第1章 C语言基础知识

本章将介绍的内容

基础部分：

- C程序的基本概念和上机步骤。
- 各种数据类型的常量和变量。
- 算术运算、赋值运算和逗号运算。

提高部分：

- Turbo C 2.0和Visual C++ 6.0集成环境。
- 进一步学习赋值运算符和自加、自减运算符。
- 进一步学习整型数据类型。

各例题的知识要点

例1.1 C程序形式和程序执行过程。

例1.2 主函数、函数体和输出控制。

例1.3 换行控制。

例1.4 数据类型的正确选用。

例1.5 常量和变量的概念。

例1.6 合法与非法的变量名。

例1.7 整型数据的各种进制输出。

例1.8 整型变量的定义和数据的溢出现象。

例1.9 实型常量的不同输出形式。

例1.10 实型变量的定义和有效数字。

例1.11 常规字符不同的格式输出以及字符常量的算术运算。

例1.12 特殊字符的输出和转义字符的概念。

例1.13 字符型变量的定义和赋值。

例1.14 将代数式转化为C语言表达式。

例1.15 强制类型转换。

例1.16 逗号表达式。

例1.17 自加、自减运算符。

（以下为提高部分例题）

例 1.18　复合赋值运算符。

例 1.19　由自加、自减运算符构成的表达式。

例 1.20　复杂形式的逗号表达式。

例 1.21　整型常量的各种进制表示法。

例 1.22　各种整型数据的输出。

例 1.23　用图理解基本型整数的变化情况。

1.1　C 语言概述

1.1.1　什么是 C 语言

人和人之间交换信息需要借助于语言工具，人和计算机交换信息也同样要用语言工具。我们将后一种语言称为计算机语言。用计算机语言编写的代码称为程序。

随着计算机技术的发展，计算机语言逐步得到完善。最初使用的计算机语言是用二进制代码表达的语言——机器语言。后来采用与机器语言相对应的助记符表达的语言——汇编语言。虽然用这两种语言编写的程序执行效率高，但程序代码很长，又都依赖于具体的计算机，因此编码、调试、阅读程序都很困难，通用性也差。我们称这两种语言为低级语言。现在使用最广的计算机语言是高级语言——更接近于人们自然语言的表达语言。高级语言独立于机器，编码相对简短，可读性强。FORTRAN、QBASIC、Pascal、COBOL、C 等都是高级语言。用高级语言编写的程序叫做源程序。由于计算机只能识别 0 和 1，因此源程序必须通过编译和连接后，才能被计算机执行。

C 语言比其他高级语言功能更强，它既具有高级语言的功能也具有低级语言的许多功能，即具有双重性，因此有的书把 C 语言称为中级语言。C 语言是由附录 A 中列出的 32 个关键字再加上语法规则构成的。

要得到 C 程序的运行结果，首先将源程序输入计算机（在计算机上输入或修改源程序的过程叫做编辑），然后还要把源程序翻译成机器能识别的目标程序，如在 Turbo C 2.0 集成环境中，将编辑后的源文件保存为 e1.c，则通过编译产生的与源文件相对应的目标程序为 e1.obj；但是，目标程序不是可执行文件，不能直接运行，还要把目标程序和系统提供的库函数等连接起来产生可执行文件 e1.exe，这时我们才可以运行程序，并看到运行结果。C 程序的编辑、编译、连接、运行过程可用图 1.1 表示。

图 1.1　C 程序的编辑、编译、连接、运行过程

C 程序的编辑、编译、连接、运行过程可以在不同的环境中进行，而本书中的所有例题是在 Turbo C 2.0 集成环境下运行通过的。

1.1.2 C 程序形式和程序执行过程

下面举一个 C 语言程序的完整例题，用此例说明 C 程序的一般形式和程序的执行过程。程序中的具体语法规则和其他细节将在后续章节中陆续介绍。

【例 1.1】 编写一个完整的 C 语言程序示例。

【解】 程序如下：

```
#include <stdio.h>                          /* 包含文件 */
#include <math.h>                           /* 包含文件 */
int mysum(int m,int n);                     /* 函数原型说明 */

main( )                                     /* 主函数首部 */
{    int a,b,x;                             /* 声明部分 */
     double c,y,z;                          /* 声明部分 */

     c = 4.0;                               /* 以下 6 行均为语句部分 */
     y = sqrt(c);
     a = 10; b = 20;
     x = mysum(a,b);
     z = x + y;
     printf("z = %lf\n",z);
}                                           /* 主函数到此结束 */

int mysum(int m,int n)                      /* mysum 函数首部 */
{    int k;                                 /* 声明部分 */

     k = m + n;                             /* 以下 2 行均为语句部分 */
     return k;
}
```

运行结果：

```
z = 32.000000
```

程序说明：

(1) 正如本例所示，C 语言程序是由若干函数构成的，函数中至少包含一个主函数，C 程序从主函数开始执行，主函数名必须是 main。例 1.1 中程序的执行过程如图 1.2 所示，程序按①到⑨的顺序执行。

(2) 程序中用“/* …… */”括起来的部分是注释部分，该部分对程序的运行无任何作用，注释的目的是方便阅读程序。注释可以出现在程序的任何位置。

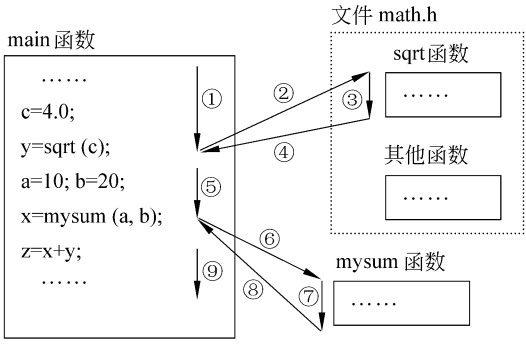


图 1.2　例 1.1 程序的执行过程

1.2　简单 C 程序与上机步骤

本节将给出几个最简单的 C 程序，通过这些例题，介绍 C 语言的基本概念，以及在 Turbo C 2.0 集成环境下的上机步骤。

1.2.1　简单 C 程序

【例 1.2】　编写在屏幕上显示一个句子“Let's study the C language.”的程序。

【解】　程序如下：

```
#include <stdio.h>
main( )
{
    printf("Let's study the C language.");
}
```

运行结果：

```
Let's study the C language.
```

程序说明：

(1) 程序中 main 是主函数名，每一个 C 程序都必须包含而且只能包含一个主函数。用一对大括号({ })括起来的部分是函数体。本例函数体中只有一条语句“printf("Let's study the C language.");”，此语句是输出语句，其作用是按原样输出双引号内的字符串“Let's study the C language.”(详见 2.3.1 节)。语句最后的“;”不能丢。

(2) C 语言中英文大小写字母被认为是不同的字符，即 main 不能写成 Main，printf 也不能写成 Printf。若程序中有此类错误，则很难发现。C 语言中的所有关键字都用小写字母。

(3) 程序中的“#include <stdio.h>”称为命令行，有了此行，就可以成功地调用 C

语言标准库中提供的输入、输出函数，所以编写程序时，在程序的第一行都写此行。

【例 1.3】 编写输出两个句子“Let's study the C language.”和“It's interesting.”的程序。

【解】 程序如下：

```
#include <stdio.h>
main( )
{
    printf("Let's study the C language.\n");      /* 输出字符串后换行 */
    printf("It's interesting.\n");
}
```

运行结果：

```
Let's study the C language.
It's interesting.
```

程序说明：

(1) 本程序的函数体包括两条输出语句，函数体可以包括任意多条语句。

(2) “\n”是换行符，如果程序中去掉“\n”，输出形式则为：

```
Let's study the C language. It's interesting.
```

【讨论题 1.1】 程序中最后一个输出语句中的换行符“\n”的作用是什么？

小结：

(1) C 程序一定有且仅有一个主函数，主函数名必须是 main，其后的圆括号内可以是空的，但圆括号不能省略。

(2) 函数体中可以有多条语句，每条语句都用分号结束。

(3) 注释用“/* …… */”形式，注意，“/”和“*”之间不能加有空格。为了提高程序的可读性，建议加上必要的注释。

(4) C 程序的书写格式比较自由。例如，一行内可以包括多条语句，一条语句也可以写在多行上，每行的内容可以从任何一列开始写。但是建议初学者每行写一条语句，而每行的语句根据需要适当向右缩进几列(参见例 4.19)，这对读懂程序和调试程序很有帮助。本书中的所有程序都按最常用的书写格式，请读者参考。

1.2.2 上机步骤

下面在 Turbo C 2.0 集成环境下，给出例 1.2 和例 1.3 的上机步骤。

先介绍例 1.2 的上机步骤。

第 1 步：在 c 盘 TC 目录下安装 Turbo C 2.0。如果已经安装，则跳过此步。

第 2 步：进入 Turbo C 集成环境。在 Windows 的资源管理器中，双击 tc.exe 文件可进入 Turbo C 集成环境，按 Alt+Enter 组合键，可使屏幕变成最大，再按一次此组合键，可以恢复原屏幕大小(详见 1.6.1 节)。

第 3 步：编辑源程序。如果主菜单处于被激活状态，则按 Esc 键。在编辑区里输入例 1.2 中的程序，如果输入有错，修改之。

第 4 步：保存文件。因为上机时经常会发生预料不到的事情，一定要养成随时存盘的好习惯。按 F10 键，激活主菜单，在 File 菜单下选择 Write to，在出现的输入框内输入“d:\cwz\exmp1_2.c”（假设已在 d 盘根下建立 cwz 文件夹），如果只输入文件名，则将文件存放在当前盘当前目录下。文件名的默认扩展名为“.c”。File 菜单中的 Write to 命令一般用于第 1 次存文件或改名存盘，而 Save 命令则用于按原名存盘。

第 5 步：编译和连接程序。C 源程序通过编译和连接之后才能运行。按 F9 键可先将程序编译成目标程序 exmp1_2.obj，再连接产生可执行文件 exmp1_2.exe。如果编译或连接时没有错误，则执行第 6 步，否则回到第 3 步。修改之后再进行存盘操作，由于不是第 1 次存盘，选择 Save 命令，F2 是 Save 命令的热键。当需要在编辑区和消息区之间切换，则按 F6 键。

第 6 步：运行程序。按 Ctrl＋F9 组合键，系统就会运行 exmp1_2.exe。如果运行时发现有错误，回到第 3 步，否则转到第 7 步。

第 7 步：查看运行结果。正常运行程序后，屏幕上仍显示 Turbo C 的窗口，为了查看运行结果，一定要切换到 DOS 屏幕。按 Alt＋F5 组合键或选择主菜单中 Run 菜单下的 User screen 命令，Turbo C 集成环境将切换到 DOS 屏幕（黑色屏幕），并显示运行结果，即显示：“Let's study the C language.”。

按任意键，回到 Turbo C 环境，如果运行结果与预期的结果不相符，则重复第 3 至 7 步，直到结果正确为止。

在进行例 1.2 的上机操作后，不必退出 Turbo C 环境，可继续进行例 1.3 的上机操作，其步骤如下：

第 1 步：新建文件。选择 File 菜单下的 New 命令。

注意：*在例 1.2 下方接着输入例 1.3 是错误的，因为系统认为这是一个 C 程序，而一个程序中不能有两个主函数。这是初学者中容易出现的错误。*

第 2 步：编辑源程序。输入例 1.3 中的程序，如果输入有错，修改之。

第 3 步：保存文件。在 File 菜单下选择 Save 或按 F2 键，第 1 次存盘也可以使用此命令。在出现的输入框内输入“d:\cwz\exmp1_3.c”。

第 4 步：编译、连接和运行程序。按 Ctrl＋F9 组合键。系统先将程序编译成目标程序“exmp1_3.obj”，再连接产生可执行文件“exmp1_3.exe”，最后运行“exmp1_3.exe”，如果编译、连接或运行时没有错误则转到第 5 步，否则回到第 2 步。这种操作方式比较简便，它将编译、连接和运行过程统一进行，但系统不显示编译时的警告错误，而警告错误也是不容忽视的，因此建议读者今后尽量按例 1.2 中的第 5、6 步操作。

第 5 步：查看运行结果。按 Alt＋F5 组合键，在 DOS 屏幕上将显示：

```
Let's study the C language.
It's interesting.
```

按任意键，回到 Turbo C 环境，如果运行结果与预期的结果不相符，则重复第 2 至 5

步，直到结果正确为止。

注意：第 1 次运行例 1.3，第 1 行的输出结果没有从第 1 列开始，这是因为在前次的运行程序中（例 1.2），输出字符串的最后没有“\n”，致使输出操作完成后，光标不能返回到屏幕的首列，例 1.3 的第 1 个输出就从光标当前停留位置开始。由于例 1.3 第 2 条输出字符串后面有“\n”，再次运行时，可以使光标返回到屏幕首列，以后的输出就从第 1 列开始。

第 6 步：退出 Turbo C 环境。按 Alt＋X 组合键可以退出 Turbo C 环境，如果程序修改后没有存盘，系统将会给予存盘提示，否则直接回到 Windows 界面。

说明：

如果在例 1.2 程序的输出语句尾部加“\n”，并在其后输入语句“printf("It's interesting.\n");”，再用 Write to 命令将文件 exmp1_2.c 另存为 exmp1_3.c，同样可以达到第 1～3 步的效果。

如果在 Turbo C 环境中，选择 File 菜单下的 Load 命令，并在出现的输入框内输入“d:\cwz\exmp1_2.c”，可以直接打开例 1.2 程序文件。

如果在选择 Load 命令后，输入原来不存在的文件名，则可新建一个 C 文件。

在 1.6.1 节和 1.6.2 节中将分别介绍 Turbo C 2.0 和 Visual C++ 6.0 集成环境，请读者参考。

1.3 数 据 类 型

C 语言里常用的数据类型有整型、实型、字符型、数组、结构体类型、指针类型等。在程序设计中，根据不同的需要正确选用数据类型是至关重要的。本节将通过一个例题说明这一点。

【例 1.4】 编写输出 5 和 6 的和与平均值的程序。

【解】 程序如下：

```
#include <stdio.h>
main( )
{    int a,b,sum;                    /* 指定变量 a、b、sum 为整型 */
     float ave;                      /* 指定变量 ave 为实型 */

     a = 5;                          /* 给整型变量 a 赋值 5 */
     b = 6;                          /* 给整型变量 b 赋值 6 */
     sum = a + b;                    /* 将 a 中值 5 和 b 中值 6 之和 11 赋给整型变量 sum */
     ave = sum/2;                    /* 将 sum 中值 11 和 2 之商 5 赋给实型变量 ave */
     printf("sum = %d,ave = %f\n",sum,ave);   /* 按指定格式输出 sum 和 ave 的值 */
}
```

运行结果：

```
sum = 11,ave = 5.000000
```

注意：输出结果不是：sum＝11,ave＝5.500000。

程序说明：

(1) a、b、sum 和 ave 是变量，其中 a、b、sum 是整型变量，而 ave 是实型变量，整型变量中只能存放整型值，实型变量中只能存放实型值。

(2) 为什么输出结果不是 sum＝11，ave＝5.500000 呢？原因是程序中的第 8 行上 sum 中值 11 和除数 2 都是整数，而在 C 语言里两个整数的商仍为整数，表达式 11/2 的值为 5。由于 ave 是实型变量，因此其中只能存放实型数 5.0 而不能存放整型数 5(参见 1.4.3 节)。如果将此行改成“ave＝sum/2.0;”，则输出：sum＝11，ave＝5.500000。

(3) 最后一条语句“printf("sum＝%d,ave＝%f\n",sum,ave);”与例 1.2 中的输出语句格式不同。本语句的作用是按原样输出双引号内除%d 和%f 以外的内容，而在%d 的位置上输出 sum 的值，%f 的位置上输出 ave 的值(小数点后保留 6 位)。%d 和%f 是输出函数的格式说明，分别用于输出整型数和实型数(详见 2.3.1 节)。

小结：

在处理数据和输出数据时，一定要选择合适的数据类型和正确的输出格式说明，否则将得到错误的运行结果或程序出错。如果在上面程序中将 sum 的数据类型改为实型，ave 的数据类型改为整型，则在执行“ave＝sum/2;”后，sum/2 的值为 5.5，但 ave 值为 5。

1.4 常量与变量

C 语言中的数据有常量与变量之分。

1.4.1 常量与变量的概念

【例 1.5】 编写输出 1000 和 100 的和与差的程序。

【解】 程序如下：

```
#include <stdio.h>
#define FIRST 1000                  /* 定义符号常量 FIRST */

main( )
{    int s;                         /* 定义整型变量 s */

     s = FIRST + 100;               /* 相当于 s = 1000 + 100; */
     printf("The sum is %d\n",s);
     s = FIRST - 100;               /* 相当于 s = 1000 - 100; */
     printf("The difference is %d\n",s);
}
```

运行结果：

```
The sum is 1100
The difference is 900
```

程序说明：

(1) 程序中第 1 行的作用是用 #define 命令行定义符号名 FIRST，并用它代表 1000。以后在本程序中出现的所有 FIRST 都代表 1000。如果在一个程序中经常使用同一个量，而且这个量很长，则用符号名代替此量比较方便，当然也有其他用处(如例 5.1)。请读者注意 #define 命令行的定义形式和位置。

(2) 程序中 FIRST(代替 1000)和 100 是固定不变的，而 s 中的值可以改变，执行语句“s=FIRST+100;”后，s 得到两数之和 1100，执行语句“s=FIRST-100;”后，s 的值又由 1100 变为 900。

以上例题中出现了两种量，一种是在程序运行过程中其值不能变的量，如 100，这种量称为常量；FIRST 在程序运行过程中其值也不能改变，因此也是常量，这种用符号名表示的常量称为符号常量；另一种是在运行过程中其值可以改变的量，如 s，这种量称为变量。注意，由于 FIRST 是常量，企图通过语句“FIRST=5;”给它赋一个新值是错误的。

每个变量都应该有一个名字，其名字由用户命名。变量的命名规则如下：

(1) 变量名由 a～z、A～Z、0～9、_(下划线)组成，并区分大小写。

(2) 变量名的第 1 个字符不能是数字。

(3) C 语言中的关键字不能作为变量名。C 语言中的关键字见附录 A。

变量名是一个标识符。在 C 语言中，用来标识变量名、符号常量名、标号名、数组名、函数名、类型名、文件名等的有效字符序列称为标识符，这些标识符的命名规则与变量名的命名规则相同。标识符一般使用小写字母。

【例 1.6】 下面变量名中哪些是合法的，哪些是不合法的？

Int、float、_123、9k、qbasic、printf、a.b、year、business

【解】 合法的变量名有 Int、_123、qbasic、printf、year、business。

注意：printf 也可作为变量名使用，在 C 语言中 printf(库函数名)、define(预编译处理命令)等等都有特定的含义，称为预定义标识符，但它们不是关键字，C 语法允许作为自定义标识符，但这样做将使它们失去系统规定的原意，例如：若将 printf 定义为变量名，系统就不能再通过 printf 调用输出函数。因此我们不提倡将预定义标识符改作为自定义标识符使用。

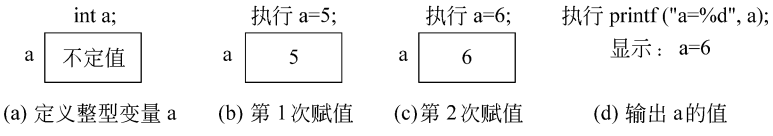
不合法的变量名有 float(关键字)、9k(数字打头)、a.b(出现非法字符)。

说明：

(1) 变量必须先定义后使用。变量就像一个可以存放“物品”的容器，定义的含义如同制造容器，通过定义的变量，系统就会为其分配一个存储单元。因此变量的定义部分必须放在本函数体内其他语句之前。

(2) 应该注意，变量中存放的“物品”只能是数据，而且只能是一个数据。往变量中存放数据的操作称为赋值，只经定义而未赋值的变量，其值是不确定的。

可以给同一个变量多次赋值，每进行一次赋值操作，系统都会用新数据替代变量中的原有数据，因此变量的值应该是最后一次存放的数据。图 1.3 表示了一个变量的定义、多次赋值以及输出的全过程。



(a) 定义整型变量 a　(b) 第 1 次赋值　(c)第 2 次赋值　(d) 输出 a的值

图 1.3　变量的定义、赋值和输出

由于只经定义而未赋值的变量，其值不确定，为了防止使用这无意义的数据，建议对那些暂时不必赋值的变量均赋 0，如“int a＝0;”。在定义变量的同时给变量赋值，如“int a＝5;”称为变量的初始化。“int a＝5;”的作用与“int a; a＝5;”相同。C 语言允许在同一个定义部分中定义多个变量，并同时对它们进行初始化。

(3) 变量的“名”和变量的“值”有区别。变量的“名”是该变量所代表的存储单元的标志，而变量的“值”是指存储单元中的内容。

【讨论题 1.2】 在 C 语言中“int a＝5，b＝6;”是合法吗？“int a＝5; b＝6;”也合法吗？

【讨论题 1.3】 假设有“int a; a＝5;”，则“a＝a＋10;”是合法的语句吗？该语句的两个 a 中，哪个 a 代表变量的“名”，哪个 a 代表变量的“值”？

常量和变量都有类型的区分，如整型、实型、字符型等，下面分别介绍。

1.4.2　整型常量与变量

1. 整型常量

【例 1.7】 编写程序，将一个整型常量分别按十进制、八进制、十六进制形式输出。

【解】 程序如下：

```
#include <stdio.h>
main( )
{  printf("Decimal: %d Octal: %o Hexadecimal: %x\n",21,21,21);
}
```

运行结果：

```
Decimal: 21 Octal: 25 Hexadecimal: 15
```

程序说明：

程序中%d、%o、%x 是格式说明符，分别表示按十进制、八进制、十六进制形式输出整型常量 21。

【讨论题 1.4】 语句“printf("%d,%x",17,17);”的运行结果是什么？

整型常量的常用类型有基本型和长整型，其数值范围如下：

整型常量{基本型，－32768～32767(即$-2^{15}\sim 2^{15}-1$)，
长整型，－2147483648～2147483647(即$-2^{31}\sim 2^{31}-1$)。

在C语言中的整型常量常按十进制形式(即用%d)输出。有关整型数据的进一步讨论请参见1.6.5节。

2. 整型变量

整型变量中只能存放整型数据。整型变量的常用类型有基本型和长整型，定义变量时必须根据需要给出其类型。

【例1.8】 编写一个整型变量的定义、赋值和输出程序。

【解】 程序如下：

```
#include <stdio.h>
main( )
{    int a = 0,b = 0;                          /* 定义基本型变量a和b */
     long int c = 0;                           /* 定义长整型变量c */

     a = 32767;
     b = 32768;
     c = 32768;
     printf("a = %d,b = %d,c = %ld\n",a,b,c);   /* %ld中的l是long的第1个字母 */
}
```

运行结果：

```
a = 32767,b = -32768,c = 32768 (注：b的值不是32768,产生溢出现象)
```

程序说明：

(1) 定义整型变量要用关键字int，定义长整型变量则用关键字long int或者long，例如：程序中的定义部分“long int c=0;”，也可简写成“long c=0;”。

(2) %d、%ld为输出格式说明，分别用于输出基本型和长整型数据。

(3) 由于32767是整型数的最大取值，32767+1的值产生溢出现象，因此得不到我们预期的值32768，其具体值的产生参见1.6.5节。

(4) 程序中定义了变量a、b、c，其名称由编程者给出。在实际开发软件时，为了增加程序的可读性，命名变量时应做到变量名“简单明了”、“见名知意”。例如，表示年份的变量名可用iYear，表示人口的变量名可用lPopulation等，其中第1个字母i和l分别表示该变量的数据类型，即基本型和长整型数据。

处理整型数据应该注意以下问题：为了防止产生数据溢出的现象，必须先估计所要处理的数据范围，再根据其范围选择合适的数据类别(基本型或长整型)，还要根据数据的类型选择与其匹配的格式说明符(%d还是%ld)，否则都将得不到正确的结果。

如果以后没有特别声明，在本书中整型变量指基本整型变量。

整型变量可以精确地存放数据，但取值范围较小。在表1.1中列出了变量a的定义形式、a在一般微型机上所占的字节数和a的取值范围。

表 1.1　整型变量的定义

a 的定义形式	a 的类型	字节数	a 的数据范围
int a;	基本型	2	−32768≤a 中的值≤32767
long [int] a;	长整型	4	−2147483648≤a 中的值≤2147483647

1.4.3　实型常量与变量

1. 实型常量

在 C 语言中将实型常量作为双精度类型处理，双精度类型能保证数据的前 15 位准确无误，因此其有效数字至少为 15。双精度类型数据的取值范围是 -1.7×10^{308} ～ -1.7×10^{-308} 或 1.7×10^{-308} ～ 1.7×10^{308}。

【例 1.9】 编写程序，将实型常量按小数形式和指数形式输出。

【解】 程序如下：

```
#include <stdio.h>
main( )
{  printf("%f--%e,%e\n",123451234512345.1,12345.6788885,0.0); }
```

运行结果：

```
123451234512345.094000--1.23457e+04,0.00000e+00
```

程序说明：

(1) 实型常量有两种不同形式的输出，使用%f，按十进制小数形式输出，小数点后有 6 位(对第 7 位四舍五入)，使用%e，按指数形式输出，小数点前有一位非零数字，小数点后输出 5 位(对第 6 位四舍五入)，如果用%e 格式说明符输出 0.0，则输出形式为 0.00000e+00。

(2) 由第 1 个实型常量的输出结果可以看出，前 15 位的数字准确无误。

在 C 语言中实型常量有两种表示形式，即十进制小数形式和指数形式。

(1) 十进制小数形式必须包括数字和小数点。如 12.3、.12、12.、0.0 都是合法的十进制小数形式，而 123 和.(只有小数点)是非法的。

(2) 指数形式类似于数学中的科学记数法。如 3.15e1、315e−1、0.315E+2 相当于 3.15×10、315×10^{-1}、0.315×10^{2}，而且都表示 31.5。使用指数形式时要注意 e(或 E)前面必须有数字，且 e 后面的指数必须为整数，如 E−1、3e1.1、.3e1、e 等都是不合法的指数形式，而 3.e2、1E+3 都是合法的。

2. 实型变量

实型变量中只能存放实型数据，内存中的存储形式一律以指数形式存放。实型变量按其所能保证的精度分为单精度型和双精度型。

【例 1.10】 编写一个实型变量的定义、赋值和输出程序。

【解】 程序如下：

```
#include <stdio.h>
main( )
{    float a = 12.3,b = 0.0;                    /* 定义 2 个单精度型变量 */
     double c = 12345.67;                       /* 定义双精度型变量 */

     b = 12345.67;                              /* 此行不能放在前一行前面 */
     printf("a = %f,b = %f,c = %lf\n",a,b,c);
}
```

运行结果：

```
a = 12.300000,b = 12345.669922,c = 12345.670000
```

程序说明：

(1) %f 用于输出单精度型和双精度型数，而%lf 用于输出双精度型数。不论用%f 还是%lf，都输出 6 位小数。

(2) 单精度型变量存放数据时，能保证 6 位有效数字，而双精度型变量能保证 15 位有效数字。由运行结果可以看出，b 保证了前 6 位数字(从第 7 位起数字已不准确)，而 c 保证了所有数字(因为不大于 15 位)。

(3) 编写程序时，建议用 f 作为变量的第 1 个字母，表示该变量为单精度型，用 d 作为变量的第 1 个字母，表示该变量为双精度型。

实型变量取值范围较大，但由于有效数字以外的数字不能保证，无法精确地存放数据，往往出现误差。在表 1.2 中列出了变量 a 的定义形式、a 在微型机上所占的字节数和 a 的取值范围。

表 1.2　实型变量的定义

定义形式	a 类型	字节数	有效数字/位	a 的取值范围
float a;	单精度	4	6 位	$-3.4\times10^{-38}\leqslant$ |a 中的值| $\leqslant 3.4\times10^{38}$
double a;	双精度	8	15 位	$-1.7\times10^{-308}\leqslant$ |a 中的值| $\leqslant 1.7\times10^{308}$

1.4.4　字符型常量与变量

1. 字符型常量

【例 1.11】 编写程序，将常规字符按不同格式输出。

【解】 程序如下：

```
#include <stdio.h>
main( )
{    printf("%c---%d,%c---%d\n",'a','a','A','A');
     printf("%d---%c,%d---%c\n",'a'+1,'a'+1,'A'+1,'A'+1);
     printf("The value of \'a\'-\'A\' is %d.\n",'a'-'A');
}
```

运行结果：

```
a---97,A---65
98---b,66---B
The value of 'a'-'A' is 32.
```

程序说明：

(1) 格式说明%c用于输出一个字符，字符常量用单引号括起来，但输出时不输出单引号。字符也可以按整型形式输出，而且输出的是该字符的ASCII码值。字符a和A的ASCII码值分别为97和65(参见附录B)。

(2) 字符常量可以参加运算，表达式'a'+1的值是字符“a”的ASCII码值97和1之和98(即“b”的ASCII码值)，而'A'+1的值是“A”的ASCII码值65和1之和66(即“B”的ASCII码值)。

(3) 小写字母与其相对应的大写字母的ASCII码值之差都是32，因此将字母“H”可表示为'h'-32，字母“m”可表示为'M'+32。

【讨论题1.5】 已知字母“g”的ASCII码值为103，如何计算字母“L”的ASCII码值？

【例1.12】 编写程序，将特殊字符显示在屏幕上。

【解】 程序如下：

```
#include <stdio.h>
main( )
{
    printf("%c---%c\n",'\1','\x1');
    printf("%c---%c\n",'\25','\x15');
    printf("I am \"OK\"\n");
}
```

运行结果：

```
☺---☺
§---§
I am "OK"
```

程序说明：

字符常量是ASCII字符集中的一个字符，但其中有些字符在键盘上找不到，例如，字符☺、§等。在程序中使用这些字符的方法是在“\”后加该字符的八进制ASCII码值(最多3位)或在“\x”后加该字符的十六进制ASCII码值(最多两位)。如果需要输出双引号“"”，则必须写成“\"”。例如，程序中若将最后一条输出语句写成“printf("I am "OK"\n");”是错误的。

用“\”开头的字符序列称为转义字符，表1.3列出了常用转义字符。

综上所述，字符常量有两种形式，一是常规字符(用单引号括起来的单个字符)，二是转义字符(用“\”开头的字符序列)。字符常量可以按其ASCII码值参加整数运算，即字符数据与整型数据可以通用。

表 1.3　常用转义字符

转义字符形式	转义字符功能	十进制 ASCII 码值
\n	换行	10
\t	横向跳格(即跳到下一个输出区)	9
\v	竖向跳格	11
\b	退格	8
\r	回车	13
\f	走纸换页	12
\\	反斜杠字符	92
\'	单引号字符	39
\"	双引号字符	34
\ddd	1 到 3 位八进制数所代表的字符,如\1 代表☺	1(☺)
\xhh	1 到 2 位十六进制数所代表的字符,如\x15 代表§	21(§)

2. 字符型变量

字符型变量中存放 ASCII 字符集中的任何一个字符,字符变量在内存中占一个字节。

【例 1.13】 编写一个字符型变量的定义、赋值和输出程序。

【解】 程序如下:

```
#include <stdio.h>
main( )
{    char c1 = '\0',c2 = 'A',c3 = '\0',c4 = '\0';      /* 定义 4 个字符型变量 */
     int sum = 0;

     c1 = 65;                                          /* 十进制数赋给字符型变量 c1 */
     c3 = '\101';                                      /* 用八进制表示的转义字符赋给 c3 */
     c4 = '\x41';                                      /* 用十六进制表示的转义字符赋给 c4 */
     sum = c2 + 30;
     printf("c1 = %c,c2 = %c,c3 = %c,c4 = %c\n",c1,c2,c3,c4);
     printf("c1 = %d,c2 = %d,c3 = %d,c4 = %d\n",c1,c2,c3,c4);
     printf("sum = %d\n",sum);
}
```

运行结果:

```
c1 = A,c2 = A,c3 = A,c4 = A
c1 = 65,c2 = 65,c3 = 65,c4 = 65
sum = 95
```

程序说明:

(1) 将常规字符或转义字符赋给字符变量时，都需要在其两侧加单引号，但将整型数据(实际上是某字符的 ASCII 码值)赋给字符变量时，不能加单引号。

(2) 转义字符“\0”的 ASCII 码值为 0。

(3) 由于字符按其 ASCII 码值参加整数运算，因此可将其运算结果直接赋给整型变量。

当把一个字符存放到字符变量中时，实际上系统将该字符的 ASCII 码值存放在变量所代表的存储单元中，即变量中的内容为该字符的 ASCII 码值，因此字符变量可以参加整型数据的任何运算。

(4) 编写程序时，建议用 c 作为变量的第 1 个字母，表示该变量为字符型。

1.5 运算符和表达式

C 语言提供非常丰富的运算符(参见附录 A)，并由这些运算符组成相应的表达式。请记住，任何 C 语言表达式都有一个确定的值。本节将介绍算术运算符、赋值运算符、逗号运算符、自加(减)运算符等以及其相应的表达式，而其他运算符和表达式将在后面陆续介绍。

1.5.1 算术运算符和表达式

1. 算术运算符

C 语言中提供的算术运算符如表 1.4 所示。

表 1.4 算术运算符

运算符	含义	优先级的数值	运算量类型	举 例
+	加	4	整型或实型	2+3.5 的结果为 5.5
−	减	4	整型或实型	10−5.0 的结果为 5.0
*	乘	3	整型或实型	3*4 的结果为 12
/	除	3	整型或实型	1/2 的结果为 0，1/2.0 的结果为 0.5
%	求余	3	整型	5%2 的结果为 1，2%5 的结果为 2
−	负号	2	整型或实型	−11.3

从上面例题可以看到，如果 +、−、*、/ 运算符的两侧运算量都是整型，则按整型计算，且运算结果为整型，例如，1/2 的结果是 0，不是 0.5；如果两侧运算量中至少有一个运算量为实型，则先将两个运算量都转化为双精度型后计算，且运行结果为双精度型。另外运算符“%”两侧的运算量必须是整型。

使用算术运算符时应注意，在进行 +、−、*、/ 算术运算时，系统自动先将数据的类型按一定的规则转换(即统一到同一种数据类型)，然后再进行运算，运算结果类型是转换

后的类型。图 1.4 给出了数据类型的转换规则，可根据“垂直降落，向上位移”的原理辅助理解。

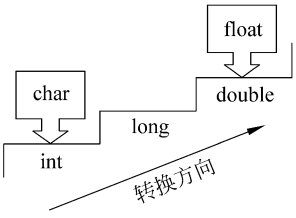


图 1.4　转换规则

(1) 垂直降落

如果运算量是 char 型，则必须先转换成 int 型，如果运算量是 float 型，则必须先转换成 double 型，然后再进行下步运算。char 型的指定地点是位于阶梯最低级的 int 型，而 float 型的指定地点是位于阶梯最高级的 double 型。

(2) 向上位移

将 int 型、long 型和 double 型看成是由低到高的 3 个台阶，如果数据类型相同，也就是处于同一阶层，则系统不进行转换，而直接运算，其运算结果类型是该数据类型。如果数据类型不同，即处于不同阶层，则系统将其中低级别类型的数据统一到高级别类型，然后再运算，而且其运算结果类型是高级别类型。

类型转换通常一步到位。例如，int 型或 long 型与 double 型进行运算，则先将 int 型或 long 型直接转换为 double 型，然后再对两个 double 型数据进行运算，其运算结果为 double 型。再如，int 型与 long 型进行运算时，要将 int 型转换为 long 型，接着再对两个 long 型数据进行运算，其运算结果为 long 型。

另外，如果两个运算量是 int 型和 float 型，由于系统必须先把 float 型转换为 double 型，因此进行运算之前 int 型也要随之转换为 double 型，其运算结果也为 double 型。

【讨论题 1.6】 运算量是 char 型和 float 型时，系统应如何处理？

2. 算术表达式

－2＊((a＋sqrt(4.0))－1) 是一个算术表达式，其中 sqrt(4.0)是利用系统提供的求平方根函数计算 4.0 的平方根值。在 C 语言中，将由算术运算符、圆括号和运算对象(包括常量、变量、函数等)组成，且符合 C 语言语法规则的表达式称为算术表达式。算术表达式的计算结果是一个数值。

算术表达式的计算示例，用表 1.5 给出。

表 1.5　算术表达式的计算

算术表达式	运 算 过 程	算术表达式的值
－2＋18/3＊5%8	第 1 步：(－2)＋6＊5 %8 第 2 步：(－2)＋30%8 第 3 步：(－2)＋6 第 4 步：4	4
a＊((6＋sqrt(9.0))/2) (假设 a 的值为 5)	第 1 步：5＊((6＋3.0)/2) 第 2 步：5＊(9.0/2) 第 3 步：5＊4.5 第 4 步：22.5	22.5

说明：

(1) 一个算术表达式中可以有多个运算符，因此对算术表达式进行计算时，要注意运

算的先后顺序。在附录C中列出了C语言中所有运算符的优先级和结合方向。优先级的数值越小，运算顺序就越在先，也就优先级别越高。例如，圆括号、加法和乘法运算符优先级的数值分别为1、4、3，因此表达式2+3*5相当于2+(3*5)，其结果为17。对于同一优先级的运算符，一定要按其结合方向进行运算，例如，运算符 / 和 * 的优先级都是3，结合方向为自左至右，因此表达式12/2*3相当于(12/2)*3，而不是12/(2*3)。

(2) 算术表达式中的变量必须有确定的值。

(3) 数学中的方括号“[ ]”和花括号“{ }”在C语言表达式中不能使用。C语言只允许使用圆括号，且可以用多层形式。

(4) 有时求得的算术表达式的值和预期的值不一样，例如，printf("%d",32767+1)的运行结果是-32768(因32767与1的和超过基本型取值范围)、printf("%ld",32767+1)的运行结果是 -1671168(按基本型计算得出错误结果后，再以长整型形式输出)，两个函数都得不到预期的值32768。为了保证数据的运算结果不超过数据范围，运算之前先正确估计结果的取值范围，并选择合适的数据类型。

【例1.14】 将代数式$\frac{\pi r^2}{a+b}$改写成C语言算术表达式。

【解】 C语言算术表达式为3.14159 * (r * r) / (a + b)。

说明：

(1) C语言不提供乘方运算符，因此只能用“*”计算乘方的值。

(2) 在C语言中，不能出现π，因为它既不是变量，也不是常量，因此改写时根据所需精度用3.14159或3.14等代替。

(3) (a+b)中的圆括号不能省略。

1.5.2 赋值运算符和表达式

1. 赋值运算符

C语言中提供的赋值运算符有=、+=、-=、*=、/=、%=等，其中后5个运算符是复合的赋值运算符(将在1.6.3节中介绍)。赋值运算符的优先级数值为14，结合方向是自右至左。

2. 赋值表达式

用赋值运算符把一个变量和一个C语言表达式连接起来的表达式称为赋值表达式。赋值表达式的一般形式是：

变量 = 表达式

例如，i=3*2是赋值表达式，其含义是将3和2的乘积6赋给变量i。

说明：

(1) 赋值表达式的处理过程是：先计算赋值运算符右边表达式的值，然后把该值赋给左边的变量。

(2) 赋值运算符的左边必须是变量(代表存储单元)，右边可以是C语言的任何合法表达式。假设i中的值为3，则i=i*2是合法的赋值表达式，其处理过程是先计算表达式

i＊2 的值(3＊2 的值为 6),然后把 6 赋给 i。注意,赋值运算符左边变量代表一个存储单元,而右边出现的变量应理解为该变量中的值,其值必须是事先被赋予的。

(3) 赋值表达式的值是赋值运算符左边的变量所得到的值,例如,赋值表达式 a＝3＊2 的值为 6;当 b 的值为 5 时,赋值表达式 b＝b＋2 的值为 7;当 a 的值为 1,b 的值为 2 时,赋值表达式 a＝b 的值是 2,但同样当 a 的值为 1,b 的值为 2 时,b＝a 的值却为 1;由于表达式 y＝8 的值为 8,因此赋值表达式 x＝(y＝8)＋1 的值为 9(即 8＋1)。

(4) 由于赋值运算符的结合方向是自右至左,因此 x＝y＝5 等价于 x＝(y＝5)。

(5) 在合法的赋值表达式中,如果赋值运算符的两边数据类型不一致,则系统先将右边表达式值的类型自动转换成左边变量的类型,然后再进行赋值。赋值时的转换规则参见 1.6.3 节。在不同数据类型之间进行赋值处理时,容易产生意想不到的错误。例如,将 double 型数据赋给 float 型变量时,由于数值范围不同,容易产生数据溢出现象,因此尽量避免使用这种赋值形式。

数据的类型也可以利用强制类型转换的方法,其一般形式为:

(类型名)(表达式)

【例 1.15】 编写一个强制类型转换的程序。

【解】 程序如下:

```
#include <stdio.h>
main( )
{    int i = 0,j = 0,k = 0;
     float x = 5.8,y = 3.7,f = 8.56;

     i = (int)(x + y);              /* 将 x + y 的结果 9.5(即 5.8 + 3.7)转换成 int 型 */
     j = (int)x + y;                /* 将从 x 中取出的值 5.8 转换成 int 型后,与 y 相加 */
     k = (int)f % 3;
     printf("i = %d,j = %d,k = %d,x = %f\n",i,j,k,x); /* x 中的值还是 5.8 */
}
```

运行结果:

```
i = 9,j = 8,k = 2,x = 5.800000
```

程序说明:

(1) 强制类型转换运算符(int)类型名外的一对圆括号不可少,例如,将 i＝(int)(x＋y)写成 i ＝ int (x＋y)是错误的。

(2) (int)(x＋y)和(int)x＋y 的含义不同,因此不要随意去掉 (x＋y) 中的括号。

(3) (int)x 的作用是将从 x 中取出的值 5.8 转换成整型 5,但没把 5 存入变量 x 中(即 x 中的值没变),也没有将 x 的类型转换成整型。

(4) %为求余运算符,要求两个运算量都是整型,因此 f%3 是不合法的表达式,但(int)f%3 是合法的,因为(int)f 的值是整型值 8。

1.5.3 逗号运算符和表达式

1. 逗号运算符

C 语言中提供的逗号运算符是“,”。逗号运算符的优先级数值为 15，是所有运算符中优先级最低的运算符，逗号运算符的结合方向是自左至右。

2. 逗号表达式

用逗号运算符把 C 语言表达式连接起来的表达式称为逗号表达式。

【例 1.16】 编写一个使用逗号表达式的程序。

【解】 程序如下：

```
#include <stdio.h>
main( )
{   int a=0,b=0,x=0,y=0;

    a=(x=8,x%5);                    /* 将逗号表达式(x=8,x%5)的值赋给 a */
    b=x=8,x%5;                      /* 先将赋值表达式 x=8 的值赋给 b,再求 x%5 的值 */
    printf("%d,%d,%d\n",a,b,(y=2,y*3));   /* 输出 a、b 和表达式(y=2,y*3)的值 */
}
```

运行结果：

```
3,8,6
```

程序说明：

(1) 逗号表达式 (x=8,x%5)的求解过程是先将 8 赋给 x，再求 8%5 的值(值为 3)。3 是此逗号表达式的值。

(2) 程序中 a=(x=8,x%5)和 b=x=8,x%5 的作用不同。a=(x=8,x%5)的求解过程是：先求表达式 (x=8,x%5)的值，后给 a 赋值，这是一个赋值表达式。b=x=8,x%5 的求解过程是：先将表达式 x=8 的值 8 赋给 b，再求 8%5 的值(赋值运算符的优先级比逗号运算符高)，即先求解第一个表达式，再求解第 2 个表达式，这是一个逗号表达式。因此在程序中不要随意添加或舍去圆括号。

(3) 程序中的有些逗号作为分隔符使用，例如，在最后一条输出语句中 a,b,(y=2,y*3)的前两个逗号是分隔符，后一个才是逗号运算符。

逗号表达式的一般形式为：

表达式 1,表达式 2,表达式 3,…,表达式 n

其求解过程是：从表达式 1 到表达式 n 按顺序求值。表达式 n 的值是逗号表达式的值。

1.5.4 自加、自减运算符

在 C 语言程序中，经常使用自加(++)、自减(--)运算符，它们的优先级数值为 2，