第一章 绪 论

本章学习重点

本章介绍 C 语言的发展历史以及 C 语言的特点，然后通过简单例子展示 C 语言的风貌，最后告诉你如何在计算机上运行 C 程序。

第一节 C 语言的产生及特点

C 语言是一种得到广泛重视并普遍应用的计算机程序设计语言，也是国际上公认的最重要的几种通用程序设计语言之一。它适合作系统描述语言，既可用来写系统软件，也可用来写应用软件。

C 语言是美国贝尔实验室的 Dennis. M. Ritchie 于 1972 年设计实现的。C 语言直接来源于 B 语言，但它的根源可以追溯到 ALGOL60。ALGOL60 结构严谨，其设计者非常注重语法、分程序结构，因此对于后来许多重要的程序设计语言，如 PASCAL，PL/I，SIMULA67 产生过重要的影响。但它是面向过程的语言，与计算机硬件相距甚远，不适合编写系统软件。1963 年英国剑桥大学在 ALGOL60 的基础上描出更接近硬件的 CPL 语言，但 CPL 太复杂，难于实现。1967 年，剑桥大学的 Matin Rinchards 对 CPL 语言作了简化，推出了 BCPL 语言。1970 年贝尔实验室的 Ken Thompson 以 BCPL 为基础，设计了更简单也更接近硬件的 B 语言(取 BCPL 的第一个字母)。B 语言是一种解释性语言，功能上也不够强，为了很好地适应系统程序设计的要求，Ritchie 把 B 发展成称之为 C 的语言(取 BCPL 的第二个字母)。C 语言既保持了 BCPL 和 B 的优点(如精练，接近硬件)，又克服了它们的缺点(如过于简单，数据无类型等)。1973 年 K. Thompson 和 D. M. Ritchie 用 C 改写了 UNIX 代码，并在 PDP—11 计算机上加以实现，即 UNIX 版本 5，这一版本奠定了 UNIX 系统的基础，使 UNIX 逐渐成为最重要的操作系统之一。

C 语言是 D. M. Ritchie1972 年设计实现的

图 1-1 展示了 C 语言的由来。

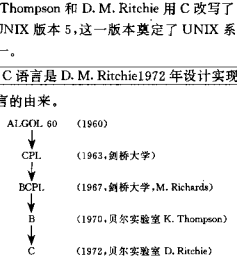

图 1-1 C 语言的由来

C 设计语言的目的是为描述和实现 UNIX 操作系统提供一种工具语言。但 C 并没有被缚束在任何特定的硬件或操作系统上，它具有良好的可移植性。1977 年出现了不依赖于具体机器的 C 语言编译文本，使向各种机器移植 C 变得更加简单，这也推动了 UNIX 系统的广泛实现。随着 UNIX 的日益普及，又反过头来带动了 C 语言的迅速推广，使它先后被移植到各种大、中、小、微型计算机上。

1978 年，贝尔实验室的 Brian. W. Kernighan 和 Dennis. M. Ritchie(合称 K&R)合著了《The C Programming Language》一书，并在附录中提供了 C 语言参考手册，这本书成为以后广泛使用的 C 语言的基础，被人们称作非官方的 C 语言标准。1983 年美国国家标准化协会(ANSI)开始制定新的标准，这就是 ANSI C 标准。1990 年，C 语言成为国际标准化组织(ISO)通过的标准语言。

C 语言是作为描述系统的语言而设计，但随着其日益广泛的应用，特别是 80 年代以后各种微机 C 语言的普及，它已经成为众多程序员最喜爱的语言，它的使用覆盖了几乎计算机的所有领域，包括操作系统、编译程序、数据库管理程序、CAD、过程控制、图形图象处理等等。

C 语言如此成功是有其自身特点的：

1. C 语言是比较"低级的"语言。

有人把 C 语言称为"高级语言中的低级语言"，也有人称它是"中级语言"。这样说不是说它功能差或难于使用，而是指它具有许多通常只有象汇编语言才具备的功能，如位操作、直接访问物理地址等等，这使 C 语言在进行系统程序设计时显得非常有效，而过去系统软件通常只能用汇编语言编写。事实上，C 语言的许多应用场合是汇编语言的传统领地，现在用 C 来代替汇编，使程序员得以减轻负担、提高效率，而写出的程序具有更好的可移植性。

C 语言具有更多的接近硬件操作的功能，而不提供直接处理复合对象，如作为整体看待的字符串、数组等的操作，这些较高级的功能必须通过显式调用函数来完成。这看起来是个缺陷，但这使语言的规模较小，更容易说明，学习起来也快。比如说，C 语言只有 32 个关键字，而一些微机上的 BASIC，关键字多达 100 个以上。

2. C 语言是结构化的语言。

C 语言在结构上类似于 ALGOL60、PASCAL 等结构化语言。这大概与它源于 ALGOL60 以及 60 年代末 70 年代初结构化程序设计方法的兴起有关。C 语言的主要结构成分是函数——C 的子程序。函数允许一个程序中的各任务分别定义和编码，使程序模块化，在函数的外部只需了解函数的功能，而将实现的细节隐藏起来，设计得好的函数能够正确地工作而对程序的其他部分不产生副作用。C 语言还提供了多种结构化的控制语句，如用于循环的 for、while、do-while 语句，用于判定的 if-else、switch 语句等，以满足结构化程序设计的要求。

3. C 语言是程序员的语言。

看到这个提法，你可能会奇怪：难道还有不是程序员的程序设计语言吗？确实有许多程序设计语言不是为专业程序员设计的，比如说，FORTRAN 是为工程师，COBOL 是为商业人员，PASCAL 是为学生，而 BASIC 是为非程序员设计的。C 是为专业程序员设计的语言。Ritchie 是专业程序员，而 C 最初是为了他自己写 UNIX 操作系统而设计的。C 语言实现了程序员的期望，很少限制，很少强求，程序设计自由度大，方便的控制结构，独立的函数，紧凑的关键字集合和较高的执行效率。用 C 编写程序可获得高效的机器代码，其效率通常只比

汇编语言生成的机器代码低 10～20%，而同时 C 又具有 PASCAL 那样的结构，这就难怪有大量的程序员喜欢它。

C 语言的语法限制不太严，例如，对数组下标越界不做检查，整型、字符型数据可以通用，不专设逻辑型数据而以整型来代替等。较少的限制给程序员带来较大自由，这就要求程序员在编程时应确实明白自己在做什么，而不要把检查错误的工作仅寄托于编译程序。当然这会带来一些麻烦，但作为程序员应该考虑好再开始编码。其实，有时候让计算机惩罚一下粗枝大叶的程序员也不一定是什么坏事。

C 语言得到广泛使用的主要原因可能就是因为程序员喜欢它，但它也不是专业人员的专利，非专业人员经过实践也会熟练地掌握它。现在有许多不同专业的非计算机专业人员正在使用或正在转向 C 语言。C 语言是我们大家共同的财富。

第二节　用 C 语言编写程序

学习一种程序设计语言唯一的有效途径就是用它编写程序。下面几个简单的程序将带你进入 C 的世界，开始我们的学习。

第一个程序总是输出一个字符串，这里，选择著名的"hello, world"程序，以表示对 K&R 的敬意。

例 1.1

```
main()
{
    printf("hello,world\n");
}
```

这个程序的运行结果是输出字符串

```
hello,world
```

上面的程序称 C 语言源程序，简称 C 程序。为了让它运行，必须先找一个编辑程序输入它，然后编译，装入，最后执行。

现在让我们来看看程序本身。main 是一个函数名，表示"主函数"。一个 C 程序，不管简单或复杂，总是由一个或多个函数组成，由这些函数实现要做的操作。函数名可以按照你的喜好去取，但 main 是一个特殊的名字。C 程序总是从 main 函数开始执行，这意味着，一个程序必须在某个地方有一个 main 函数。

花括号{}括起构造函数的语句，称函数体。函数体中只有一条函数调用语句

```
printf("hello,world\n");
```

其功能是将双引号""中的内容输出。printf 是 C 语言中的库函数，它来自计算机厂家提供的输入输出库，你可以直接去调用它。双引号中的内容是调用 printf 时提供的参数，表示为一个字符串。字符串中的\n 是 C 语言中的换行字符，在输出时，它将终端的光标移动到下一行的开始。如果不使用\n，输出时就会发现，光标停在输出字符串的后面。printf 不提供自动换行的功能，每当在程序中需要执行换行操作时，都必须写出\n。例如，我们把例 1.1 改写一下，

```
main()
{
    printf("hello,");
    printf("world");
    printf("\n");
}
```

执行结果也是一样的。如果要将程序改成

```
main()
{
    printf("hello,\n");
    printf("world\n");
}
```

你会发现输出变为：

```
hello,
world
```

在这里我们可以看到，C 语言倾向于把怎样处理的权利交给程序员。

printf 语句的最后跟着一个分号";"，它表明一个语句的结束。

例 1.2

```
/* 求两个整数之和 */
main()
{
    int a, b, sum;
    a = 12;
    b = 34;
    sum = a + b;
    printf("sum is %d\n", sum);
}
```

本程序的功能是求两整数之和。/*…*/部分是注释，是为了便于阅读程序而安排的，对程序的编译和执行没有影响。注释可以加在程序的任何位置。注释可以是任意一串字符，但不能再含有/*…*/，也就是说，注释是不能嵌套的(也没有必要)。这里的注释是用汉字给出的，你的计算机可能没装汉字，可用英文或汉语拼音来写注释。本书中采用汉字注释完全是为了阅读的方便。

程序中的

```
int a, b, sum;
```

是变量定义部分，这里我们用 int 定义变量 a、b 和 sum 为整型。C 语言规定使用任何变量之前都要先定义。

```
a = 12;
```

```
    b = 34 ;
    sum = a + b ;
```
是三个赋值语句，“=”是赋值运算符，它的作用是把其右部表达式的值赋给左部的变量，这里是把值 12 赋给变量 a，把值 34 赋给变量 b，把 a+b 的结果赋给变量 sum。

```
    printf("sum is %d \n",sum);
```
中的"sum is"和"\n"我们已经熟悉，%d 是一个输出转换说明，意思是：输出时用一个整型值来代替它，这里，这个整型值是变量 sum 的值。于是本程序的输出结果为：

```
    sum is 46
```

例 1.3

```
main()/* 求两个数中较大者 */
{
    int a, b, c ;
    printf("a,b=") ;
    scanf("%d,%d", &a, &b) ;
    c = max(a, b) ;
printf("max=%d\n", c) ;
}
int max(a, b)   /* 返回 x、y 中较大者 */
int x, y ;
{
    int   z;
    if ( x > y )
            z = x;
    else
            z = y ;
    return(z) ;
}
```

本程序定义了两个函数：main 和被调函数 max。在 main 中，语句

```
    printf("a,b=") ;
```
输出

```
    a,b=
```
然后光标停在=之后。下一句

```
    scanf("%d,%d",&a, &b) ;
```
用于接收两个整型数据到变量 a 和 b 中。scanf 是 c 输入输出库中提供的输入函数，它要求按照转换字符（这里是%d，表示整型）将相应类型的数据输入到指定变量的存储单元中去，&a 表示取变量 a 的地址。

语句

```
    c = max(a, b) ;
```

是将函数 max 的值赋给变量 c，其中 a 和 b 是参数(称为实在参数)，执行时将变量 a 和 b 的值传给被调函数 max。max 的作用是将 x、y 中较大的数送 z，并通过语句

```
return(z);
```

将其作为函数值返回给主调函数。函数头

```
int max(x, y)
```

```
int x, y;
```

表示 max 带有两个形式参数 x 和 y，在 max 被调用时，这两个参数的值由主调函数中对应的实在参数(本程序中是 a 和 b 的值)传过来，函数体中求出 x、y 中较大的数值送 z，并返回主调函数，语句

```
if( x > y)
    z = x;
else
    z = y;
```

是 C 中的一种判定控制语句结构，它根据 x 是否大于 y 来选择执行

```
z = x
```

或

```
z = y
```

本程序的执行情况如下：

```
a,b=4,6↵
max=6
```

例 1.4

```
#include "stdio.h"
main()
{
    int c;
    while((c = getchar()) != EOF)
            putchar(c);
}
```

这个程序的功能是将输入的内容复制到输出，第一行

```
#include "stdio.h"
```

是 C 语言中的文件包含，stdio.h 是一个头文件，通过文件包含命令 #include，程序把 stdio.h 包含在自己所在的文件中。stdio.h 是关于标准输入输出的头文件，其中包括使用标准输入输出库函数的许多信息。在本程序中，由于使用 getchar 和 putchar 函数(实际上是宏，将在第九章中介绍)，因此需要用命令包含，将存有必要信息的 stdio.h 包含进来。符号 EOF 也在 stdio.h 中定义，实际中，它定义为

```
#define EOF -1
```

意思是在程序中遇到 EOF 就用 -1 代替，EOF 是文件结束标志，对于任意一个文件，系统会自动在其尾部加一个 EOF 标志。当文件是从键盘上输入时，在微机 DOS 操作系统下，打入

^ z(ctrl+z)后，系统自动为输入流加上一个 EOF(UNIX 系统下，可打入 ^ d)。函数 getchar 用于读一个字符，putchar 用于输出一个字符，语句 while 表示当括号内的表达式值为真时，反复地执行后面跟着的语句，因此

```
while((c=getcharc())! =EOF)
    putchar(c);
```

的意思是：当读入的字符不是 EOF 时，就输出它，直到读入 EOF，所以我们可以这样运行这个程序

```
abc1234  xyz789 ^ z(输入)
abc1234  xyz789  (输出)
```

在本章的例子中，我们让每个语句单独占据一行的位置，并且书写时作了适当的缩进，这些都不是 C 语言语法所要求的，而是为了阅读方便，如例 1.4 也可以写成

```
#include "stdio.h"
main(){int c;while((c=getcharc())! =EOF)putchar(c);}
```

其结果也是一样的。C 语言不是靠回车，而是靠分号来标识一个语句的结束，回车的功能与一个空格是相同的。虽然可以把程序写成上面那个样子，而且可以正确地执行，但实际中几乎没有人那么写，即使是很短的程序。计算机界有句名言：程序是给人读的。看到这个命题，你也许会奇怪，程序不是为了在计算机上运行的吗？确实如此，但不仅仅如此，你写一个程序以后可能要修改，别人也可能会阅读或修改你的程序，在没有完全读懂的情况下做修改，将会带来许多麻烦，即使不改，一个表达清楚的程序也会给人带来愉快。再者，连你自己都不能清楚地表达思路的话，还能指望计算机作什么呢？

为了清晰而正确地写程序，首先要使程序具有良好的结构，如使用将实现细节隐蔽得很好的函数，少用外部变量，不采用非结构化的控制结构等。同时，也要注意书写风格，如每个语句独立占一行，适当的缩进，尽量选择有实际意义的标识符，适当的加入注释等等，这些在我们今后的讨论中还会反复提到。总之，从一开始写程序就要培养好的风格。

第三节　C 程序上机运行

学习编程，上机是非常重要的，在计算机上编程并运行，从结果中了解程序的运行过程，进而了解程序的结构，比只看书要有效得多。另外，你可以对已有的程序做各种修改，进而了解语言更多的知识，也可以向别人学到很多东西。上机可以提高你编程的兴趣，也可以提高自信心，当看到自己编写的程序在计算机上正确地运行时，你会感到自己的创造力。

本节中，我们介绍 turbo　C 上机的简单步骤，如果你使用其他的 C，或希望对 turbo　C 有更多的了解，请参考相应的用户手册。

用 turbo C 上机的步骤：

先将 turbo C 软件安装到机器的硬盘上。

1. 调入 turbo　C

只要打入 tc，然后回车，屏幕顶部将显示出 turbo C 的命令菜单(见图 1-2)

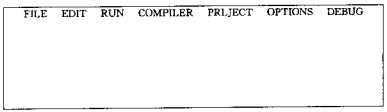

图 1-2 命令菜单

用键盘上的"←"和"→"键可以移动屏幕上的光标，如果光标不在顶部的命令菜单行中，可以打 F10 键将光标移到顶部。将光标移到"FILE"处，打回车键，FILE 下面出现一个子菜单，如图 1-3，用"↓"键将光标移到 Load(或 New)处打回车键，表示要输入源程序，屏幕上又出现一个小窗口，如图 1.4 所示，要求你输入文件名，此时可打入

file1.c↵

如果原来没有这个名字的文件，将建立一个新文件，如果已有此文件，则将此文件调入并在屏幕上显示，然后自动转为编辑(EDIT)状态。

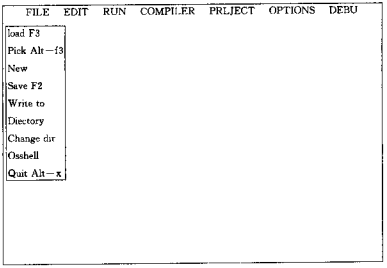

图 1-3 子菜单

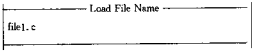

图 1-4 文件名输入框

你也可以直接进入编辑状态，这时在 DOS 命令行中打入

tc file1.c

系统就自动进入编辑屏幕。也可不打 c，这时系统取缺省值.c。

2. 编辑源文件。

你可以根据需要输入或修改源程序。

3. 编译源程序

只需要简单地打 F9 键，就可以进行源程序的编译和连接，并在屏幕上显示有无错误。如有错误，按任一键后，屏幕显示源程序，光标停在出错的位置上。屏幕的下半部分将显示出

有错误的行和错误原因。根据提供的信息可以修改错误，这时按下“F6”键后就可修改错误，修改完后再按“F6”键，接着打“↓”键，光标就停到下一个出错的行。所有错误修改完后，再按“F9”进行编译，如此反复，直到没有编译错误为止。

4. 执行程序

你只要打入 ^ F9(按下 ctrl 键的同时按 F9 键)即可执行程序。如果程序中需要输入数据，就在此时将所需数据输入，接着程序执行，输出结果。

如果发现结果不对，可再按步骤 2、3、4 修改程序、编译、执行程序，直到得到正确的结果。

5. 保存文件

编辑完源程序或准备退出 turbo C 时，也可打“F2”键将源程序存在硬盘上。

6. 退出 turbo C

同时按下 Alt 和 X 键，便退出 turbo C，回到操作系统命令状态。此时可以用系统命令来显示源程序或运行程序。如

type file1.c↓(显示源程序清单)

file1↓(执行程序 file1.exe)

如果想再输入或修改源程序，可以再从步骤 1 开始。

以上的步骤是对一个程序只放在一个源文件中时采用的，如果一个程序放在多个文件中，就要使用 PROJECT 项生成所谓 prj 文件。考虑到初学时，程序通常都较小，不必放在多个文件中分别编译，这里就不多介绍了。

最后讲一下本书附带磁盘的使用方法。本书中全部例子的源程序附在一张磁盘上，文件名是字母 E 开头，后跟章、例号，如例 1.1 为

E1－1.c

你可以直接调入 turbo C 编译并执行这些例子，有些例子有多个程序，这时文件名中再含有一个标识版本的数字放在最后，如例 1.1 有两个版本的话，程序名分别是

E1－1－1.c

和

E1－1－2.c.

书中有些例子只给出了一个被调函数，而没有 main，为了便于上机操作，盘中都给了用于测试这些例子的主调函数，上机时也可以直接编译运行。

第二章　数据类型、运算符与表达式

本章学习重点

本章介绍C语言的基本数据类型：整型、浮点型、字符型，相应类型的变量和常量以及类型之间的转换，并介绍算术、赋值和逗号表达式的概念和使用。通过本章的学习，应能掌握C语言数据和运算的基本概念，为以后各章的学习打下基础。

第一节　C语言的数据类型

许多人总爱形象地把计算机程序与做菜用的菜谱做个类比。确实如此，菜谱中规定了厨师烹饪的步骤，就象计算机程序中一条条指令规定了计算机应怎样运行。菜谱中的指令总是把各种食品原料当作自己操作的对象，而计算机中要被处理的对象就是数据。

图2.1给出了C语言中所能处理的各种数据类型。

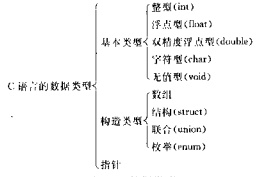

图2-1　数据类型

在本章中我们介绍基本数据类型，而其他数据类型留在后面章节中讨论。

第二节　标识符

标识符是一个字符序列，在C语言中用来标识常量、变量、函数、数据类型的名字。C语言规定：标识符只能由字母、数字及下划线"_"(不是减号)组成，并且数字不能做为标识符的第一个字符，下面是一些合法的标识符：

sum
a1
i
J5x3
Num_of_lines
iobuf

不合法的标识符如：

5t　　以数字开头

U.S　　出现了字符

Good bye　中间留有空格

C 语言规定，字母的大小写是有区别的，比如，SUM、Sum 和 sum 代表三个不同的标识符，如果你曾有过 PASCAL 或 FORTRAN 等语言的编程经历，那么一定要注意这一点。

C 语言本身并没有要求标识符的长度，但在任何机器上，所能识别的标识符的长度总是有限的，有些系统可以识别长达 31 个字符的标识符（如 VAX－11 VMS C），而有些系统只能识别 8 个字符长度的标识符，这意味着：即使第 9 个字符不同，只要前 8 个字符一样，系统也认为是同一个标识符，如 category1 和 category2。因此，为了避免出错和增加可移植性，标识符最好前 8 个字符有所区别。

一个写得好的程序，特别是一个有实际意义的程序，标识符的选择应尽量反映出所代表对象的实际意思。如表示年可以用 year，表示长度可以用 length，表示和可以用 sum 等，这样的标识符本身就增加了可读性，使程序更加清晰。

C 语言中有一些标识符被称为关键字，它们是系统留做特殊用途的，不能作为一般标识符使用，如我们见过的整型变量说明 int 等。附录 B 中给出了 C 语言中全部关键字。

有些标识符虽不是关键字，但 C 语言总是以固定的形式用于专门的地方，因此，用户也不要把它们当做一般标识符使用，以免造成混乱。这些标识符常用的有：

define include ifdef ifndef endif elif

第三节　常量

常量是指在程序运行过程中其值不能被改变的量。

2.3.1　整型常量

整型常量简称整数，C 语言中有三种形式的整型常量。

1. 十进制整数

以我们通常习惯的十进制整数形式给出，如：

10，－345，0，＋5

2. 八进制整数

八进制整数是以 0 开头的八进制数字串，只能使用数字 0～7。八进制数没有符号，如果一个八进制前带有负号，实际上是对这个八进制数进行了单目减运算。下面是一些合法的八进制整数：

0123　　（十进 83）

0400　　（十进制 256）

0177777　　（十进制 65535）

3. 十六进制整数

十六进制整数是以 0x 或 0X 开头的十六进制字符串。十六进制字符有 0～9，a～f（或 A～F），其中 a～f（或 A～F）表示十进制值 10～15，下面是一些合法的十六进制整数

0x12　　（十进制 18）

0xb　　（十进制 11）

0xA8　　（十进制 168）

0xFFFF　　（十进制 65535）

整型数的取值范围通常是由机器的字长决定的。在一个字长 16 位的计算机上，带符号整数值的范围是十进制－32768～32767，无符号整数值的范围是十进制 0～65535。数值超过这个范围时，系统自动按长整数对待。长整数每个数值占 4 个字节(32 位)存储空间，长整数的表示形式是在数字的后面跟上字母 L，如

12L　　（十进制 12）

－100L　　（十进制－100）

1234567L　　（十进制 1234567）

012L　　（十进制 10）

0200000L　　（十进制 65536）

0x12L　　（十进制 18）

0x10000L　　（十进制 65536）

看起来 12L 与 12 好象没有什么差别，但在一个用 16 位二进制表示整数的计算机上，12 占据 2 个字节存储空间，而 12L 占据 4 个字节。

2.3.2 浮点型常量

浮点型常量也称实数。C 语言中实数有两种表示形式。

1. 十进制数形式

由数字，小数点和可能的正负号组成，如

0.345，.345，345.，345.0，0.0，－3.45

2. 指数形式

也称科学计数法，用 e 或 E 表示指数，其一般形式为

aEb

表示 $a\times10^b$，其中 b 必须是整数，下面给出一些指数形式实数的例子：

345e2　　（相当于 345×10^2）

－3.2e5　　（相当于 -3.2×10^5）

.5e－2　　（相当于 0.5×10^{-2}）

实数不管表现形式如何，总是占据 8 个字节的存储空间(即以双精度的形式出现)。

2.3.3 字符型常量

C 语言中的字符型常量是用单引号(向左撇)括起来的一个字符，如

'a','x','0','*','A',' '

C 语言的字符常量占据一个字节的存储空间，在那个存储单元中存放的实际上并不是字符本身，而是该字符在所在机器采用的字符集中的编码，也就是一个整数值。大多数机器系统采用 ASC Ⅱ 字符集(本书中也假定是 ASC Ⅱ 字符集，见附录 A)，在这种情况下，'a' 在内存中表示为对应的 ASC Ⅱ 码 97，'0' 表示为 48，形式如图 2-2 所示。

图 2-2　字符常数的表示

由于字符常量是个整数，因此它可以象整数一样参加数值运算。当然，它的主要用途是与其他字符作比较。

除了以上形式的字符常量外，C 语言还允许使用一种以特殊形式出现的字符常量，以表示某些非图形字符，这就是以"\"开头的转义字符序列。在上一章中，我们用"\n"表示换行，\n 实际上是一个字符，它的 ASC Ⅱ 码值为 10，因此可以表示为 '\n'。常见的以\开头的转义序列见表 2.1

表 2.1　转义字符序列表

字符形式	功能
\n	换行
\t	水平制表(下一个 tab 键的位置)
\v	垂直制表(竖向跳格)
\b	退格
\r	回车
\f	走纸
\0	空字符
\	反斜线
\'	单引号'
\ "	双引号"
\ddd	1～3 位 8 进制所代表的字符

\ddd 是用 1～3 位八进制数来表示相应的 ASC Ⅱ 码，如 '\n' 也可以表示为 '\012' 或 '\12'。

2.3.4　字符串常量

字符串常量是用一对双引号(")括起来的零个或多个字符的序列，如

"This is a character string."

"CHINA"

"0123456789"

"$10000.00"

"a"

" "（引号中有一个空格）

""（引号中什么也没有）

"\n"（引号中有一个转义字符）

字符串常量在内存中存储时，系统自动为每个字符串常量的尾部加一个字符\0，用以标识这字符串的结束，如字符串

"CHINA"

在内存中的形式是

图 2-3　字符常数的表示

(实际上，字母应当用对应的 ASC Ⅱ 码表示，但为了方便，以后表示字符时，直接用字符

本身表示)。

了解了这一点,你就可以区分字符串"a"和字符'a'有何不同了。它们在内存中的形式如图 2-4 所示

(a)'a'的存储表示　　(b)"a"的存储表示

图 2-4

字符串以\0 结束这种形式表明,C 中的字符串的长度是不受限制,其长度可以靠\0 来判断。

2.2.5 符号常量

常量也可用一个标识符来命名,这就是符号常量,如

例 2.1

```
#define PI 3.1415926
main()
{
    float radius,circum,area;
    printf("%d",&radius) ;
    circum = 2 * PI * radius ;
    area = PI * radius * radius ;
    printf("circumference is%f\n", circum);
    printf("area is%f\n", area) ;
}
```

运行结果

```
Input radius:3
circumference is 18.849556
area is 28.274334
```

这个程序是按输入的半径值,求圆周及圆的面积。第一行的

```
#define PI 3.1415926
```

定义了一个符号常量 PI,以后凡在程序中出现 PI,都表示 3.1415926。PI 是一个常量,在程序中只能引用,而不能被改变。用符号常量来代替常数本身,至少有两个好处:

其一,可以使程序更清晰易读,如我们定义了一个表示每页行数的符号常量

```
#define PAGESIZE 55
```

则语句

```
while (line <= PAGESIZE)
    line ++ ;
```

比较清楚地表现了当行(line)小于每页尺寸时应做的工作。如果用

```
while l(line<=55)
    linc ++ ;
```

就很难弄清楚 55 代表什么。

其二，程序更易修改。当我们把页的大小改变为 66 行时，只需要改写

#define PAGESIZE 66

而程序的其余部分不用改变。如果我们不用符号常量而用数值本身，就需要在程序中到处寻找数值 55，而且并不是所有 55 都表示是每页的尺寸，这就大大增加了修改的难度，也很容易出错。

通常的习惯是用大写字母表示符号常量，用小写字母表示变量等，以示区别。

最后提醒一点，定义符号常量的 define 行不要以分号结束，它不是一个语句。

第四节　变量及其说明

变量是指程序执行过程中，其值可以改变的量。

变量有一个名字，称为变量名，用标识符表示。每个变量都与一个数据类型相联系，类型决定了变量可以取值的范围和它可以参加的运算。C 语言规定：

所有变量在使用之前必须加以说明

变量的说明形式为

类型名　变量名，变量名，……变量名；

2.4.1　整型变量

整型变量用关键字 int 说明。

如

```
int i, j, k ;
```

说明了三个整型变量 i，j 和 k。

整型变量的取值范围由机器字长决定，如微机的整型变量占据二个字节，取值在 $-32768 \sim 32767(-2^{15} \sim 2^{15}-1)$ 之间，而整型变量占 4 个字节的机器（如 VAX－11），整型变量取值范围是 $-2147483648 \sim 2147483647(-2^{31} \sim 2^{31}-1)$。

除了基本 int 型外，还可以在 int 前加上修饰符来改变 int 型的意义，修饰符有：

signed（有符号）

unsigned（无符号）

long（长型）

short（短型）

由于整型的缺省形式是有符号的，所以 signed 可以不用（确实极少看到有人用），加上修饰符后，整型变量的形式有：

short int　　可简写为 short

long int　　可简写为 long

unsigned int　　可简写为 unsigned

unsigned short int　可简写为 unsigned short

unsigned long int　可简写为 unsigned long

也就是说，整型变量的说明加上修饰符后，int 可以省略。整型变量加上修饰符后，其取值范围有所变化。以 16 位机为例，表 2.2 给出了各种形式整型变量的取值范围。

表 2.2 整型量取值范围

类型	所占位数	取值范围
int	16	-32768~32767
short	16	--32768~32767
long	32	-2147483648~2147483647
unsigned	16	0~65535
unsigned short	16	0~65535
unsigned long	32	0~429497295

如果是 32 位机，则 int 和 unsigned 的取值与 long 和 unsigned long 相同。

例 2.2

```
main()
{
    int a, b, c, d ;
    a = -1 ;
    b = 4 ;
    c = 6 ;
    d = (a + b) * c ;
    printf("%d\n", d) ;
}
```

运行结果

18

2.4.2 浮点型变量

浮点型变量也称实型变量，有单精度和双精度之分。

单精度浮点型变量用关键字 float 说明，双精度浮点型变量用关键字 double 说明。如：

```
float f, g ;
double d ;
```

一般来讲，float 型变量占据 4 个字节(32 位)存储空间，绝对值取值范围约在 10^{-78}~10^{38}之间，取 6~7 位有效数字；double 型占据 8 个字节(64 位)存储空间，取值范围由具体机器决定，有 12~16 位有效数字，除 float 和 double 型外，ANSI C 还增加了 long double 类型，占 16 字节(128 位)，有效数字约 24 位，但多数 C 编译中没有设置。

前面的例 2.1 就是一个使用浮点型变量的例子。

2.4.3 字符型变量

字符型变量由关键字 char 说明，

如

```
char c1, c2 ;
```

一个字符型变量占据一个字节(8 位)存储空间，只能存放一个字符。不要以为它能够存放一个字符串，如：

```
c1 = 'a' ;
```
是正确的，而
```
c2 = "a" ;
```
则是错误的。

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

C 语言把字符型量当做一个较小的整型量，可以象整型量一样使用它。

例 2.3

```
main()
{
    char c1 = 'a' ;
    char c2 = 'b' ;
    char c3,c4;
    c3 = c1 - ('a' - 'A') ;
    c4 = c2 - ('a' - 'A') ;
    printf("%c%c\n", c3, c4) ;
}
```

运行结果为

```
A  B
```

程序中的 'a' － 'A' 是大小写字母之间的差值（别忘了字符对应相应的 ASCⅡ码），其值为 32。也可以把程序写成

```
c3 = c1 - 32 ;
c4 = c2 - 32 ;
```

效果是一样的，printf 中的%c 表示输出一个字符。

字符型数据也可以用整数形式输出，如将输出语句改为

```
printf("%d%d\n", c3, c4) ;
```

则运行结果为

```
65  66
```

2.4.4 变量赋初值

变量定义之后，我们可以用赋值的形式给它确定的值，也可以在定义变量的同时给它设置初值。如例 2.3 也可写成

```
main()
{
    char  c1 = 'a' ;
    char  c2 = 'b' ;
    char  c3,  c4;
    c3 = c1 - ('a' - 'A') ;
    c4 = c2 - ('a' - 'A') ;
```