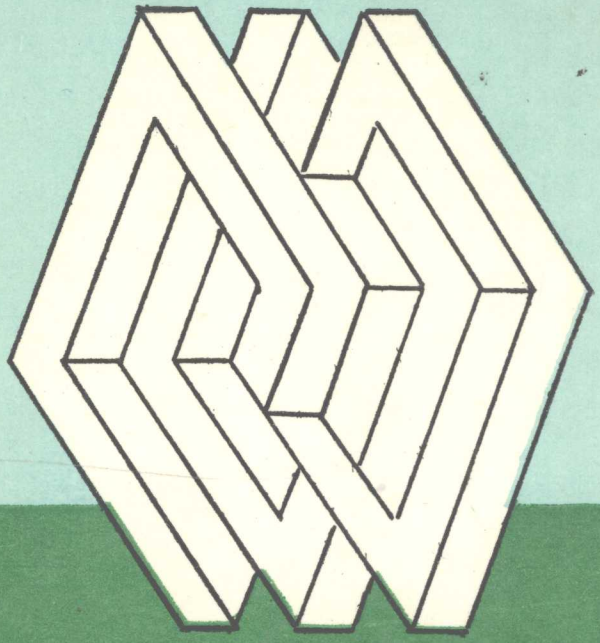
南京大学出版社



C和C++程序设计基础

杜建成　潘金贵　编著
顾铁成　审校

C和C++程序设计基础

杜建成　潘金贵　编著

顾铁成　审校

南京大学出版社

1995·南京

(苏)新登字第 011 号

内 容 简 介

C++语言是在C语言的基础上扩充而成的，它对C语言功能的最大扩充在于支持面向对象的程序设计(OOP)。

本书简要地介绍C语言和OOP的概貌，从基础知识入手，循序渐进，把C语言和C++语言的共同部分集中在一起介绍，而仅与C++语言有关的功能则作专门介绍。

全书共分十三章一个附录，叙述通俗易懂，示例十分丰富，非常便于自学(给出的每一个例子都可实际运行)。本书适于作为教材和教学参考书使用。

欲引进C和C++系统盘和本书例子盘的读者可与编者联系。

C和C++程序设计基础
杜建成　潘金贵　编著
顾铁成　审校
*
南京大学出版社出版
(南京大学校内，邮编 210008)
江苏省新华书店发行　　南京通达彩印厂印刷

开本 787×1092　1/16　印张 10.75 字数 263 千
1993 年 1 月第 1 版　　1995 年 1 月第 2 次印刷
印数 5001－13000
ISBN 7-305-01859-7/TP・58
定价：7.40 元
责任编辑　顾其兵

前　言

C 语言作为一种计算机程序设计语言，其显著的优点之一是可移植性好，因此可用于从个人计算机到大型计算机的各种计算机系统。

C 语言起初是为了开发 UNIX 系统而设计的一种高级语言。由于系统程序往往要求有相当高的运行速度，充分小的程序规模，因而多数情形下是用与硬件的依赖关系很强的汇编语言编制的。汇编语言一般一行对应于一条机器指令，优秀的程序员可以用它编制出相当漂亮的程序。但是，用汇编语言编制程序很费工夫，程序开发所需时间较长，这是汇编语言的一大缺点。此外，几乎没有什么统一的普遍适用的编程规则可以遵守，而特别的限制却非常之多，容易出错，要想得心应手，需要记忆很多东西，因而只有少数高级程序员能够熟练掌握，这是汇编语言的又一大缺点。程序的规模越大，汇编语言的这些缺点就越显得致命。要想在短时间内对一个用汇编语言编制的大型程序进行修改和扩充而不出错几乎是不可能的。C 语言作为一种为编制系统程序而设计的高级语言，为克服汇编语言的这些缺点提供了强有力的工具。

另一方面，即使是编制应用程序，所要处理的数据结构也越来越复杂，像本书中将要介绍的那些结构已屡见不鲜。处理如此复杂的数据是汇编语言所难以胜任的。而且汇编语言与硬件的依赖关系很强，一般的技术人员使用起来颇感困难。而 C 语言是一种高级语言，易于记忆，处理复杂数据结构的能力又强，因而也不失为编制应用程序的一种强有力的工具。所以说 C 语言是一种应用范围最为广泛的高级语言，目前越来越多的程序员正在转向用 C 语言编程。虽说 C 语言比汇编语言要简单得多，但与 BASIC 和 FORTRAN 等高级语言相比却未必简单，这是由于 C 语言中引入了许多其他高级语言所不具有的概念。

C++语言是对 C 语言进行扩充而成的一种程序设计语言，它几乎包括了 C 语言的全部功能。因此，要学好 C++语言，首先要学好 C 语言。当然，由于 C++语言又引入了一些连同 C 语言在内的其他高级语言所不具有的概念，要想全部理解并运用自如诚非易事。但是，可以预期，C++语言今后将成为最流行的一种计算机语言。

C++语言对 C 语言功能上的最大扩充在于支持面向对象的程序设计(OOP：Object-oriented Programming)。与迄今为止的结构化程序设计所不同的是，OOP 是从一种旨在提高软件的可扩充性和重用性的全新概念中产生出来的。以往的高级语言都是以结构化程序为目标而设计的，因而对数据和处理分别加以定义，仅在处理的时候要遵守一定的约束。这样，数据和处理的结合比较弱，不可能用与数据相关的结构来描述程序，因此在扩充和重用的时候要花不少工夫，并且经常出错。而在 C++语言中，数据被抽象成所谓的类(Class)，凡与该数据相关的处理都可以包装在类中。此外，这样的类还可以分层、继承等等。

一个程序员只要利用 C 语言原有的功能和少许扩充功能就可以编制出相当漂亮的程序来，若能灵活应用 C++语言的全部功能，特别是 OOP 当然再好不过了。从这一观点出发，本书将通俗地介绍 C 语言和 C++语言的基本内容，读者通过本书既可以学习 C 语言，也可以学习 C++语言。如果只想学习 C 语言，只要略去那些介绍扩充功能的部分即可。但

是，也许C++语言比C语言更能引起读者的兴趣。我们希望读者能体会到这一点，亦即，即使不掌握C++语言的高级使用方法，哪怕只利用一些简单的扩充功能，也将使编制出来的程序比只用C语言编制出来的程序要简洁得多。

基于上述考虑，本书是按照这样的结构写成的：

- 把C语言和C++语言的共同部分集中在一起进行介绍；
- 仅与C++语言有关的功能另作介绍；
- 简要地介绍C语言的概貌，从基础知识入手，循序渐进；
- 关于OOP，只介绍一些入门性的基础知识；
- 各部分都给出许多可以实际运行的完整的程序例子（所有例子程序都在Borland C++下运行通过）。

C语言以及在此基础上扩充而成的C++语言，都是高级语言，与汇编语言相比，有其易于记忆的优点，但它们却并不像其他高级语言那么易于掌握。一种有效的学习方法是通过大量简单的程序例子来学习编程思路和技巧，边学边用，不断充实自己的编程能力。

编著者

一九九二年五月于南京

目　录

第一章　C语言的特点和C程序

在详细介绍C语言和C++语言之前，本章首先对这两种语言各自的特点作一概述。接着，举一个简单的程序例子，说明程序的书写方法，供读者参考。此外，还要说明一下程序的书写格式和注释的写法及其作用等等。

§1.1　C语言和C++语言的特点

C++语言基本上包括了C语言的功能，因此，这里先简单介绍一下C语言和C++语言的共同特点，至于C++语言本身的特点将在后面介绍。

主要使用英文小写字母

所使用的保留字全部由英文小写字母构成，变量等标识符通常也由英文小写字母构成。因此，习惯上都用英文小写字母编写程序。但是，也可以用英文大写字母构成变量等标识符。此时，大写字母和小写字母是当作不同符号来处理的。在字符串内可以使用汉字等特殊符号(但是，只有在汉字能够受到正确处理的情形下方能使用汉字)。

表示简洁

保留字拼写较短，运算符种类较多，分隔符和运算符用的都是记号而不是通常的文字形式，因而表示简洁，使得程序输入时的输入符号数较少。

程序是函数的集合

程序的执行部分(执行语句)写在函数的内部，因此，程序成了各种函数的集合。一个程序至少由一个函数组成。与BASIC语言和FORTRAN语言不同，这里不存在子例程(subroutine)的概念。

函数是语句的集合

在函数内部可以写函数的原型说明语句，变量的类型定义语句，描述处理过程的执行语句等。执行语句以表达式为基础，凡可以指定表达式的位置均可指定对表达式的执行进行控制(进行条件判断和重复执行等)的控制结构。与BASIC语言和FORTRAN语言不同，这里不存在命令和子例程这样的概念。

函数均可递归调用

任何函数均可直接或间接地调用其自身。函数的这种调用方法就叫递归。利用这种性质，可以方便地用简洁的程序描述数学上由递归公式表示的式子，并可以用较少的程序实现一些较高级的处理。

变量和函数均须先定义(或者说明)再引用

变量意味着数据存放的场所，在引用变量之前，必须用定义语句进行定义，确保数据有存放的场所，或者用说明语句预先通告该变量数据存放的场所另有定义，然后才能引用。变量的定义语句和说明语句可以放在函数之中或者之外，依该语句出现的位置不同而有不同的作用域。对于函数，也可以指定其作用域。变量和函数一经定义，同时也就进行了说明。

表达式必定有值

一个常数、一个变量、一个函数都可称为表达式。而由一个或一个以上的表达式进行各种运算的式子也称为表达式。表达式后面若跟着分号(;)的话,便成了一个执行语句,执行一个表达式就是求出该表达式的值。表达式执行后所得到的值叫作评价值。

有高级的控制结构

条件分支和循环等控制功能丰富,可以编制出易于理解且便于修改和扩充的结构化程序。

指针(地址)可作为数据使用

指示数据存放场所的数据称作地址。其值为地址的变量称作指针。地址、指针这些概念在汇编语言中并不少见,由于C语言可以处理与汇编语言级别相当的数据,因而可以实现与汇编语言同等的功能。用在函数间传递地址的方法可以编出一些通用的函数,同时可以高速地进行某些复杂的处理,而用通常的高级语言是不能进行这些处理的。不过,地址和指针的概念难以理解,使用起来易于出错,且这些错误在多数情形下很难查出。

可以处理由基本数据组合而成的数据

作为基本数据而提供的有字符型和整型等几种,可以定义并使用由这些基本数据自由组合而成的集合数据。集合数据中有数组和结构,特别是结构对于复杂的数据处理尤为重要,在C++语言中它被扩充后起着重要的作用。

运算符种类丰富

汇编语言中可以利用的运算大多数作为C语言的运算符来定义。除了在通常的高级语言中也可使用的算术运算符、逻辑运算符、关系运算符等以外,还定义了按位逻辑运算符、移位运算符、赋值运算符、逗点运算符等。指明函数用的括号"("和")"以及数组中所用的括号"["和"]"等等也被定义成运算符。

没有字符串处理功能

字符串和字符有着明确的区别,字符串是作为字符的序列而处理的。有对作为基本数据的字符进行运算的运算符,却不存在对作为集合数据的字符串(一般说来对数组)进行运算的运算符,像BASIC语言那样用运算符进行字符串的赋值、比较等运算是做不到的。所有这些运算均利用函数进行。但在C++语言中可以自由定义并使用对字符串进行处理的运算符。

没有输入输出功能

其他高级语言必备的输入输出功能,C语言中却没有。所有的输入输出均由函数进行。由于那些使得编译程序复杂许多的文件处理等功能均由函数实现,编译程序变得非常紧凑。又由于编译程序中不含有那些受硬件影响较大的输入输出功能,程序在不同机种间的可移植性很强。这也是C语言得以广泛普及的原因之一。

具有预处理功能

利用宏功能和文件包含(嵌入)功能,可以编制出不受机种差异的影响、可移植性好的程序。此外,通过编制一些可为其他程序共同使用的包含文件,可以提高程序的开发效率。还可进行条件编译,即告知编译程序一个特定部分是否应成为编译对象。

一个程序可以分割成多个部分进行编制

可以把一个程序分割成多个模块进行编制,对它们分别进行编译,最后再把它们连接成

一个可执行的程序文件。分割后的文件中可以仅含有说明语句和定义语句，也可以仅含有函数。利用这种分割功能和变量等的有效范围等特性，便可以由多人共同对一个程序进行开发，这样有利于开发大型程序。不过，虽然一个文件中所含的函数个数没有限制，但不管哪一个函数都必须在一个文件中结束(把一个函数分散在多个文件里是不容许的)。

可以方便地构成命令

在启动用户编制的程序时，可以把命令行的参数传递给执行程序。被传递的参数在程序内进行解释，并据此进行适当的处理，因此可以用用户自编的程序对操作系统(OS)的功能进行扩充，这些追加的程序实际上成了与OS所提供的系统命令用法相同的可执行命令，即外部命令。

从以上介绍不难看出，仅仅利用C语言的功能已能显示出迄今为止的语言所不具有的优良特色，用扩充后的C++语言将能编制出更为漂亮的程序来。

C++语言基本上包含了C语言的功能。因此，在下面的介绍中，凡说到"C语言的情形"的时候，意为对C语言和C++语言均成立。至于那些只对其中一种语言成立的情形，均特别加以声明。

§1.2 一个简单程序

C语言程序中执行部分写在函数之内。因此，C语言的程序可以称之为函数的集合。有两种函数，一种是用户依据一定的规则自行编制的函数，另一种则是附属于编译程序或解释程序的通用标准函数。不用说，用户也可以在自己的应用范围内编制一些通用性较强的函数，在不同的程序里反复使用。

程序启动时第一个被执行的函数通常是用户自编的。这个在启动时第一个被执行的函数的名字是事先约定好的，必须是main。因此，一个程序中必定有一个且只有一个名为main的函数。

main函数最简单的书写形式如下：

```
main(){
        各种语句(变量的说明语句和执行语句等)
}
```

由于main是一个函数，其名字main之后必须紧跟着一对括号"("和")"，以表明它是一个函数，此外，为了标明函数的范围，要用一对括号"{"和"}"把函数体括起来。函数体由变量等的定义语句或说明语句以及各种执行语句组成。当然，变量的定义和说明可以放在函数之外进行，将在以后介绍。

【例1-1】 本例从键盘输入整数，在显示器上显示结果。

```
#include  <stdio.h>

main()  {
    int  a;

    printf("请输入整数值:");
```

```
    scanf("%d",&a);
    printf("%d 的平方为%d\n",a,a*a);
}
```

下面是一次运行的例子：

```
请输入整数值:4      (4 作为数据输入)
4 的平方为 16       (显示运算结果)
```

在这次运行的例子中，整数值 4 作为数据从键盘输入(4 之后必须键入一个 return 键)。作为结果显示出来的是输入值 4 及其平方值 16。

程序的第一行以 # 开头的

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

要在编译之前进行处理，在此情形下是由编译程序将系统配备的名为 stdio.h 的文件插入相应的位置。这一文件中写着与标准函数 printf 和 scanf 有关的一些规定，因此在程序的开头(在写变量的定义、说明和函数之前)必须写上这一行。较早的 C 语言中不写这一行也能正常工作，但有时会发生编译错误，因而还是写上这一行比较安全(在 C++ 语言中则务请写上这一行)。stdio.h 等文件名在 MS-DOS 的情形下大小写均可，但我们建议最好采用小写。像这样由 #include 插入的文件称为包含文件或称嵌入文件或称头文件(文件名中的.h 就是取自 header 的第一个字母)。此外，在编译之前对以 # 开头的行进行处理的程序叫作预处理程序。

整型变量 a 是由

```
int  a;
```

加以定义的。作为整型数据所能处理的数值大小是因机种和编译程序而异的，在 16 位处理的情形下为 −32768～32767。在程序内部所使用的变量必须事先加以定义。

标准函数 printf 是一个用于显示的函数，

```
printf("请输入整数值:");
```

显示的是字符串常数“请输入整数值:”。用双引号(〃)括起来的字符序列称作字符串，它被当作一个不能中途切分的逻辑整体进行处理。字符串常数内的字符可以是除了 return 等少数几个字符以外的几乎所有字符(包括汉字)。字符串中不能含有双引号。

标准函数 scanf 是一个用于从键盘接收输入的函数，在

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

这一行中，第一个参数(这里为"%d")指定的是说明格式的字符串，第二个参数(这里为 &a)以后指定的是数据输入的场所。往整型变量中输入整数值的情形下用 %d 这样的格式。&a 意指变量 a 的存放场所。其他格式以及变量前面的运算符 & 下面还将详述。

运算结果是用标准函数 printf 进行显示的，在

```
printf("%d 的平方为%d\n",a,a*a);
```

这一行中是用第一个参数中的第一个格式 %d 来显示第二个参数中变量 a 的值，用第二个格式 %d 来显示第三个参数中 a*a 的结果。printf 中的格式 %d 意指将整型数据以十进制整数的形式显示出来。非格式字符原封不动地进行输出，但\n 这样一些字符有着特殊的含义。\n 是一个控制字符，用于回车换行。这种具有特殊含义的字符下面还将详述。

在这个程序例子中，函数 main 含有一个定义语句和由三个标准函数构成的执行语句。

像这样，通过在一个函数中调用（执行）其他函数，可以简洁地描述一个复杂的处理过程。在使用函数的时候，如果有必要的话，应把函数内部所用到的数据作为其参数予以指定。参数有多少个，各对应于什么类型的数据，是因函数而异的。即使是同一函数，参数的个数也未必一样。

在这一程序例子中，main 的前面和 printf 的前面各有一个空行，虽说并不是非有不可，但为醒目起见，一般都在定义语句、说明语句和执行语句之间加一个空行。不过，在C++语言中，定义语句和说明语句是可以随用随写的，因此这样的空行就没有多大的意义了。

§1.3 自由格式和缩进格式

程序书写的格式是完全自由的，从什么地方开始写都无关紧要。这时，tab（严格地讲应为水平 tab）和 return 键视同于 space（空白）。以下把 tab，return 和 space 统称为白空字符。因此，在变量名、函数名和常数等单词与单词之间，凡可以书写空白的地方，原则上都可以自由地进行换行。由于随意换行和加空，可能使写出来的程序很难读，但是，像例 1-1 这样，在每一行的开头加适当的空白，可以使得程序更易于理解。这种在每一行的开头加适当空白的方法就叫作缩进。此外，为提高程序的可读性，在功能模块之间也可插一些空行。在字符串数据内部直接插进 return 是不行的，加进 tab 和 space 则被视为不同的字符串数据。

【例 1-2】 用自由格式书写的程序例子。

```
#include  <stdio.h>          /*这一行为一行*/
main(){int  a;printf("请输入整数值:");
scanf ("%d",
      &a);
printf("%d 的平方为%d\n",
      a,
      a*a);
      }
```

这一程序例子的执行内容与例 1-1 相同，读起来却很不舒服，应避免书写这样的程序。我们希望读者像例 1-1 那样用缩进格式书写可读性强的程序。

当字符串常数的长度太长需要分行书写的时候，可在换行之前加一个“\”符号（反斜杠），然后从下一行的开头接着书写。这样，编译的时候会忽略“\”符号和紧跟其后的 return，接着下一行进行处理。最近的 C 语言还把夹着白空字符的字符串常数当作一个连续的字符串常数进行处理。

【例 1-3】 字符串常数分行书写的例子。

```
#include  <stdio.h>
main(){
      int  a;
      printf("请输入\
整数值:");/*从该行开头写起*/
      scanf("%d",&a);
```

```
    printf("%d 的平"
    "方为%d\n",a,a*a);/*上接前一字符串*/
}
```

这一程序例子的执行内容与例 1-1 相同，字符串常数中的“\”和 return 被忽略，把“\”之前的字符和从下一行开头写起的字符连接起来进行处理。夹着白空字符的字符串常数作为一个连续的字符串常数进行处理。

§1.4 注 释

由“/*”和“*/”所界定的任意字符串被当作注释处理，编译时被忽略。在一个程序中，凡可插入白空字符(space，水平 tab，return)等的地方均可加适当的注释。注释跨行书写也行，但注释的嵌套(即在注释中加注释)一般是不行的(在 Borland C++中，可以指定注释可否嵌套)。在 C++语言中一行内结束的注释是用“//”打头的，该行结束的地方也就是注释结束的地方。虽然以“/*”和“*/”界定的注释是不能嵌套的，但是在这样的注释中容许指定以“//”打头的注释。在以“//”打头的注释中也可以指定以“/*”和“*/”界定的注释。

注释不是写给计算机看的，而是写给人看的，凡与程序有关的各种信息均可写在注释中。利用注释可以记载注意事项，说明处理内容等等。凡可以插入白空字符的地方均可插入适当的注释。但是保持程序的可读性仍很重要。例如，像

```
int  /*注释不当一例*/a;
```

这样的注释，就使得程序读起来很不舒服。

【例 1-4】 注释的例子。

```
/*说明注释用法的程序例子
    这种类型的注释
    可以跨行书写
*/
#include <stdio.h>
main() {                                    //执行始于此
    int a;                                  //变量定义
    printf("请输入整数值:");                //提示数据输入
    scanf("%d",&a);                         //数据输入
    printf("%d 的平方为%d\n",a,a*a);        //显示结果
}
```

这一程序例子的执行内容与例 1-1 相同。以“/*”和“*/”界定的字符串作为注释被编译程序忽略。在 C++语言中，“//”以下的字符串直到该行的结束均视为注释。如果用的是 C 语言，请把以“//”打头的注释改写成以“/*”和“*/”界定的注释。

第二章　数据类型、运算符、表达式和语句

在编制程序之前，首先要搞清一些必备的基本知识，如所能处理的数据的性质及其定义方法，运算符的功能及其使用方法，表达式和语句的书写方法等等，本章就要对此详加说明。

§2.1　数据类型

在C语言中事先定义好的数据叫作基本数据。基本数据从类型上分有字符型(char)，整型(int)，单精度浮点型(float)，双精度浮点型(doule)，扩充精度浮点型(long(double)。浮点型又称为实型。

整型数又有短整数(short int)，通常的整数(int)和长整数(long int)之别，所能处理的数据大小范围各异，在定义上，短整数所能表示的范围通常比长整数所能表示的范围小。具体各用多少位并没有作为语言规格确定下来，因而可随编译程序和机型而异。由于通常的整数其大小应最便于处理，因此在16位计算机中采用16位，在32位计算机中采用32位是再自然不过的了。

字符型数据虽然通常是用表示字符的字符编码(下面将要说明)赋值的，但是把它看成一种可以处理较小整数值的整型数据也未尝不可，把字符型数据视为字符编码还是视为整数，由编程者自便。

字符型数据和整型数据还有无符号字符(unsigned char)或无符号整数(unsigned int)和有符号字符(signed char)或有符号整数(signed int)之别，前者仅能处理非负值，后者也可以处理负值。当不指明 signed 或 unsigned 的时候，视同于 signed(亦即 signed 为隐含值)。

在用16位(2字节)表示一个整数的时候，如果是正负数均能处理的整数(int 型)，其取值范围从-2^{15}(=−32768)到$2^{15}-1$(=32767)。如果是仅能处理非负值的整数(unsigned int 型)，其取值范围从0到$2^{16}-1$(=65535)。字符型数据通常用8位，在 signed 情形下，其取值范围从-2^{7}(=−128)到$2^{7}-1$(=127)，在 unsigned 情形下，其取值范围从0到$2^{8}-1$(=255)。短整数的规定或从字符型，或从整型。长整数多用32位。另一方面，对于实型数据，单精度的情形下多用16位，双精度的情形下多用32位。各种基本数据所能表示的最大值等是在嵌入文件 limit.h 中加以规定的，因而可以用编辑程序查阅得到。

通常可以把多个数据集中起来，作为一个整体进行处理。这种数据的集合叫作集合数据。集合数据可以当作一个数据单位加以利用。C语言中只有集合数据，而C++语言中还有编程者自定义的新类型的数据，并可以把该数据所固有的有关处理一并加以定义。关于这一点后面还要详述。

【例2-1】　基本数据举例。

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

```
    long int c; unsigned long int d;
    float e; double f;
    a=-1;b=-1;
    c=123456789; d=-1;
    e=3.14; f=e*e;
    printf("%d %u\n",a,b);
    printf("%ld %lu\n",c,d);
    printf("%f %f\n",e,f);
}
```

把作为有符号数据的－1赋给无符号整数 b 和无符号长整数 d 并不会出错，这时－1成了无符号整数所能表示的最大值。关于变量的定义方法和标准函数 printf 中的格式下面将加以详述。单精度实数 e 和双精度实数 f 二者均用%f 的格式显示。关于这一点下面还将详述，但我们在这里就可以指出，通常单精度实数是被转换成双精度实数加以处理的。

§2.2 常　　数

整型数值常数的表示方法有三种。一种是作为十进制数按通常的数值数据来表示。例如

0　　100　　－123

都是正确的表示法。在较早的 C 语言中，像＋123这样带有正号的数有时是被当作出错处理的。另一种表示方法是八进制数表示法。例如，像

0123　　0777

这样，在数字前冠以符号0。此外，第三种表示方法是十六进制数表示法，它是在数字前冠以 0X 来表示的。例如

0x123　　0x7ff

都是正确的表示法。字符 x 和十六进制数中的数字 a～f 大小写均可。八进制数和十六进制数都是作为无符号整数进行处理的。

【例2-2】 整数常数举例。

```
#include <stdio.h>
main() {
   printf("%d%d\n", 123,-456);
   printf("0123=%d 0×ff=%d\n",0123,0xff);
   printf("0xffff=%d 0xffff+1=%d\n", 0xffff,0xffff+1);
}
```

例中第三个 printf 仅为验证之用，没有什么实际意义。第一个和第二个 printf 是常用的（一般地，从第二个参数起像例中这样指定常数的情形并不多见，多指定变量和表达式等）。在整数作16位处理的情形下，最后一个 printf 显示出来的是－1和0，因为0xffff 的每一位均为1，用有符号整数来解释它恰好为－1。因此0xffff＋1恰好为0。

在把大小为普通整型数据所能处理的整数常数定义为长整型数据的时候，要像

123L　　0L

这样，在数值的后面紧跟着一个小写字母 l 或大写字母 L。如果其值超过了普通整型数据所能处理的范围，例如

123456

等，其字面本身就决定了要当作长整型数据加以处理。此外，整数常数还有如下一些后缀：

123U　　无符号整数(unsigned int 型)

123uL　　无符号长整数(unsigned long int 型)

【例2-3】 长整型常数举例。

```
#include <stdio.h>
main() {
  long a;
  a=123;
  printf("%ld  %ld\n",a,456L);
  printf("%ld\n",123);/*写成123L方为正确*/
}
```

第二个 printf 中的123被视为普通的整数值，在%ld 这样的格式下不能正确显示。要在常数之后紧跟着一个小写字母 l 或大写字母 L，写成123l 或123L，才能正确显示。

实型常数(浮点型常数)有两种表示方法。一种是通常直接写出小数部分的方法，例如

12.345　　999.9

第二种表示方法在科学计算中最为常用，这是一种指定指数部分的方法。例如

12.345e2　　意为$12.345\times10^{2}=1234.5$

3E−5　　意为$3\times10^{-5}=0.00003$

字母 E(大小写均可)以后表示的是指数部分。这样的浮点常数现在都被当作双精度浮点常数处理，这样做是为了保持与较早的 C 语言的兼容性。此外，浮点常数可以有下述这些后缀：

1.23F　　单精度浮点数(float 型)

1.23L　　扩充精度浮点数(long double 型)

【例2-4】 实型常数举例。

```
#include <stdio.h>
main() {
  float a;
  a=123;
  printf("%f  %f\n",a,123e-2);
  printf("%f\n",123);/*写成123.0方为正确*/
}
```

第二个 printf 中的123被视为普通的整数值，在%f 这样的格式下不能正确显示，要写成123.0使之成为实型才能正确显示。

字符常数的值被定义为字符编码。用一对单引号界定的单个字符便是一个字符常数。例如

'A'　　'a'　　'0'　　'%'