
3. 分析程序的运行结果。

```
#include<stdio.h>
int main()
{
  int x=1,y=2;
  printf("%d+%d=%d\n",x,y,x+y);
  printf("10 Squared is:%d\n",10*10);
  return 0;
}
```

4. 写出程序的运行结果。

```
#include<stdio.h>
int main()
{
  char c='A';
  printf("c:dec=%d,oct=%o,hex=%x,ASCII=%c\n",c,c,c,c);
  return 0;
}
```

5. 写出下列表达式的值。

(1)10/3*9=_____。
(2)10+16%3=_____。
(3)设 a=7,x=3.5,y=4.7,则 x+a%3*(int)(x+y)%24=_____。
(4)设 x=5,y=10,则表达式++x+y--的值为_____。
(5)表达式(a=3*5,a*4),a+5 的值为_____。
(6)若 x 为 int 型变量,则执行以下语句后的 x 值为_____。

```
x=7;x+=x--=x+x;
```

(7)若已定义 x 和 y 为 double 类型,且 x=1,则表达式 y=x+3/2 的值为_____。
(8)若 x=5,则表达式 x*=2+4 的值为_____。

6. 编写一个 C 程序,要求输出本学期的课程表。

参考格式如下:

	星期一	星期二	星期三	星期四	星期五
1-2	英语	高数	C 语言	英语	C 语言
3-4	上机	英语	高数	美术	高数
5-6	体育	美术	音乐	自习	自习

第 3 章　结构化程序设计

通过前面的学习，我们已经掌握了程序设计中用到的基本数据类型，数据是程序处理的对象，构成程序的基本组成部分。人们使用计算机，就是要利用计算机处理各种不同的问题，而要做到这一点，就必须首先对各类问题进行分析，确定解决问题的具体方法和步骤，再编制好一组让计算机执行的指令即程序，交给计算机，让计算机按人们指定的步骤有效地工作。这个过程就是程序设计。C 语言是一门结构化的程序设计语言。本章将介绍结构化程序设计的基本思想和基本结构以及实现基本结构的语句。

3.1　算　　法

3.1.1　算法概述

在日常生活中做任何一件事情，都是按照一定规则，一步一步进行。例如，在工厂中生产一部机器，先把零件按一道道工序进行加工，然后再把各种零件按一定法则组装成一部完整的机器，这些工艺流程就是算法；在农村种庄稼，有耕地、播种、育苗、施肥、中耕、收割等各个环节，这些耕种技术也是算法。

1. 算法的概念

众所周知，计算机可以完成许多极其复杂的工作，但实质上，这些工作都是按照人们事先编制好的程序进行的。

一个程序应包括以下两个方面的内容：

(1)对数据的描述。在程序中要指定数据的类型和数据的组合形式，即数据结构(data structure)。

(2)对数据的操作。即操作步骤，也就是算法(algorithm)。

作为程序设计人员，必须认真考虑和设计数据结构和算法。因此，著名计算机科学家 Nikiklaus Wirth 提出了一个公式：

数据结构＋算法＝程序

所谓算法，就是计算机解决某一个问题的具体方法和步骤，即算法是解决“做什么”和“怎么做”的问题。程序中的操作语句，实际上就是算法的体现。算法是程序设计的灵魂，数据结构是加工和处理的对象。

计算机用于解决数值计算，如科学计算中的数值积分、解线性方程等的计算方法，就是数值计算的算法；用于解决非数值计算，如用于管理、文字处理、图像图形等的排序、分类、查找，就是非数值计算的算法。

2. 算法的特性

简单地说，算法就是进行操作的方法和步骤。例如，菜谱实质上就是做菜肴的算法，乐

谱实际上是演奏的算法。计算机程序是用某种程序设计语言所描述的解题算法。通常，一个算法应该具有以下 5 个重要的特征：

1)有穷性

一个算法应包含有限的操作步骤，而不能是无限的。

2)确定性

算法中操作步骤的顺序和每一步的内容都应当是确定的，不应当是含糊不清的，即不能具有“二义性”。

3)有零个或多个输入

所谓输入，是指在执行算法时需要从外界取得必要的信息。例如，求两个数的和，需要输入两个数，再计算它们的和。一个算法也可以没有输入。例如，计算 5!。

4)有一个或多个输出

算法的目的是求解，输出就可以得到结果。一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。

5)有效性

算法中每一个步骤都应当能有效地执行，并得到确定的结果。例如，若 b＝0，则 a/b 是不能有效执行的。

3.1.2　算法的描述

描述算法有多种不同的工具，采用不同的算法描述工具对算法的质量有很大的影响。描述一个算法可以采用自然语言、计算机程序设计语言、流程图、N-S 图、伪代码语言等。其中，自然语言就是人们日常使用的语言，可以是汉语、英语或其他语言。用自然语言表示通俗易懂，但文字冗长，容易出现“歧义性”。自然语言表示的含义往往不太严格，要根据上下文才能判断其正确含义。例如，对于以下这句话：如果 A 大于 B，就给它加 1。在理解时就可能出现，是给 A 加 1 还是给 B 加 1？因此，除了很简单的问题以外，一般不用自然语言描述算法。

下面介绍程序设计中常用的两种算法描述工具：一是流程图，二是 N-S 图。

1. 流程图

流程图是一种流传很广的算法描述工具，它是一种用规定的图形、指向线及文字说明来准确表示算法的图形，具有直观、形象的特点，能清楚地展现算法的逻辑结构，而且它独立于任何一种程序设计语言。美国国家标准化协会(ANSI)规定了一些常用的流程图符号，已为世界各国程序工作者普遍采用。

常用的流程图符号如图 3-1 所示。

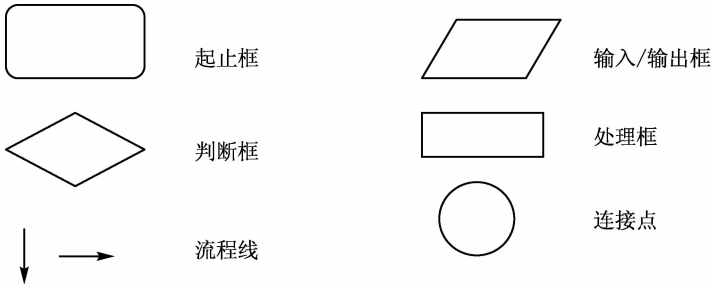


图 3-1　常用的流程图符号

图 3-1 中的菱形框的作用是对一个给定的条件进行判断，根据给定的条件是否成立决定如何执行其后的操作。它有一个入口和两个出口。

利用流程图进行程序设计是一种基本方法，对于复杂问题可以先将其分成一个个模块，即大事化小，然后再对每个模块进行设计，逐步求精，对一个个具体的模块又可使用流程图来设计。

流程图的作用就像写文章先列出一个提纲一样，对程序的结构作全局性的安排，先做什么，后做什么，将程序的先后次序、执行步骤用框图直观清晰地表示出来。

一个流程图包括以下几个部分：

(1)表示相应操作的框。

(2)带箭头的流程线。

(3)框内外必要的文字说明。

例如，求两个数中的较大数，其流程图如图 3-2 所示。

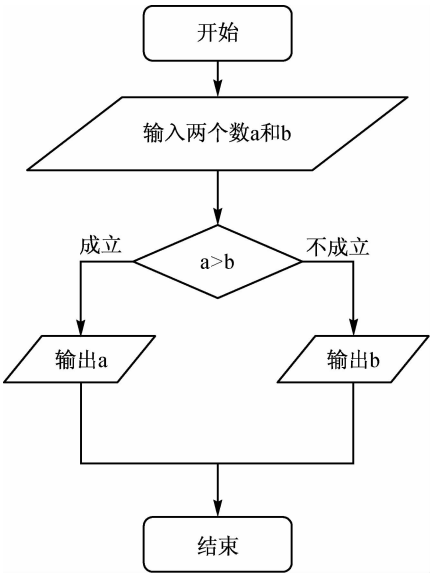


图 3-2　求两个数中的较大数的流程图

画程序流程图的规则：使用标准的流程图符号；流程图一般按从上到下，从左到右的方向画；除判断框外，大多数程序流程图的符号只有一个进入点和一个退出点，而判断框是具有超过一个退出点的唯一符号。

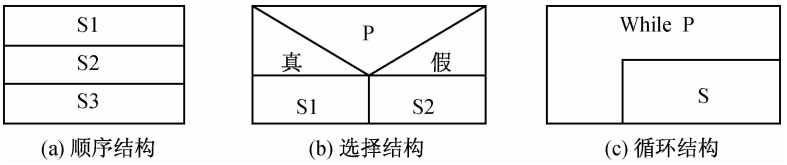
2. N-S 图

灵活的流程线是程序中隐藏错误的祸根。针对这一弊病，1973 年，美国学者 I. Nassi 和 B. Shneiderman 提出了一种新的流程图形式，称为 N-S 图。这种流程图，完全去掉了带箭头的流程线。全部算法写在一个矩形框内，在该框内还可以包含其他的从属于它的框，或者说，由一些基本的框组成一个大的框。这种流程图适合于结构化程序设计，因而很受欢迎。

用 N-S 图描述三种基本结构，如图 3-3 所示。

N-S 图的每一种基本结构都是一个矩形框，整个算法可以像搭积木一样堆成。其中，图 3-3(a)所示为三个操作所组成的顺序结构；图 3-3(b)所示为二分支的选择结构；图 3-3(c)所

示为 while 循环结构，即当命题 P 为“真”时，就重复执行 S。



(a) 顺序结构　(b) 选择结构　(c) 循环结构

图 3-3　用 N-S 图描述三种基本结构

N-S 图避免了流程图在描述程序逻辑时的随意性和灵活性。

用 N-S 图描述求两个数中的较大数算法。具体如图 3-4 所示。

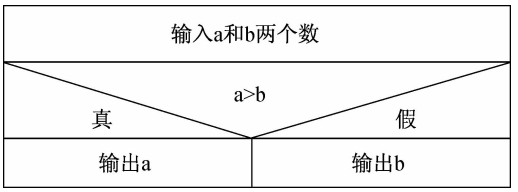


图 3-4　求两个数中的较大数算法的 N-S 图

3.1.3　算法实现的过程

要完成一项工作，包括设计算法和实现算法两个部分。例如，作曲家创作一首乐谱就是设计一个算法，但它仅仅是一个乐谱，并未变成音乐。而作曲家的目的是希望人们听到悦耳动听的音乐。当演奏家按照乐谱的规定进行演奏时，就是“实现算法”。一个菜谱是一个算法，厨师炒菜就是实现这个算法。设计算法的目的是为了实现算法。因此，不仅要考虑如何设计一个算法，还要考虑如何实现一个算法。

用计算机来解决问题，即用计算机实现算法。计算机是无法识别流程图的。只有用计算机语言编写的程序才能被计算机执行(当然，还要经过编译生成目标程序才能被计算机识别和执行)。因此，在用流程图描述一个算法后，还要将它转变成计算机语言程序。

3.1.4　C 语言程序设计的基本步骤

1. 一般程序设计的要求

一个好的程序应该满足设计要求，除了能够正常运行和实现预定的功能以外，还应满足以下方面：

(1)程序要结构化、简明、易读和易调试。

(2)执行速度快。

(3)占用存储空间少。

2. C 语言程序设计的基本步骤

(1)分析问题，抽象出描述问题的数学模型。

(2)确定解决问题的算法。

(3)绘制出程序流程图或 N-S 图。

(4)编写程序。

(5)检查和调试程序。

3.2 顺序结构程序设计

C 语言是一种结构化程序设计语言,结构化程序由三种基本结构组成,即顺序结构、选择结构和循环结构。

顺序结构是这三种基本结构中最简单的一种。在顺序结构程序中,各语句是按照物理位置的先后次序顺序执行的,且每个语句都会被执行到。

下面通过几个例子来了解顺序结构程序设计。

【例 3-1】 从键盘输入一个大写字母,要求转换为小写字母输出。

该程序要解决的问题是将输入的大写字母转换为小写字母并输出。输入的是大写字母,最终输出的是小写字母,如何将大写字母转换为小写字母是数据的处理过程。小写字母的十进制 ASCII 码值比大写字母的 ASCII 码值大 32,所以只要将大写字母的 ASCII 码值加上 32,即可得到小写字母的 ASCII 码值,该问题通过字符的运算就可以解决了。

参考程序如下:

```
/* Chap3_1.c:将大写字母转换为小写字母 */
#include<stdio.h>
int main()
{
  char ch1,ch2;
  ch1=getchar();
  printf("%c,%d\n",ch1,ch1);
  ch2=ch1+32;
  printf("%c,%d\n",ch2,ch2);
  return 0;
}
```

程序运行结果如下:

```
A↙
A,65
a,97
```

在该程序中,用 getchar()函数得到从键盘输入的大写字母 A,赋值给字符型变量 ch1。“ch2=ch1+32;”是将 A 的 ASCII 码值加上 32 再赋值给字符型变量 ch2。字符型数据和整型数据之间可以混合使用。

【例 3-2】 交换两个变量的值。设两个变量分别为 a 和 b,要交换两个变量的值需引入中间变量。

参考程序如下:

```
/*Chap3_2.c:交换两个变量的值*/
#include<stdio.h>
int main()
```

```
{
  int a,b;
  int temp;
  scanf("%d%d",&a,&b);
  printf("a=%d,b=%d\n",a,b);
  temp=a;
  a=b;
  b=temp;
  printf("a=%d,b=%d\n",a,b);
  return 0;
}
```

程序运行结果如下：

```
5□□□10↙
a=5,b=10
a=10,b=5
```

说明：程序的执行过程分为 3 个步骤：键盘输入两个数；交换两个数；输出交换后的两个数。

【例 3-3】 求一个圆的面积和周长。

该程序中需要输入的数据是圆的半径，输出的数据是圆的面积和周长，计算圆的面积和周长的过程是处理和运算的过程。圆的面积＝圆周率×半径×半径，圆的周长＝2×圆周率×半径。

参考程序如下：

```
/*Chap3_3.c：计算一个圆的面积和周长*/
#include<stdio.h>
#define PI 3.1415926
int main()
{
  float r;
  float l,s;
  scanf("%f",&r);
  s=PI*r*r;
  l=2*PI*r;
  printf("半径为：%f,周长为：%f,面积为：%f\n",r,l,s);
  return 0;
}
```

程序运行结果如下：

```
4↙
半径为：4.000000,周长为：25.132740,面积为：50.265480
```

关于本程序，需要注意以下两点：

(1)圆周率被多次用到，通过“#define PI 3.141 592 6”将其定义为一个符号常量 PI，使

用起来比较方便。

(2)在 C 语言中,乘号用“*”表示。

【例 3-4】 输入三角形的三边,计算三角形的面积。

为了简单起见,在运行程序时,假设输入的三角形三边能构成三角形。由数学知识可知求三角形的面积公式为:

$$area=\sqrt{s(s-a)(s-b)(s-c)}$$

其中,s=(a+b+c)/2。

参考程序如下:

```
/* Chap3_4.c:计算三角形的面积 */
#include<stdio.h>
#include<math.h>
int main()
{
  int a,b,c;
  float s,area;
  scanf("%d,%d,%d",&a,&b,&c);
  s=1.0/2*(a+b+c);
  area=sqrt(s*(s-a)*(s-b)*(s-c));
  printf("面积为:%4.2f\n",area);
  return 0;
}
```

程序运行结果如下:

```
4,5,6↙
面积为:9.92
```

在本程序中,sqrt 是求平方根函数,使用该函数时必须在程序开头加上一条“#include”命令,将头文件“math.h”包含到程序中。

注意:以后凡在程序中要用到数学函数库中的函数,都应该包含“math.h”头文件。

需要说明的是,本程序没有考虑三角形成立的条件:任意两边之和都要大于第三边。因此,运行程序时输入的三边要符合构成三角形的条件。

3.3 选择结构程序设计

前面已经学习了顺序结构,本节主要介绍选择结构程序设计。

在实际生活中,常会碰到一些要作选择的事情。例如,今天天气好,我们就去郊游;天气不好,就留在家里。这就是平常所说的根据不同的条件做出不同的选择。

程序设计也是如此,程序在执行时,常通过条件来决定往下执行的流程,若满足条件则执行下一个流程;不满足条件则执行另一个流程。不管条件满足还是不满足,在执行完流程后,都要交汇到一起,再继续执行下面的其他语句。用条件来判断、选择程序执行流程,这种程序结构称为选择结构。

从上面的例子中分析可以看出，选择结构有两个要素：一个是条件，一个是执行的操作。天气的好坏情况是条件，要执行的操作有两个：一是郊游，一是留在家里。但是要根据天气的好与坏在两者之间做出一个选择。

选择结构的类型一般有三种：单分支选择结构、两分支选择结构和多分支选择结构。但是不管程序中有多少个分支，只能选择一个分支执行。

3.3.1　条件的描述和条件表达式

在 C 语言中，表示条件的表达式主要有关系表达式和逻辑表达式，其表达式的运算结果都会得到一个逻辑值。逻辑值只有两个，用“真”和“假”来表示，即关系成立为“真”，关系不成立为“假”。在 C 语言中，没有专门的“逻辑值”，而用非零表示“真”值，用零表示“假”值。因此，对于任意一个表达式，如果值为零时，就代表一个“假”值；只要值为非零，无论是正数还是负数，都代表一个“真”值。

1. 关系运算符和关系表达式

1）关系运算符

比较两个操作数大小的运算符称为关系运算符。例如，a＞b 是比较运算，“＞”是关系运算符。

C 语言有 6 种关系运算符，见表 3-1。

表 3-1　关系运算符

关系运算符	含　义	运算级别	
＜	小于	优先级相同	高 ↑
＜＝	小于或等于		
＞	大于		
＞＝	大于或等于		
＝＝	等于	优先级相同	↓ 低
!＝	不等于		

对于表 3-1，说明以下几点：

(1)其中前 4 种运算符的优先级相同，后两种相同，且前 4 种的优先级别高于后两种。

(2)关系运算符的优先级低于算术运算符(＋、－、＊、/、％)。

(3)关系运算符的优先级高于赋值运算符(＝)。

例如：

- c＞a＋b　　等效于　c＞(a＋b)
- a＞b＝＝c　等效于　(a＞b)＝＝c
- a＝b＞c　　等效于　a＝(b＞c)

提示：建议初学者在编写程序时最好把括号加上，防止混淆。

2）关系表达式

用关系运算符将两个表达式(可以是算术表达式、关系表达式、逻辑表达式、赋值表达式、字符表达式)连接起来的式子称为关系表达式。关系表达式的值是一个逻辑值(“真”或“假”)，用“1”或“0”表示。

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
- a+b>b+c
- (a=3)>(b=5)
- 'a'<'b'
- (a>b)<(b<c)

【例 3-5】 理解关系运算。

参考程序如下：

```
/* Chap3_5.c:关系运算符的使用 */
#include<stdio.h>
int main()
{
  int a=3,b=4;
  printf("%d\n",a==b);
  printf("%d\n",a<b);
  printf("%d\n",a<=b);
  printf("%d\n",'a'<'b');
  return 0;
}
```

程序运行结果如下：

```
0
1
1
1
```

说明：关系表达式'a'<'b'的结果为 1(真)，因为两个字符常量之间的比较是以它们对应的 ASCII 码值参与运算的，字符 a 的 ASCII 码值小于字符 b 的 ASCII 码值，所以关系表达式的值为 1。

2. 逻辑运算符和逻辑表达式

1)逻辑运算符

C 语言中提供了 3 种逻辑运算符，分别是逻辑非(!)、逻辑与(&&)和逻辑或(||)。具体含义如表 3-2 所示。

表 3-2 逻辑运算符

逻辑运算符	含 义	运算级别
!(逻辑非)	相当于日常用语中的“不是”	高 ↑
&&(逻辑与)	相当于日常用语中的“且”、“和”	
||(逻辑或)	相当于日常用语中的“或”	↓ 低

逻辑运算的值只有“真”和“假”两种，分别用“1”和“0”表示。

逻辑运算的真值表见表 3-3。

表 3-3　逻辑运算的真值表

a	b	a&&b	a||b	!a
真	真	真	真	假
真	假	假	真	假
假	真	假	真	真
假	假	假	假	真

2)逻辑表达式

用逻辑运算符将关系表达式或逻辑量连接起来构成的式子称为逻辑表达式。逻辑表达式的一般形式为：

表达式 逻辑运算符 表达式

在一个逻辑表达式中可以包含多个逻辑运算符，例如：

!a&&b||c>d&&y

逻辑运算符中的“&&”和“||”的优先级低于关系运算符，而“!”高于算术运算符。例如：

- (a>b)&&(x>y)可写成 a>b && x>y。
- (!a)||(a==b)可写成!a|| a==b。

【例 3-6】 理解逻辑运算。

参考程序如下：

```
/* Chap3_6.c:逻辑运算符的使用 */
#include<stdio.h>
int main()
{
  int x,y;
  int a=3,b=4,c=5;
  printf("%d\n",a<b && b<c);
  printf("%d\n",a<b || b>c);
  printf("%d\n",a||b+c && b-c);
  printf("%d\n",!(a>b) && !c||1);
  printf("%d\n",a+b>c && b==c);
  printf("%d\n",!(x=a) &&(y=b)&&0);
  return 0;
}
```

程序运行结果如下：

```
1
1
1
1
0
0
```

在多个运算符进行运算时，应该注意运算符的优先级，详见附录 A。本例中，在

“!(x=a) &&(y=b)&&0”表达式中，先是将 a 的值赋值给 x 再计算!x 的值，然后与赋值表达式 y=b 进行“逻辑与”运算，最后再与 0 进行“逻辑与”运算。

在 C 语言中，“&&”和“||”被称为短路运算符，即在一个或多个“逻辑与”连接的表达式中，只要有一个操作数为 0，就不必做后面的逻辑与运算，整个表达式的值为 0。而由一个或多个“逻辑或”连接而成的表达式中，只要有一个操作数为非 0，就不需再进行后面的运算，则整个表达式的值为 1。

【例 3-7】 “逻辑与”和“逻辑或”的使用。

参考程序如下：

```
/* Chap3_7.c:“逻辑与”和“逻辑或”的使用 */
#include<stdio.h>
int main()
{
  int a=0,b=1,c=2;
  a&&++b&&++c;                    /* a 的值为 0,则整个表达式的值为 0 */
  printf("表达式 1 的值为:\n");
  printf("a=%d,b=%d,c=%d\n",a,b,c);
  ++b||++a&&++c;                  /* ++b 的值为 2,则整个表达式的值为 1 */
  printf("表达式 2 的值为:\n");
  printf("a=%d,b=%d,c=%d\n",a,b,c);
  return 0;
}
```

程序运行结果如下：

```
表达式 1 的值为:
a=0,b=1,c=2
表达式 2 的值为:
a=0,b=2,c=2
```

3. 常用运算符的优先级

学习过的常用运算符有赋值运算符、算术运算符、关系运算符、逻辑运算符等，其优先级见表 3-4。

表 3-4 常用运算符的优先级

运 算 符	运算符类型	优 先 级
()	圆括号	高 ↑ 低
!	逻辑运算符	
*,/,%	算术运算符	
+,-	算术运算符	
<,<=,>,>=	关系运算符	
==,!=	关系运算符	
&&	逻辑运算符	
||	逻辑运算符	
=	赋值运算符	
,	逗号运算符	

3.3.2　if 语句

if 语句用来判断所给的条件是否满足，根据判断的结果（“真”或“假”）决定执行给出的两种操作之一。

C 语言中提供了 3 种形式的 if 语句。

1. if 语句

if 语句的格式为：

```
if(表达式) 语句 1
```

例如：

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

这种 if 语句的执行过程如图 3-5 所示。

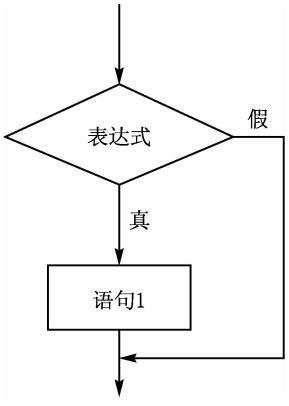


图 3-5　if 语句的执行过程

如果表达式的值为真，则执行“语句 1”，否则不执行“语句 1”。这种结构被称为单分支的选择结构。需要注意的是，语句 1 可以是一条语句，还可以是多条语句。如果为多条语句，则要使用花括号括起来。

【例 3-8】 if 语句的应用。

参考程序如下：

```
/* Chap3_8.c:if 语句的格式与应用 */
#include<stdio.h>
int main()
{
  int a,b;
  scanf("%d,%d",&a,&b);
  if(a>b)
    printf("%d>%d\n",a,b);
  return 0;
}
```

程序运行结果如下：

5,4↙

5>4

该程序中,如果输入的第 1 个数不大于第 2 个数,如 4 和 5,则不执行语句"printf("%d>%d\n",a,b);",程序没有运行结果。

【例 3-9】 输入两个实数,按照从小到大的次序输出这两个数。

两个数的排序问题比较简单,只需要将两个数进行比较,如果第 1 个数大于第 2 个数,则将两个数进行交换。

参考程序如下:

```
/* Chap3_9.c:两个数的排序问题 */
#include<stdio.h>
int main()
{
  float x,y,t;
  scanf("%f,%f",&x,&y);
  if(x>y)
  {
    t=x;
    x=y;
    y=t;
  }
  printf("%.2f,%.2f\n",x,y);
  return 0;
}
```

程序运行结果如下:

9,3↙

3.00,9.00

本程序中的 if 语句,如果条件(x>y)成立,执行语句 1,该语句是由 3 个语句组成,因此应该使用花括号括起来。

思考:如果是 3 个数,如何实现从小到大的排序?

2. if…else 语句

if…else 语句的格式为:

```
if(表达式)
  语句 1
else
  语句 2
```

这种 if 语句的执行过程如图 3-6 所示。

若表达式的值为真(非 0),就执行语句 1;若表达式的值为假(0),则执行语句 2,然后执行 if 语句后的下一条语句。这就是典型的两分支选择结构。

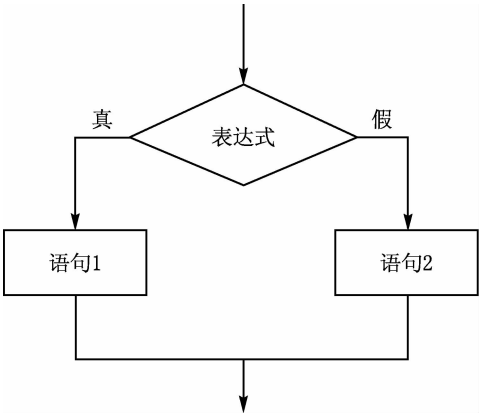


图 3-6　if…else 语句执行过程

【例 3-10】 输入两个整数，输出其中的较大数。

求两个数中的较大数，其算法如图 3-2 所示。该程序的流程有两个，是一个典型的选择结构，可采用单分支和两分支结构来解决。在这里，采用两分支结构来实现其算法。

参考程序如下：

```
/* Chap3_10.c:两个数中的较大数问题 */
#include<stdio.h>
int main()
{
  int a,b,max;
  scanf("%d,%d",&a,&b);
  if(a>b)
    max=a;
  else
    max=b;
  printf("max:%d\n",max);
  return 0;
}
```

程序运行结果如下：

```
8,10↙
max:10
```

本程序的执行过程分为 3 步：输入两个数，求两个数中的较大数，输出较大的数。其中第 2 步是由两分支的选择结构来实现的。

【例 3-11】 输入 3 个整数，输出其中的最大数。

参考程序如下：

```
/*Chap3_11.c:求 3 个数中最大数问题*/
#include<stdio.h>
int main()
{
```

```
  int a,b,c,max;
  scanf("%d,%d,%d",&a,&b,&c);
  if(a>b)
    max=a;
  else
    max=b;
  if(c>max)
    max=c;
  printf("max:%d\n",max);
  return 0;
}
```

程序运行结果如下：

```
12,9,36↙
max:36
```

说明：求 3 个数的最大数问题，还可以采用其他方法(如标志法)，有兴趣的读者可以多思考，设计其他算法和流程图。

【例 3-12】 判断某一年是否为闰年。

分析：闰年的条件是能被 4 整除但不能被 100 整除，或能被 400 整除。判断闰年的条件可以用一个逻辑表达式来表示：

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

参考程序如下：

```
/* Chap3_12.c:判断某一年是否为闰年 */
#include<stdio.h>
int main()
{
  int year;
  printf("Please input year:");
  scanf("%d",&year);
  if((year%4==0 && year%100!=0)||year%400==0)
    printf("%d 是闰年! \n",year);
  else
    printf("%d 不是闰年! \n",year);
  return 0;
}
```

程序运行结果如下：

```
Please input year:1996↙
1996 是闰年!
```

3. if…else…if 语句

当有多个分支选择时，可采用 if…else…if 语句，其一般形式如下：

```
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
```

这种 if 语句的执行过程如图 3-7 所示。

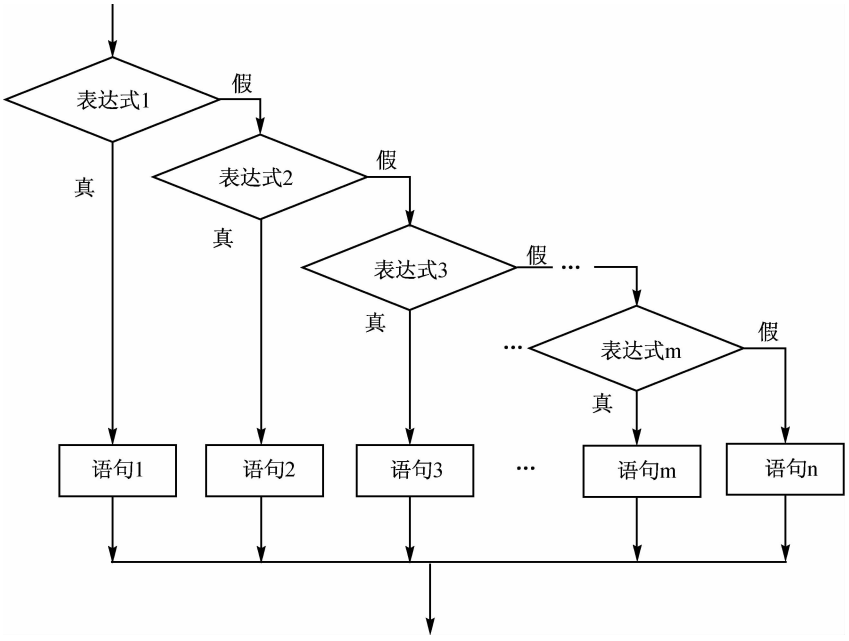


图 3-7　多分支选择结构

关于多分支选择结构，说明以下两点：

(1)先判断表达式 1 的值，当表达式 1 的值为真时，则执行语句 1，执行完之后，跳出 if 语句；当表达式 1 的值为假时，再判断表达式 2 的值，当表达式 2 的值为真时，则执行语句 2，执行完之后，跳出 if 语句；当表达式 2 的值为假时，再判断表达式 3 的值……当所有表达式的值都为假时，执行语句 n，执行完之后，跳出 if 语句。

(2)最后一个 else 可以不存在。例如：

```
if(x>0)
    y=1;
else if (x==0)
  y=0;
else if(x<0)
```

```
  y=-1;
```

【例 3-13】 阅读下面的程序,分析程序的功能。

参考程序如下:

```
/* Chap3_13.c:理解 if…else…if 语句的执行过程 */
#include<stdio.h>
int main()
{
  int x,y;
  printf("Please input x:");
  scanf("%d",&x);
  if(x<1) y=x;
  else if(x>=1&&x<15) y=2*x+1;
  else if(x>=15&&x<24) y=6*x;
  else y=10*x;
  printf("x=%d,y=%d\n",x,y);
  return 0;
}
```

程序运行结果如下:

```
Please input x:-5↙
x=-5,y=-5
```

该程序实现的是求一个分段函数的功能,当 x 的值处于不同的取值范围时,y 的值有 4 种不同的情况,该问题采用 4 个分支选择结构来实现。但是,即使程序有 4 个分支,也只能根据不同的情况选择某一个分支执行。

【例 3-14】 用 if 语句实现对学生成绩等级的判断。

根据学生的成绩来判断学生的成绩等级:90 分以上为优秀,80～89 为良好,60～79 为及格,低于 60 分为不及格。

参考程序如下:

```
/* Chap3_14.c:if 语句的应用 */
#include<stdio.h>
int main()
{
  float cj;
  scanf("%f",&cj);
  if(cj>=90)
    printf("优秀\n");
  else if(cj>=80)
    printf("良好\n");
  else if(cj>=60)
    printf("及格\n");
```

```
    else
      printf("不及格\n");
    return 0;
}
```

程序运行结果如下：

```
32↙
不及格
```

在使用 if 语句时应注意以下问题：

3 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，但也可以是其他表达式(如赋值表达式等)，甚至还可以是一个变量。例如：if(a=5)语句和 if(b) 语句都是允许的。只要表达式的值为非 0，即为“真”。如在“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，但它是合法的。

又如，有程序段：

```
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”。这种用法在程序中是经常出现的。

3.3.3　if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句，称为 if 语句的嵌套。其一般格式为：

```
if(表达式)
  if(表达式)语句 1
  else  语句 2
else
  if(表达式)语句 3
  else  语句 4
```

应当注意 if 与 else 的配对关系。else 总是与上面距它最近的 if 配对。假如写成下面这样也可以。

```
if(表达式)
  if(表达式)语句 1
else
  if(表达式)语句 2
else  语句 3
```

编写者把 else 写在与第 1 个 if(外层 if)同一列，希望 else 与第 1 个 if 对应，但是实际上 else 还是与第 2 个 if 配对，因为它们相距最近。因此，最好使内嵌 if 语句也包含 else 部分，这样 if 的数目与 else 的数目相同，从内层到外层一一对应，不致出错。

如果 if 和 else 的数目不一样，为实现程序设计者的意图，可以使用花括号来确定配对关系。例如：

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

这样，“{}”就限定了内嵌 if 语句的范围，因此，else 与第 1 个 if 配对。例如：

```
if(a>=b)
{
  if(a>b)
    printf("a>b");
  else
    printf("a=b");
}
else
printf("a<b");
```

【例 3-15】 有一个函数：

$$y=\begin{cases}1 & (x>0)\\ 0 & (x=0)\\ -1 & (x<0)\end{cases}$$

编写一个程序，输入一个 x 的值，输出 y 的值。其算法可以用图 3-8 所示的流程图来表示。参考程序如下：

```
/* Chap3_15.c:符号函数的应用 */
#include<stdio.h>
int main()
{
  int x,y;
  scanf("%d",&x);
  if(x>0) y=1;
  else
    if (x==0) y=0;
    else y=-1;
  printf("x=%d,y=%d\n",x,y);
  return 0;
}
```

程序运行结果如下：

```
6↙
x=6,y=1
```

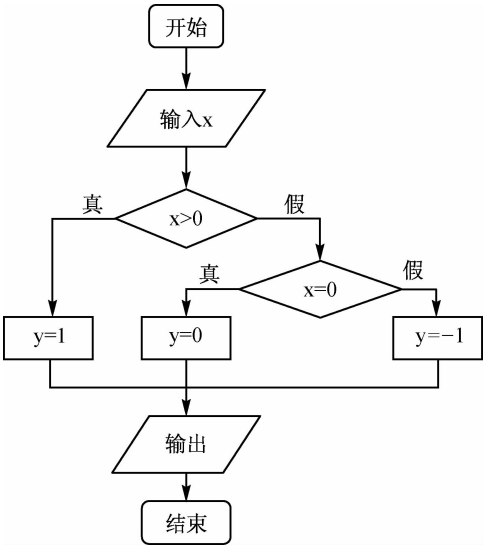


图 3-8　流程图

说明:根据不同的条件,该问题的流程也可以不同。读者可以多思考,设计不同的算法,画出不同的流程图,从而设计出不同的程序。

【例 3-16】 用 if 语句的嵌套来编写程序,判断某一个年份是否是闰年。

前面介绍过闰年的条件,在本程序中,用变量 leap 代表是否是闰年的信息。若为闰年,使 leap=1;不是闰年,leap=0。最后根据变量 leap 的值输出相关信息。

参考程序如下:

```
/*Chap3_16.c:判断某一个年份是否为闰年*/
#include<stdio.h>
int main()
{
  int year,leap;
  scanf("%d",&year);
  if(year%4==0)
  {
    if(year%100!=0)
      leap=1;
    else
    {
      if(year%400==0)
        leap=1;
      else
        leap=0;
    }
  }
  else
```

```
    leap=0;
  if(leap)
    printf("%d 是闰年!",year);
  else
    printf("%d 不是闰年!",year);
  return 0;
}
```

程序运行结果如下：

```
1990↙
1990 不是闰年!
```

3.3.4 条件运算符

在 if 语句中，当表达式的值无论真或假时都只执行一个赋值语句给同一个变量赋值时，可以用简单的条件运算符来处理。例如，以下的 if 语句：

```
if(a>b) max=a;
else max=b;
```

可以用下面的条件运算符来处理：

```
max=(a>b)? a:b;
```

其中，(a>b)? a:b 是一个条件表达式。它是这样执行的：如果 a>b 条件成立，则条件表达式取 a 值，否则取 b 值。

条件运算符要求有 3 个操作对象，称为三目运算符，它是 C 语言中唯一的一个三目运算符。条件表达式的一般形式为：

```
表达式 1? 表达式 2:表达式 3
```

说明以下几点：

(1)如果在条件语句中，只执行单个的赋值语句时，常可使用条件表达式来实现。不但程序简洁，也提高了运行效率。求值规则为：如果表达式 1 的值为真，则以表达式 2 的值作为整个条件表达式的值，否则以表达式 3 的值作为整个条件表达式的值。条件表达式通常用于赋值语句之中。

(2)条件运算符优先于赋值运算符，但其优先级比关系运算符和算术运算符都低。因此，“max=(a>b)? a:b;”中的括号可以不要，可写成：

```
max=a>b? a:b;
```

(3)条件运算符的结合方向为“自左向右”。如果有以下条件表达式：

```
a>b? a:c>d? c:d
```

相当于：

```
a>b? a:(c>d? c:d)
```

(4)条件运算符不能取代一般的 if 语句，只有在 if 语句中内嵌的语句为赋值语句，且两个分支都给同一个变量赋值时才能代替 if 语句。像下面的 if 语句就无法用一个条件表达式代替：

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

但可以用下面的语句代替：

```
printf("%d",a>b? a:b);
```

即将条件表达式的值输出。

(5)条件表达式中，表达式 1 的类型可以与表达式 2 和表达式 3 的类型不同。例如：

```
x?'a':'b'
```

x 为整型变量，若 x=0，则条件表达式的值为'b'。表达式 2 和表达式 3 的类型也可以不同，此时条件表达式的值的类型为两者中较高的类型。例如：

```
x>y? 1:1.5
```

如果 x≤y，则条件表达式的值为 1.5；若 x>y，值应为 1。由于 1.5 是实型，级别比整型高，因此，将 1 转换成实型值 1.0。

【例 3-17】 输入一个字符，判断它是否为大写字母。如果是，将它转换成小写字母；如果不是，不转换，然后输出最后得到的字符。

同一个字符的小写字母和大写字母的 ASCII 码值相差 32，因此，将一个大写字母转换成小写字母可以将其 ASCII 码值加 32。

参考程序如下：

```
/* Chap3_17.c:大小写字母的转换 */
#include<stdio.h>
int main()
{
  char ch;
  scanf("%c",&ch);
  ch=(ch>='A'&&ch<='Z')? (ch+32):ch;
  printf("%c\n",ch);
  return 0;
}
```

程序运行结果如下：

```
B↙
b
```

【例 3-18】 用条件运算符来实现求 3 个数中的最大数问题。

参考程序如下：

```
/* Chap3_18.c:输入 3 个数，求其中的最大数 */
#include<stdio.h>
int main()
{
  long a,b,c,max;
  printf("Please input a,b,c:");
  scanf("%ld,%ld,%ld",&a,&b,&c);
  max=(a>b)? a:b;
  max=(max>c)? max:c;
  printf("The biggest is %ld\n",max);
```

```
  return 0;
}
```

程序运行结果如下：

```
Please input a,b,c:3,6,9↙
The biggest is 9
```

3.3.5 switch 语句

多分支选择结构可以使用嵌套的 if 语句来处理，但如果分支较多，嵌套的 if 语句层数多，程序冗长而且可读性降低。C 语言还提供了另一种用于多分支选择的语句——switch 语句，其一般形式为：

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

switch 语句的执行过程为：当表达式与常量表达式 1 相等时，执行语句 1，用 break 语句终止 switch 语句，不再进行其他条件判断；执行右花括号后的下一条语句。当表达式与常量表达式 1 不相等时，判断表达式与常量表达式 2 是否相等，若相等，则执行语句 2，用 break 语句终止 switch 语句，不再进行其他条件判断，执行右花括号后的下一条语句；若不相等，再判断表达式与常量表达式 3 是否相等，依此类推。当所有条件都不满足时，则执行语句 n，default 表示其他条件。

例如，要求按考试成绩的等级打印其百分制分数段，可以用 switch 语句来实现：

```
switch(grade)
{
  case 'A':printf("90～100\n");
  case 'B':printf("80～89\n");
  case 'C':printf("60～79\n");
  case 'D':printf("＜60\n");
}
```

【例 3-19】 思考程序的运行结果，理解 switch 语句。

参考程序如下：

```
/* Chap3_19.c:switch 语句的使用 */
#include<stdio.h>
int main()
{
  int x=3;
  switch(x)
  {
    case 1:
    case 2:printf("x<3\n");
    case 3:printf("x=3\n");
    case 4:
    case 5:printf("x>3\n");
    default:printf("x unknown\n");
  }
  return 0;
}
```

程序运行结果如下：

```
x=3
x>3
x unknown
```

思考：该程序为什么是这个结果？如果将程序改为以下形式，那结果呢？

```
#include<stdio.h>
int main()
{
  int x=3;
  switch(x)
  {
    case 1:
    case 2:printf("x<3\n");break;
    case 3:printf("x=3\n");break;
    case 4:
    case 5:printf("x>3\n");break;
    default:printf("x unknown\n");break;
   }
  return 0;
}
```

使用 switch 语句时，需要注意以下几点：

(1)一个 switch 语句是由一些 case 子语句和一个可缺省的 default 子结构所组成的复合语句，case 子语句和 default 子结构位于一对花括号之内。

(2)switch 后面的表达式只能对整数求值,可以使用字符或整数,但不能使用实数表达式。case 子结构的表达式应该是整型常数表达式,不能含有变量或函数。例如,可以为:

```
case 3+4;
```

但不允许写成:

```
int x=2,y=5;
switch (z)
{
  …
  case x+y:
  …
}
```

(3)一个 switch 语句中不能出现两个具有相同值的常量表达式。例如:

```
case 3+5:
…
case 2+6:
```

(4)switch 的匹配测试只能测试是否相等,不能测试关系表达式或逻辑表达式。

(5)各个 case 语句和 default 的出现次序不影响执行结果。例如,可以先出现“default:…”,再出现“case 'D',…”,然后是“case 'A':…”。

(6)switch 语句允许嵌套。

【例 3-20】 用 switch 语句实现对学生成绩等级的判断。

根据学生的分数,对学生的成绩进行等级判断。学生的成绩等级分为 A,B,C,D 四个等级:成绩在 60 分以下的为 D 级;60～79 的为 C 级;80～89 的为 B 级,90 分以上的为 A 级。

参考程序如下:

```
/* Chap3_20.c: switch 语句的应用 */
#include<stdio.h>
int main()
{
  float cj;
  char ch;
  scanf("%f",&cj);
  switch((int)cj/10)
  {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:ch='D';break;
    case 6:
    case 7:ch='C';break;
```

```
    case 8:ch='B';break;
    case 9:
    case 10:ch='A';break;
    default:printf("输入数据有误\n");
  }
  printf("成绩 %.2f 的等级为:%c\n",cj,ch);
  return 0;
}
```

程序运行结果如下：

```
96↙
成绩 96.00 的等级为:A
```

本程序中多个 case 共用一组执行语句，当 cj＜60 时，即(int)cj/10 的值为 0,1,2,3,4,5 时，都执行“ch='D';break;”语句。

【例 3-21】 用 switch 语句实现简单的计算器程序。

程序运行时，用户输入加、减、乘、除中的任何一个字符，然后再输入两个操作数，程序将会显示相应的计算结果。

参考程序如下：

```
/*Chap3_21.c:简单计算器程序*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
  char ch;
  int a,b,result;
  printf("请输入加减乘除四个运算符中的一种:\n");
  scanf("%c",&ch);
  printf("请输入两个操作数:");
  scanf("%d,%d",&a,&b);
  switch(ch)
  {
    case'+':
      result=a+b;break;
    case'-':
      result=a-b;break;
    case'*':
      result=a*b;break;
    case'/':
    if(b==0)
```

```
            printf("除数不能为零! \n");
        else
            result=a/b;
        break;
        default:exit(0);
    }
    printf("%d%c%d=%d\n",a,ch,b,result);
    return 0;
}
```

程序运行结果如下：

```
请输入加减乘除四个运算符中的一种：
*↙
请输入两个操作数:12,5↙
12*5=60
```

说明：函数 exit()通常是在子程序中用来终结程序的，使用后程序自动结束返回操作系统。在程序中使用函数 exit()时，应该在程序开头包含定义它的头文件 stdlib.h。

3.4 循环结构程序设计

循环结构是程序控制结构中的第 3 种结构，也是应用比较广泛的一种结构。循环结构又称为重复结构，可以完成重复性、规律性的操作。在人们所需处理的运算任务中，常常需要用到循环，如求若干个数的和、迭代求根等。

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。循环结构是由循环语句来实现的。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。

3.4.1 迭代与穷举算法

1.迭代法

迭代是一个不断用新值取代变量的旧值，或由旧值递推得出变量新值的过程。下面讲述一个经典的兔子繁殖问题的例子。

著名意大利数学家 Fibonacci 曾提出了一个有趣的问题：设有一对新生兔子，从第 3 个月开始它们每月都生育一对兔子。小兔子长到第 3 个月后每个月又生一对兔子。按此规律，并假设兔子没有死亡，一年后将共有多少对兔子？

人们发现每月的兔子对数组成了如下数列：

1,1,2,3,5,8,13,21,34…

并把它称为 Fibonacci 数列。那么，这个数列是如何推导出来的呢？

观察 Fibonacci 数列后，可以发现这样一个规律：从第 3 个数开始，每一个数都是它前面两个相邻数之和。这是因为，在没有兔子死亡的情况下，每个月的兔子对数由两部分组成：相邻上一月的老兔子对数和这月刚生下来的新兔子对数。上一个月的老兔子对数即其前一个数，这个月刚生下来的新兔子恰好为上上月的兔子对数。因为，上一个月的兔子中有一部分在这个月还不能生小兔子，只有上上月的兔子才能每对生育一对小兔子。

上述兔子繁殖的规律可以用数学公式表示为：

$$\begin{cases} F_1=1 & (n=1) \\ F_2=1 & (n=2) \\ F_n=F_{n-1}+F_{n-2} & (n\geqslant 3) \end{cases}$$

兔子总数变化规律具体情况如表 3-5 所示。

表 3-5　兔子数量随时间变化规律

第几个月	1	2	3	4	5	6	7	8	9	10	…
当月新生兔子对数	1	0	1	1	2	3	5	8	13	21	…
上个月兔子总对数	0	1	1	2	3	5	8	13	21	34	…
当月兔子总对数	1	1	2	3	5	8	13	21	34	55	…

用 C 语言来描述上述问题时，可表示为：

```
fib=fib1+fib2;
fib1=fib2;
fib2=fib;
```

其中，fib1 表示当月新生的兔子对数，fib2 表示上个月的兔子总对数，fib 是当月的兔子总对数。

把兔子数量变化规律抽象成 Fibonacci 数列问题后，其流程图如图 3-9 所示。

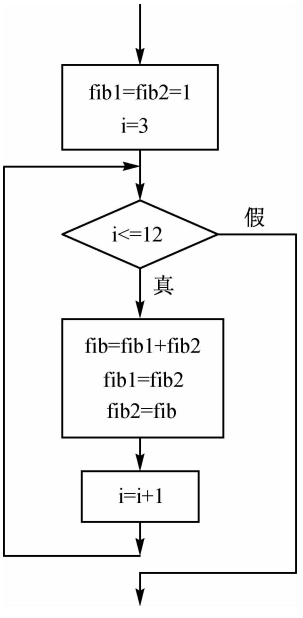


图 3-9　Fibonacci 数列流程图

2. 穷举法

穷举是另一种重复性算法。它的基本思想是，对问题的所有可能状态逐一测试，直到找到解或全部可能状态都测试完为止。

循环控制有两种方法：计数法与标志法。计数法要先确定循环的次数，然后逐次进行测试，达到测试次数后，循环结束。标志法用来判断是否达到某一目标，一旦达到，便使循环结束。

【例 3-22】 输入 10 个数，将最大数打印出来。

从所输入的 10 个数中选取最大数的算法：

S1：先输入一个数，暂时将它作为最大数。

S2：再输入 9 个数，穷举这 9 个数，每输入一个，把它与最大数进行比较，把较大者作为新的最大数。

S3：输出最大数。

这个算法用如图 3-10 所示的流程图表示。

这里需要一个变量 i（相当于计数器）来统计输入数的个数，每输入一个数，它都要增 1。

计数法使用起来简单、方便，但它需要事先知道循环的次数。如果输入的数据量很大时，会使操作人员感到很不方便。

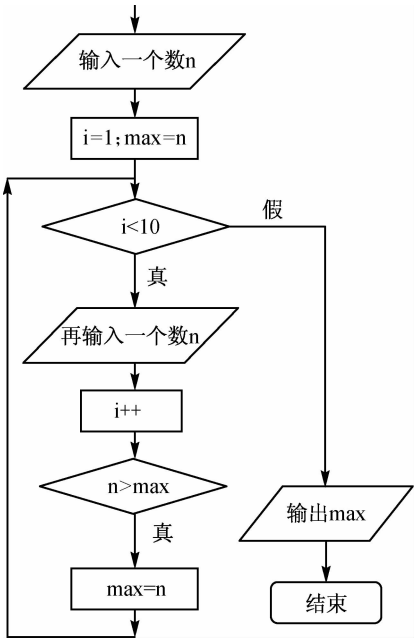


图 3-10 用计数法求 10 个数中的最大数

当不愿意或没有条件使用计数法时，可使用标志法。具体过程是：在测试之前先设置一个标志性数据，如本例中可以设为－32 768（所输入的数据是不可能用到的）。这样，操作者在没有数据可以输入时，就输入该数来表示已经没有数据。这样对输入的数据就没有限定了。

用标志法求多个数中的最大数算法及流程图，请读者自己设计并绘制。

【例 3-23】 百钱买百鸡问题。

公元前 5 世纪，我国古代数学家张丘在《算经》一书中提出了“百鸡问题”：鸡翁一值钱

五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡，问：鸡翁、鸡母、鸡雏各几何？

(1)基本解题思路。

用以下不定方程求解问题：

cocks＋hens＋chicks＝100

5×cocks＋3×hens＋chicks/3＝100

其中，cocks 表示鸡翁数，hens 表示鸡母数，chicks 表示鸡雏数。

对于上述不定方程，要先确定一个变量的值，才能求解。由题目所给出的条件，可以得到 3 个变量的取值范围，具体如下：

- cocks：0～19 中的整数。
- hens：0～33 中的整数。
- chicks：0～100 中的整数。

基本的解题思路是：依次让 cocks 取值域 0～19 中的一个整数，然后求 hens 和 chicks 的个数是否符合题意，合乎者为其解。

(2)百钱买百鸡的算法具体如图 3-11 所示。

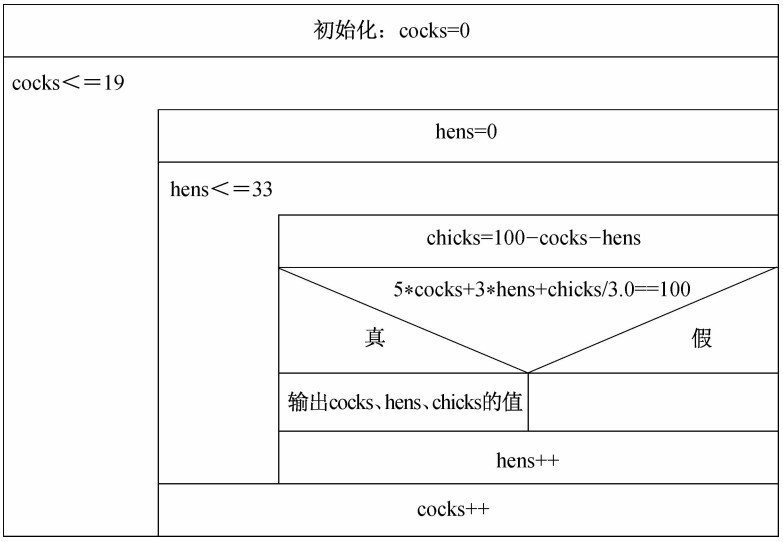


图 3-11　百钱买百鸡的算法

3.4.2　while 语句

while 语句用于实现“当型”循环结构。其一般形式为：

```
while(表达式)
  语句
```

其中，表达式是循环的条件，语句为循环体，它可以由一个或多个语句构成。它的执行过程是：

(1)先判断表达式的值为真(非 0)或为假(0)；

(2)如果表达式的值为真(非 0)，执行循环体语句，再重复步骤(1)；如果表达式的值为假(0)，循环结束，执行 while 语句后面的语句。

while 语句的执行特点是:先判断表达式的值,后执行循环体语句。

其流程图如图 3-12 所示。

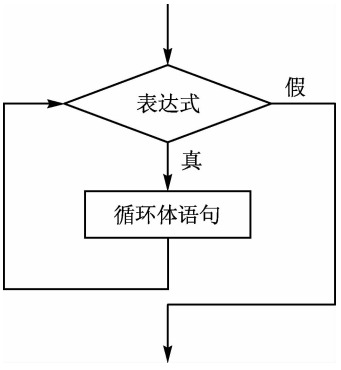


图 3-12 while 语句流程图

【例 3-24】 用 while 语句来实现以下的求和:S=1+2+3+…+10。

这是一个累加问题,要计算的是从 1～10 十个数字的和,属于循环计数问题。

参考程序如下:

```
/* Chap3_24.c:求和问题 */
#include<stdio.h>
int main()
{
  int i,sum=0;              /* sum 表示和,其初始值为 0 */
  i=1;                      /* 循环变量 i 的初始值为 1*/
  while(i<=10)
  {
    sum=sum+i;
    i=i+1;
  }
  printf("sum:%d\n",sum);
  return 0;
}
```

程序运行结果如下:

```
sum:55
```

说明以下几点:

(1)循环体如果由一个以上的语句构成,应该用花括号括起来,以复合语句的形式出现;否则 while 语句的范围只到 while 后面的第 1 个语句的分号处。例如,本例中 while 语句中如无花括号,则循环只执行到“sum=sum+i;”。

(2)在循环中应有使循环趋向于结束的语句,否则会出现死循环(循环无限次地进行)。例如,本例中循环结束的条件是 i>10,循环变量 i 的初始值为 1,因此,在循环体中应有使 i 增值的语句,如 i=i+1,从而使循环到一定阶段后结束。又如:

```
int main()
{
  int a,n=0;
  while(a=5)
    printf("%d",n++);
  return 0;
}
```

本例中,while 语句的循环条件为赋值表达式 a=5,该表达式的值永远为真;而循环体中又没有其他终止循环的手段,因此,该循环将无休止地进行下去,形成死循环。

(3)在循环中通常用一个变量来控制循环的开始和结束,这个变量被称为循环变量。例如,本程序中的变量 i 就是循环变量。

【例 3-25】 用键盘输入若干学生的成绩,计算并输出它们的和,当输入-1 时结束。

这是一个非计数循环问题,事先循环的次数不确定,需要根据用户的意图来确定。while 语句不仅可以实现计数循环,也可以实现非计数循环。

参考程序如下:

```
/* Chap3_25.c:求若干学生的成绩的和 */
#include<stdio.h>
int main()
{
  float cj,sum=0;
  scanf("%f",&cj);
  while(cj!=-1)
  {
    sum=sum+cj;
    scanf("%f",&cj);
  }
  printf("%.2f\n",sum);
  return 0;
}
```

程序运行结果如下:

```
65 55 85 -1↙
205.00
```

程序中循环结束条件是成绩 cj=-1,-1 只是一个标志,当用户不想再输入数据时,可以用-1 来结束。

使用 while 语句应注意以下几点:

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

```
int main()
{
```

```
    int a=0,n;
    printf("\n input n: ");
    scanf("%d",&n);
    while (n--)
      printf("%d ",a++);
    return 0;
}
```

本例程序将执行 n 次循环，每执行一次，n 值减 1，直到 n 值等于 0 为止。

(2)应注意循环条件的选择以避免死循环。

(3)允许 while 语句的循环体又是 while 语句，从而形成双重循环。

3.4.3 do…while 语句

while 语句是在结构头部检验循环条件，当循环条件为真时进入循环体；若为假则直接退出循环。do…while 语句与 while 语句不同，do…while 语句是在循环的尾部检验循环条件，也就是说，do…while 语句至少执行一次循环体。

do…while 语句的一般形式为：

```
do
  循环体语句
while(表达式);
```

其中，表达式是循环条件。它的执行过程是：

(1)先执行循环体语句一次，再判别表达式的值。

(2)如果表达式的值为假(0)，循环结束；如果表达式的值为真(非 0)，重复执行(1)，继续进行循环。

do…while 语句的特点是：先执行循环体语句，后判断表达式。其流程可用图 3-13 所示。

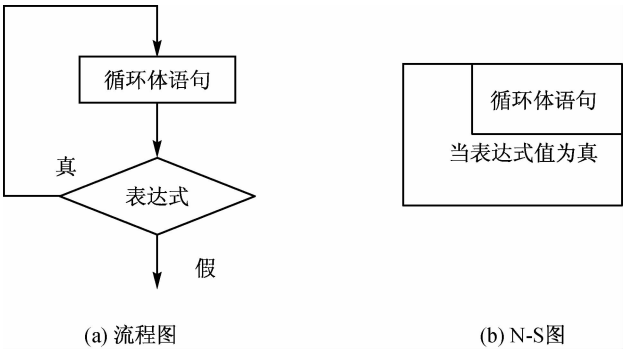


图 3-13 do…while 语句流程图和 N-S 图

while 语句和 do…while 语句一般都可以相互改写，但是要注意循环条件的变化。

【例 3-26】 用 do…while 语句来实现求 1+2+3+…+10。

参考程序如下：

```
/* Chap3_26.c:do…while 语句的应用 */
#include<stdio.h>
int main()
{
  int i,sum=0;
  i=1;
  do
  {
    sum=sum+i;
    i++;
  }while(i<=10);
  printf("%d\n",sum);
  return 0;
}
```

程序运行结果如下：

```
55
```

对于 do…while 语句还应注意以下几点：

(1)在 if 语句和 while 语句中，表达式后面都不能加分号，而在 do…while 语句的表达式后面则必须加分号。

(2)do…while 语句可以组成多重循环，也可以和 while 语句相互嵌套。

(3)当 do 和 while 之间的循环体由多个语句组成时，必须用“{}”括起来组成一个复合语句。

(4)do…while 和 while 语句可以相互替换，但两者之间的区别在于：do…while 是先执行后判断，因此，do…while 至少要执行一次循环体；而 while 是先判断后执行，如果条件不满足，则一次循环体也不执行。

【例 3-27】 编写一个程序，输入若干数，输出其中最大的数。

参考程序如下：

```
/* Chap3_27.c:求多个数的最大数问题 */
#include<stdio.h>
#define FLAG -32768
int main()
{
  int max,n;
  scanf("%d",&n);
  max=n;
  do
  {
    scanf("%d",&n);
    if(n>max)
```

```
        max=n;
    }
    while(n!=FLAG);
    printf("max: %d\n",max);
    return 0;
}
```

程序运行结果如下：

```
19↙
34↙
56↙
86↙
65↙
-32768↙
max:86
```

在程序中，FLAG 是作为循环结束的一个标志。本程序中把它定义为符号常量，用它代表－32 768，因为所输入的数据不可能用到－32 768。

3.4.4 for 语句

for 语句是 C 语言所提供的功能更强、使用更广泛的一种循环语句，它不仅可以用于循环次数确定的情况，而且可以用于循环次数不确定而只给出循环结束条件的情况。

1. for 语句的一般形式

for 语句的一般形式为：

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

for 语句中的 3 个成分都是表达式，其含义可以理解为：

(1)表达式 1 通常用来给循环变量赋初值，一般是赋值表达式。当然，也允许在 for 语句之前给循环变量赋初值，此时可以省略该表达式。

(2)表达式 2 通常是循环条件，以便决定是否继续执行循环，一般为关系表达式或逻辑表达式，也可以是其他任意类型的数值表达式，只要它的值为真，就可以执行循环体。

(3)表达式 3 通常可用来修改循环变量的值，一般是赋值语句。

for 语句的执行过程为：

(1)求表达式 1 的值。

(2)求表达式 2 的值，如果它的值为真(非 0)，则执行循环体语句，然后执行第(3)步；如果它的值为假(0)，则结束循环，执行 for 语句之后的语句。

(3)求表达式 3 的值，然后返回第(2)步重复执行。

在整个 for 循环过程中，表达式 1 只计算一次，表达式 2 和表达式 3 则可能计算多次。循环体可能多次执行，也可能一次都不执行。for 语句的执行过程如图 3-14 所示。

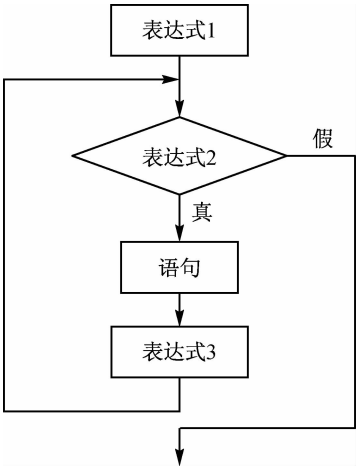


图 3-14　for 语句的执行过程

for 语句最简单、最易理解的应用形式如下：

```
for(循环变量初值;循环条件;循环变量增值)
  循环体语句
```

例如：

```
for(i=1;i<=10;i++)
sum=sum+i;
```

【例 3-28】 用 for 语句改写【例 3-24】。

参考程序如下：

```
/* Chap3_28.c:for 语句的使用 */
#include<stdio.h>
int main()
{
  int i,sum=0;
  for(i=1;i<=10;i++)
    sum=sum+i;
  printf("sum= %d\n",sum);
  return 0;
}
```

程序运行结果如下：

```
sum=55
```

本程序中，循环体语句只有一条，即 sum=sum+i。如果循环体是由多条语句构成，应该使用花括号括起来。

思考：如何用 for 语句计算 s=1+3+5+…+99?

【例 3-29】 计算 5!。

求一个整数 n 的阶乘值，就是将整数 1～n 这 n 个数连续相乘得到的结果。例如，计算 5!，5!=1×2×3×4×5，属于循环问题。

参考程序如下：

```
/* Chap3_29.c:计算 5! */
#include<stdio.h>
int main()
{
  int i=1;
  long fact=1; /* 变量 fact 用来表示阶乘值 */
  for(i=1;i<=5;i++)
    fact=fact*i;
  printf("%ld\n",fact);
  return 0;
}
```

程序运行结果如下：

```
120
```

本程序中的变量 fact 用来表示阶乘值，它的初始值必须为 1。循环体语句是 fact=fact*i，如果将它改为 fact=fact*5，那么程序运行结果又如何？请读者自己验证。

2. for 语句的变化

for 语句允许有多种变化形式，以增强其功能和灵活性。它的各个表达式都是任选项，都可以省略，但分号间隔符不能省略。

(1)在循环变量已赋初值时，可省去表达式 1。

```
int i=1,sum=0;
for(;i<=10;i++)
  sum=sum+i;
```

(2)如果表达式 2 省略，即不判断循环条件，循环会无限次地进行下去，构成无限循环。

```
int sum=0;
for(i=1;;i++)
{
  sum=sum+i;
}
```

相当于循环条件始终为真的无限循环：while(1) 。

(3)在循环体中改变循环变量的值，可省去表达式 3。

```
for(i=1;i<=10;)
{
  sum=sum+i;
  i++;
}
```

(4)3 个表达式都可省略，也是一个无限循环。

```
for(;;)
  语句
```

(5)循环体可以是空语句。空语句可以用来实现延时的功能。

```
for(;i<=10;i++);
```

注意:语句末尾的分号不能省略,它代表了一个空语句。

(6)表达式 1 和表达式 3 中可以用逗号表达式给与循环变量无关的变量赋值。例如:

```
for(sum=0,i=1;i<=10;i++)
  sum=sum+i;
```

或

```
for(i=0,j=10;i<=j;i++,j--)
  k=i+j;
```

(7)表达式 2 一般为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,只要其值为真,就执行循环体。例如:

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

表达式 2 先从终端接收一个字符赋给变量 ch,然后再判断表达式的值是否等于'\n'(换行符);如果不等于'\n',就执行循环体。

【例 3-30】 用 for 语句实现前 20 项的 Fibonacci 数。

本程序是要输出前 20 项的 Fibonacci 数,属于一个计数循环问题,采用 for 语句来实现比较方便。

参考程序如下:

```
/* Chap3_30.c:Fibonacci 数的输出 */
#include<stdio.h>
int main()
{
  int i;
  int fib1=1,fib2=1,fib;
  printf("%10d%10d",fib1,fib2);
  for(i=3;i<=20;i++)
  {
    fib=fib1+fib2;
    fib1=fib2;
    fib2=fib;
    printf("%10d",fib);
  }
  return 0;
}
```

程序运行结果如下:

```
     1         1         2         3         5         8        13        21
    34        55        89       144       233       377       610       987
  1597      2584      4181      6765
```

【例 3-31】 输入一行字符,统计其中字符的个数。

参考程序如下:

```
/* Chap3_31.c:统计字符的个数 */
```

```
#include<stdio.h>
int main()
{
  char ch;
  int i;
  for(i=0;(ch=getchar())!='\n';)
  {
    i=i+1;
  }
  printf("%d\n",i);
  return 0;
}
```

程序运行结果如下：

```
123456789ABCDEFG↙
16
```

说明：循环结束的条件是输入的字符是换行符('\n')。

3.4.5 循环中断控制语句

有两种循环中断控制语句：break 语句和 continue 语句。

1. break 语句

break 语句有以下两个作用：

(1)可以使程序流程跳出 switch 结构，继续执行 switch 语句下面的一个语句。

(2)强迫循环立即终止，使程序流程跳出循环结构，执行循环下面的语句。

break 语句的一般形式为：

```
break;
```

使用 break 语句可以使循环语句有多个出口，使程序避免一些不必要的重复，提高程序效率。

【例 3-32】 判断一个整数是否为素数。

素数是只能被 1 和它本身整除的整数。判断正整数 num 是否是素数，一种最直观的方法是在 2 到 num 之间能否找到一个整数 i 将 num 整除。若存在这样的整数 i，则 num 不是素数；若找不到这样的整数，则 num 是素数。这是一个穷举验证素数的算法。

这个问题可以用 for 语句来解决，表示除数的变量 i 的初值为 2；循环条件是 i<num；循环变量的改变是 i++。

参考程序如下：

```
/* Chap3_32.c:素数问题 */
#include<stdio.h>
int main()
{
  int num;
```

```
    int i,flag=1;                    /* flag是素数的标志 */
    scanf("%d",&num);
    for(i=2;i<num;i++)
    if (num%i==0)
    {
      flag=0;
      break;                         /* 跳出循环 */
    }
    if (flag)
      printf("%d是素数! \n",num);
    else
      printf("%d不是素数! \n",num);
    return 0;
}
```

程序运行结果如下：

```
13↙
13是素数!
```

2. continue 语句

continue 语句的作用是结束本次循环，即跳过循环体中下面尚未执行的语句，接着进行下一次循环条件的判断。

continue 语句的一般形式为：

```
continue;
```

continue 语句与 break 语句的区别是：continue 语句只结束本次循环，而不是终止整个循环的执行；而 break 语句则是结束整个循环过程，不再判断执行循环的条件。

【例 3-33】 输出 100 以内不能被 3 整除的数。

参考程序如下：

```
/* Chap3_33.c:输出100以内不能被3整除的数 */
#include<stdio.h>
int main()
{
  int num;
  for(num=1;num<100;num++)
  {
    if(num%3==0)
      continue;
    printf("%5d",num);
  }
  return 0;
```

```
}
```

程序运行结果如下：

```
 1   2   4   5   7   8  10  11  13  14  16  17  19  20  22  23
25  26  28  29  31  32  34  35  37  38  40  41  43  44  46  47
49  50  52  53  55  56  58  59  61  62  64  65  67  68  70  71
73  74  76  77  79  80  82  83  85  86  88  89  91  92  94  95
97  98
```

在循环体中，当 num 能被 3 整除时，执行 continue 语句，跳过其后的语句“printf("%5d", num);”结束本次循环，继续下次循环条件的判断。当 num 不能被 3 整除时，才执行语句“printf("%5d",num);”。

3.4.6 循环的嵌套

一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。内嵌的循环中还可以嵌套循环，这就是多层嵌套循环。

3 种循环语句都可以互相嵌套。例如，下面几种循环的嵌套都是合法的形式：

```
for(;;)
{
  for(;;)
  {
    …
  }
}
```

```
while()
{
  …
  while ()
  {
    …
  }
}
```

```
do
{
  …
  for(;;)
  {
    …
  }
}
while( );
```

【例 3-34】 用 for 循环输出以下图案：

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

该图案是由 4 行组成，每行又由 10 个星号(*)组成。可以用双重循环来解决该问题，外循环来控制图案的行数，内循环控制每行星号的个数。

参考程序如下：

```
/*Chap3_34.c:循环的嵌套*/
#include<stdio.h>
int main()
{
```

```
    int i,j;
    for(i=1;i<=4;i++)
    {
      for(j=1;j<=10;j++)
        printf("*");
      printf("\n");                    /* 换行 */
    }
    return 0;
}
```

程序中,“printf("\n");”这条语句表示换行,同内循环一起构成外循环体,即每一行上输出 10 个星号时,就要换行。注意它的位置一定要准确。

思考:如果要实现以下图案的输出,那么上面的程序又将如何修改?

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

【例 3-35】 打印九九乘法表。

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

该乘法表共有 9 行 9 列构成,但不是一个方阵,而是一个直角三角形,因此不仅要考虑如何实现 9 行的输出,更要思考每一行上所要显示的表达式的个数。从该乘法表可以清楚地看到第 1 行有 1 个表达式,第 2 行有 2 个表达式,第 3 行有 3 个表达式……第 9 行有 9 个表达式,从而得到每行上表达式的个数正好等于行数。

参考程序如下:

```
/* Chap3_35.c:打印九九乘法表 */
#include<stdio.h>
int main()
{
  int i,j;
  for(i=1;i<=9;i++)                    /*外循环 i 表示行数*/
  {
    for(j=1;j<=i;j++)                  /*内循环 j 表示列数*/
```

```
        printf("%d* %d=%-4d ",i,j,i*j);
      printf("\n");
    }
    return 0;
}
```

本程序中每一行上显示的表达式的个数等于行数，即内循环变量 j 的初始值是 1，循环条件是“j<=i”。

思考：如果将内循环的循环条件改变为“j<=9”，那么，程序的运行结果又将如何？

【例 3-36】 输出 100 以内的所有素数。

在【例 3-32】中，判断一个正整数是否是素数，采用的是循环结构，现在要测试 100 以内的所有整数，就需要采用双重循环来解决，即把判断一个整数是否为素数的问题作为内循环体来实现。

参考程序如下：

```
/* Chap3_36.c:输出 100 以内的所有素数 */
#include<stdio.h>
int main()
{
  int m,n,flag;
  for(n=2;n<100;n++)
  {
    flag=1;                  /* 设置素数的标志 */
    for(m=2;m<n;m++)
    {
      if(n%m==0)
      {
        flag=0;              /* 改变标志 */
        break;               /* 跳出内循环 */
      }
    }
    if(flag==0)
      continue;
    printf("%5d",n);
  }
  return 0;
}
```

程序运行结果如下：

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

【例 3-37】 百钱买百鸡。

百钱买百鸡问题的算法见图 3-11。

参考程序如下：

```
/* Chap3_37.c:百钱买百鸡 */
#include<stdio.h>
int main()
{
  int cocks=0,hens,chicks;
  printf("%10s%10s%10s\n","cocks","hens","chicks");
  while(cocks<=19)
  {
    hens=0;
    while(hens<=33)
    {
      chicks=100-cocks-hens;
      if(5*cocks+3*hens+chicks/3.0==100)
        printf("%10d%10d%10d\n",cocks,hens,chicks);
      hens++;
    }
    cocks++;
  }
  return 0;
}
```

程序运行结果如下：

cocks	hens	chicks
0	25	75
4	18	78
8	11	81
12	4	84

本 章 小 结

本章介绍了算法的概念以及结构化程序设计的 3 种基本结构——顺序结构、选择结构和循环结构。

在选择结构中，介绍了选择结构的 3 种基本类型：单分支选择结构、两分支选择结构和多分支选择结构。需要注意的是，不管程序中有多少个分支，只能选择一个分支执行。

在循环结构程序设计一节中，介绍了循环结构的概念以及循环结构在 C 语言中的实现方法，主要有 for 语句、while 语句和 do…while 语句。这 3 个语句可以处理相同的问题，一般情况下它们可以相互替代，但是要注意使用上的区别。其中，for 语句使用灵活而且应用广泛，它不仅可以实现计数循环，也可以实现非计数循环，几乎可以解决所有的循环问题。而 while 语句和 do…while 语句比较适合解决非计数循环问题。

遇到复杂的问题时需要使用循环的嵌套来解决，在使用循环的嵌套时，需要注意分清楚内层循环和外层循环的不同执行特点。在循环的过程中需要终止循环时，可以使用 break

语句或 continue 语句，同时需要注意它们之间的区别。

习 题 3

1. 什么是算法？常用的描述算法的工具有哪些？

2. 试用流程图表示求 3 个数中的最大数的算法。

3. 什么是结构化程序设计？它的主要内容是什么？

4. 从键盘输入 3 个整数，按照从小到大的顺序输出。

5. 输入一个华氏温度 F，要求输出摄氏温度 C。公式为：

$$C=\frac{5}{9}(F-32)$$

要求要有文字说明，取两位小数。

6. 输入一个 3 位整数，要求输出它的百位数、十位数和个位数。

7. 小红有面值 1 分的硬币 638 枚，可以兑换成几元几角剩余几分？

说明：100 枚 1 分的硬币可以兑换成 1 元，10 枚 1 分的硬币可以兑换成 1 角。兑换原则是面额从大到小。

8. 设圆柱的半径为 2.5，圆柱的高为 4，求圆柱的体积。用 scanf 函数输入数据，printf 函数输出计算结果，输出时要求有文字说明，取两位小数。

9. 输入三角形的三边，计算三角形的面积。要求：对输入的 3 个边长进行判断，如果能构成三角形，则计算三角形的面积；否则，则显示“构不成三角形！”信息。

10. 设计一个 C 语言程序，求 5 个数中的最大数和最小数。

11. 判断输入的字符属于哪一类字符：大写字母、小写字母、数字还是其他字符。

12. 用 if 语句和 switch 语句分别编写程序，实现以下功能：

从键盘输入数字 1，2，3，4，分别显示 Excellent，Good，Pass，Fail。输入其他字符时显示 Error。

13. 假设国家对个人收入所得税是按这样的标准规定的：起征点是 2 000 元，2 000～3 000 元为 5%，3 000～5 000 元为 15%，5 000～10 000 元为 20%，10 000 元以上为 30%。编程输入工资，计算实际工资所得及税金。

14. 求方程 $ax^2+bx+c=0$ 的根。

说明：当 $b^2-4ac>0$ 时，有两个不等的实根；当 $b^2-4ac=0$ 时，有两个相等的实根；当 $b^2-4ac<0$ 时，有两个虚根。

15. 从 1～100 中找出能被 5 或 7 整除的数。

16. 计算 sum=1+11+111+1111+11111。

17. 输入两个正整数 m 和 n，求出它们的最大公约数和最小公倍数。

18. 输入一行字符，分别统计出其中英文字母、空格、数字和其他字符的个数。

19. 从键盘输入 10 个学生的成绩，试统计出他们的成绩总和及平均值。

20. 猴子吃桃问题。猴子第一天摘下了若干桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第十天早上想要再吃时，只剩下一个桃子了。求第一天猴子共摘了多少桃子。

21. 打印出所有的“水仙花数”。所谓“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，153 就是一个水仙花数，因为 $153=1^3+5^3+3^3$。

22. 用泰勒级数求 e 的近似值，直到最后一项小于 10^{-6} 为止。

$$e=1+1/1!+1/2!+1/3!+\cdots+1/n!$$

23. 输入一个整数，分解各位数字。例如，输入“1298”，输出“8-9-2-1-”。请编写一个 C 语言程序解决该问题。

24. 有一个分数序列 2/1，3/2，5/3，8/5，13/8，21/13…求出这个数列的前 20 项之和。

25. 打印出以下图案。

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

26. 百马百担问题：有 100 匹马，驮 100 担货，大马驮 3 担，中马驮 2 担，两匹小马驮 1 担，问用大马、中马、小马各多少匹？请设计解决该问题的 C 语言程序。

27. 计算 1～10 十个整数的阶乘和，即计算 1!＋2!＋3!＋…＋9!＋10!。