第3章 数据存储类型与相关运算

程序运算的对象是数据，程序运行时数据需要存储、调用和运算。数据是描述和表达客观事物的符号形式，是信息的载体，计算机在识别和处理数据时，按数据类型占用系统资源。数据类型定义说明了数据运算性质、占据存储空间大小及内部存储模式等。C 语言可以使用多种数据类型，每个数据都属于某一种数据类型，每种数据类型又有常量和变量两种。不同类型的数据，其取值范围、表达形式及其在计算机存储器中的存放格式是不相同的。程序中用到的所有数据都必须通过定义指定其数据类型。本章主要内容如下：

- 数据存储方式与数据类型；
- 存储地址与占用空间；
- 数据常量与变量定义；
- 数据存储的正负数问题；
- 数据变量取值范围；
- 各种存储类型的混合运算；
- 运算符优先级与数据类型转换；
- 各类运算符与运算表达式。

3.1 数据存储方式

计算机中所有数据和指令都是以二进制形式存放与运行的，应用 C 语言编程必须了解数据在计算机中的存储与定义。

3.1.1 数据存储与数制转换

计算机中信息的数据表示是多样而复杂的，不仅有能够进行数学运算的数值型数据，还有表达自然语言的字符数据，以及图形图像数据、音视频数据等，最终都要转换成计算机可以识别的二进制形式。由于具备二态稳定的物理状态比较容易实现，如电路电压的高与低、电容的充电与放电、磁场磁畴的变换、开关接通与断开、晶体管导通与截止等，使用二进制数编码，技术上容易实现，因此，计算机中的数据最终以二进制数码表示与存储。

机器语言中的二进制数只有 0 和 1 两个数码，分别代表逻辑代数中的“假”和“真”两种状态。现实生活中人们习惯使用十进制，这样就存在着机器与人之间数据交流时

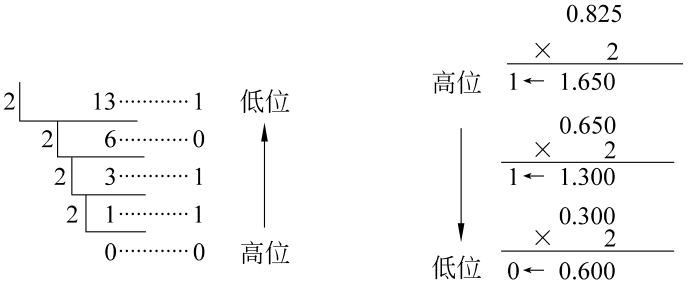
的进制转换问题。一般有二进制、十进制、八进制、十六进制几种数制之间的转换与应用，通常由计算机自动实现数制之间的转换。常用的几种数制之间的转换关系如表 3-1 所示。

表 3-1　常用数制间的转换关系

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	000	0	0	9	1001	11	9
1	001	1	1	10	1010	12	A
2	010	2	2	11	1011	13	B
3	011	3	3	12	1100	14	C
4	100	4	4	13	1101	15	D
5	101	5	5	14	1110	16	E
6	110	6	6	15	1111	17	F
7	111	7	7	16	10000	20	10
8	1000	10	8				

学习 C 语言程序设计需要熟悉并掌握各种数据类型的存储性质与使用。十进制整数转换成二进制数，整数部分用除 2 取余方法，小数部分用乘 2 取整方法。

例如，求$(13.825)_{10}$的二进制形式：



求得：$(13.825)_{10}=(1101.11)_2$。

其中精度问题由计算机根据数据类型定义自动取舍来控制。

任何数制的数值都可以展开成为一个多项式，其中每个数据项是所在位的位权与位系数的乘积，这个多项式求和结果是该数所对应的十进制数。

例如，求$(1101.11)_2$ 的十进制形式：

$$(1101.11)_2 = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$$
$$= 8+4+1+0.5+0.25=(13.75)_{10}$$

求得：$(1101.11)_2=(13.75)_{10}$。

二进制转换为八进制时，整数部分从低位到高位，小数部分则从小数点开始往右，每 3 位为一组，不足 3 位时补 0，每组 3 位二进制数对应 1 位八进制数。

例如，求$(100111101.11011)_2$ 的八进制形式：

将$(100111101.11011)_2$ 分组为100 111 101 . 110 110

对应八进制数　　4　7　5　.　6　6

求得：$(100111101.11011)_2=(475.66)_8$。

二进制数转换为十六进制数时，每 4 位为一组，不足 4 位用 0 补齐。

例如，求$(1010110.11101)_2$的十六进制形式：

二进制数分组	101	0110	.	1110	1000
对应十六进制数	5	6	.	E	8

求得：$(1010110.11101)_2=(56.E8)_{16}$。

3.1.2 数据存储类型与定义

程序设计一般包括数据描述和流程控制描述。数据描述主要指数据结构，在 C 语言中，数据结构以数据类型的存储形式体现，通过定义内存变量，确定其存储空间的大小及取值范围。

标准 C 语言并没有规定各种数据类型占有多少字节，只要求 int 类型（普通整型）长度应大于或等于 short 类型（短整型），并且小于或等于 long 类型（长整型）。同一数据类型在不同的编译系统中的存储形式各有差异，因此数据类型的最大值和最小值就会有所不同，可以使用字节长度测试函数 sizeof(类型名)，在实际使用的编译系统环境中测试数据类型的实际字节长度，了解数据类型的取值范围。

C 语言提供了丰富的数据类型，这些数据类型及定义关键字如图 3-1 所示。

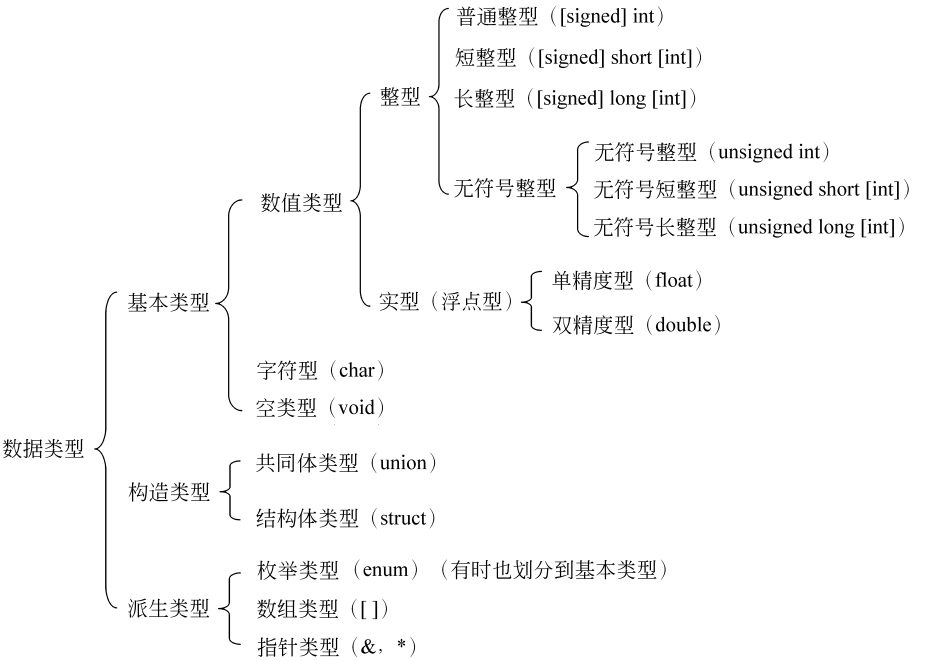


图 3-1 数据类型及定义关键字

由于 ANSI C 并没有规定每一种数据类型的长度、精度和取值范围，不同的 C 语言编译系统对于同一数据类型的存储长度可能有差别，应用时注意测试，以免影响正常使用。

例 3-1 编程在 Microsoft Visual C++ 6.0 和 Turbo C 2.0 集成环境下分别测试常用数据类型的实际字节长度。

程序源代码：

```
/*L3_1.c*/
#include "stdio.h"
main()
{
    printf("the char is%d\n", sizeof(char));
    printf("the int is%d\n", sizeof(int));
    printf("the short int is%d\n", sizeof(short));
    printf("the long int is%d\n", sizeof(long));
    printf("the float is%d\n", sizeof(float));
    printf("the double is%d\n", sizeof(double));
    printf("the long double is%d\n", sizeof(long double));
    printf("the void is%d\n", sizeof(void));
}
```

该程序在 Microsoft Visual C++ 6.0 集成环境下运行后，显示各种常用数据类型的实际字节长度，结果如图 3-2 所示。

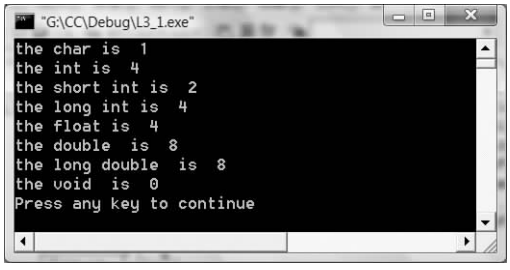


图 3-2 Visual C++ 6.0 数据类型的实际字节长度

由这个案例程序的运行结果可见，在 Microsoft Visual C++ 6.0 集成环境下，字符型长度为 1B、普通整型长度为 4B、短整型长度为 2B、长整型长度为 4B、实数浮点型长度为 4B、实数浮点双精度型长度为 8B、长实数浮点双精度型长度也为 8B、空类型长度为 0。

该程序在 Turbo C 2.0 集成环境下运行后，显示结果如图 3-3 所示。

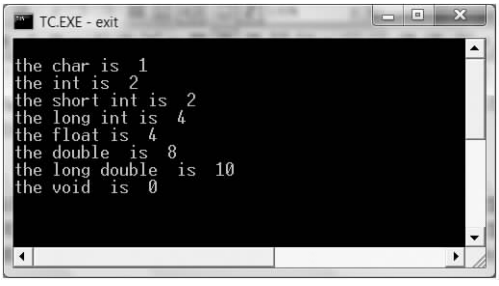


图 3-3 Turbo C 2.0 数据类型的实际字节长度

由这个案例程序的运行结果可见，在 Turbo C 2.0 集成环境下，字符型长度为 1B，普通整型长度为 2B，短整型长度为 2B，长整型长度为 4B，实数浮点型长度为 4B，实数浮点双精度型长度为 8B，长实数浮点双精度型长度为 10B，空类型长度为 0。

在常用的 C 语言编译系统中，整型数据在内存中占 2B 或 4B，即 16b 或 32b。

表 3-2 列出在 Microsoft Visual C++ 6.0 和 Turbo C 2.0 集成环境下各种基本数据类型的位数和取值范围。

表 3-2　数据类型的位数及取值范围

数据类型	Microsoft Visual C++ 6.0		Turbo C 2.0	
	长度/b	取值范围	长度/b	取值范围
[signed] int	32	−2 147 483 648～2 147 483 647	16	−32 768～32 767
[signed] short [int]	16	−32 768～32 767	16	−32 768～32 767
[signed] long [int]	32	−2 147 483 648～2 147 483 647	32	−2 147 483 648～2 147 483 647
unsigned [int]	32	0～4 294 967 295(即 $2^{32}-1$)	16	0～65 535
unsigned short [int]	16	0～65 535	16	0～65 535
unsigned long [int]	32	0～4 294 967 295(即 $2^{32}-1$)	32	0～4 294 967 295
float	32	3.4×10^{-38}～3.4×10^{38}	32	3.4×10^{-38}～3.4×10^{38}
double	64	1.7×10^{-308}～1.7×10^{308}	64	1.7×10^{-308}～1.7×10^{308}
[signed] char	8	−127～127	8	−127～127
unsigned char	8	0～255	8	0～255
void	0	无值	0	无值

这些数据类型还可以构成更为复杂的数据结构，例如利用指针和结构体类型可以构成链表、树和栈等数据结构。

在程序运行过程中其值保持不变的量称为常量。存放数据常量的是数据变量，简称变量，变量是在程序运行过程中其值能够被改变的量。常量和变量都表现为或属于某一数据类型。在 C 语言中，常量不需要类型说明，变量则需要类型定义与说明，需要先定义再使用。

3.1.3　存储地址与占用空间

计算机程序在运行时需要调入内存工作区，程序中定义的数据在编译后也占有各自的内存区域，数据所占有的存储单元个数是由其类型决定的。内存空间通常划分为一个个存储单元，每个存储单元最小为一个字节(B)，即 8 个二进制位(b)。每个存储单元均有一个唯一的编号，就是内存地址，计算机系统访问内存时是按地址操作的，如图 3-4 所示。

内存单元的地址与存储在单元里的数据是有区别的，单元地址如同房间号码，存储在单元里的数据则如同房间里住的人。不同类型的数据占有的内存大小不同，字符类型占一个存储单元，整形占两个连续存储单元，浮点类型则占 4 个连续存储单元。各种数据类型的第一个存储单元地址称为首地址，是计算机系统寻找数据存储单元时的起始地址。

例 3-2　编写一个程序，定义不同数据类型的变量并赋值，分别输出各种数据类型的

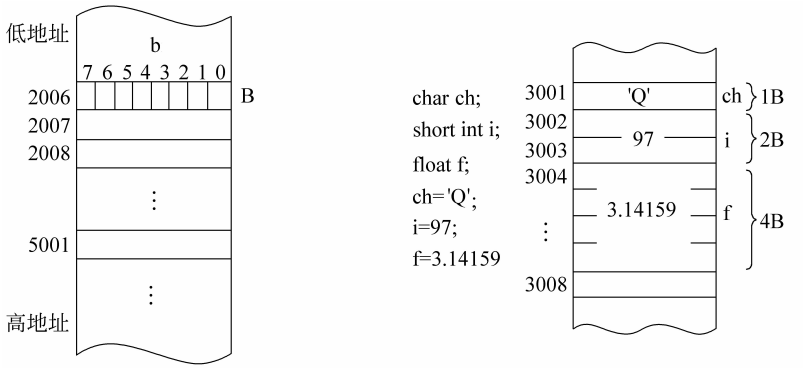


图 3-4　内存单元与地址

变量值和所占内存空间大小。

程序源代码：

```
/*L3_2.c*/
#include "stdio.h"
main()
{
    char c;                          /*创建字符型变量 c*/
    short int s_i;                   /*创建短整型变量 s_i*/
    int i;                           /*创建整型变量 i*/
    long int l_i;                    /*创建长整型变量 l_i*/
    float f;                         /*创建浮点型变量 f*/
    double d;                        /*创建双精度浮点型变量 d*/
    long double l_d;                 /*创建长双精度浮点型变量 l_d*/
    /*分别对变量赋值*/
    c='B';s_i=123;i=456;l_i=789;f=123.5;d=678.9;l_d=1.7e302;
    /*按数据类型分别输出各变量值,并输出各数据类型所占用内存字节数*/
    printf("the variable c is %c, save long as%d\n", c,sizeof(c));
    printf("the variable s_i is %d, save long as%d\n",s_i, sizeof(i));
    printf("the variable i is %d, save long as%d\n",i, sizeof(s_i));
    printf("the variable l_i is %ld, save long as%d\n",l_i, sizeof(l_i));
    printf("the variable f is %f, save long as%d\n",f, sizeof(f));
    printf("the variable d is %lf, save long as%d\n", d,sizeof(d));
    printf("the variable l_d is %e, save long as%d\n",l_d, sizeof(l_d));
}
```

编译运行后，显示输出各变量的值和变量所占内存空间的字节数，如图 3-5 所示。

计算机操作的最小存储单位是二进制位，简称位；计算机数据操作的最小存储单位是字节，8 个二进制位为一个字节；存储单元则是存放指令和数据的基本单位，每一类数据的存储空间可以由一个或若干个存储单元组成。

存储容量通常以字节为单位来表示，如 1KB 表示 1×2^{10}B，即 1×1024B；1MB 是 1×2^{20}B，即 1×1024KB；1GB 是 1×2^{30}B；1TB 是 1×2^{40}B；1PB 则是 1×2^{50}B。

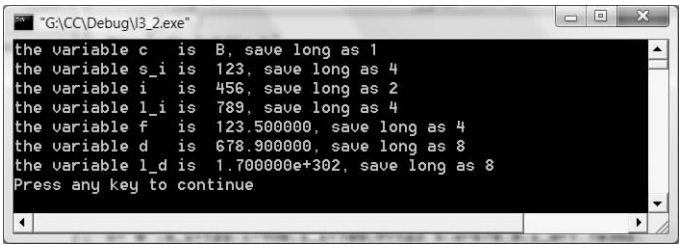


图 3-5　各种数据类型变量值及所占内存字节数

3.1.4　数据常量分类

C 语言的常量，分为数值常量和字符常量，数值常量又分为整型常量和实型常量。这些常量无须说明就可以使用，一般从字面形式就可以判别它们的类型。

1. 整型常量

整型常量可以用十进制、八进制和十六进制 3 种形式表示。

- 十进制整数：由数字 0～9 和正负号表示，如 0、－123、456 等。
- 八进制整数：由数字 0 开头，后随数字 0～7 表示，如 0123、075 等。八进制转换为十进制数符合进位记数制运算规则。例如，将 0123 转换为十进制数，即

$$1\times8^2+2\times8^1+3\times8^0=(83)_{10}$$

- 十六进制整数：由 0x 开头，后随 0～9、a～f(或 A～F)表示，如 0x1B3、0xdf 等。

例如－7、45、6789u、05706、0x67f9 和 6789L 等，其中－7、45、6789u 为整型十进制数，05706 为整型八进制数，0x67f9 为整型十六进制数，6789L 则为长整型十进制数。

整型常量后面加上小写字母 u 或大写字母 U，表示为无符号整型常量(unsigned int)，数据占用的存储单元的最高位不作为符号位，而用来表示数据。

长整型常量后面加上小写字母 l 或大写字母 L。

八进制数或十六进制数转换为十进制数，运算方法符合进位记数制运算规则。例如，将十六进制常数 0x123 转换为十进制数，即

$$1\times16^2+2\times16^1+3\times16^0=(291)_{10}$$

注意，在实际应用中，八进制和十六进制的表示没有负数，这是因为所有数据均以二进制数存放，八进制数和十六进制数是由二进制表示转换而来的，负数转换成二进制数时是以补码的形式表示的。例如，八进制数 07602 是正确的，八进制数－0357 则是错误的。

在程序设计中，各种类型的数据在使用过程中一定要匹配，比如变量的赋值操作、函数的参数传递、输入与输出类型控制等。

2. 实型常量

实型常量也称为浮点型常量、实数或浮点数。在计算机程序设计中，除了整数以外还有小数，如 1234.567。小数只能用实数方法表示和使用，实数只采用十进制数表示。

实型常量有十进制小数形式和十进制指数形式两种表示方法。

- 十进制小数形式：由数字 0～9 和小数点组成，实数必须有小数点，例如 0.16、.5、2.6、9.、2.34、－678.921 等均为十进制实数。
- 十进制指数形式：也称科学记数法，以指数形式表示浮点型数据，如 2.3E－5、6.542e6 等。在计算机系统中，指数形式用来表示非常大的数值或非常小的数值，如 2.34E－12 表示数值 2.34×10^{-12}。

指数形式由 3 部分组成，中间的字母 e 或 E 代表数字 10，e 或 E 之前必须有数字，e 或 E 之后的指数部分必须为整数。例如，7.1E5、－2.8E－2 为合法的实型常量，而 2.7E、59、e7、－257、46.－E4 均为非法的实型常量。

一个实数可以有多种指数表示形式，通常用标准指数形式表示，即在字母 e 或 E 之前的小数部分中，小数点左边保留一位非零数字，例如 1.3562e6、3.0254E5。实数在以科学记数法输出时是按标准的指数形式输出的。

3. 字符常量

字符常量包括键盘上的大小写字母（A～Z、a～z）、数字字符（0～9）和专用字符（!、@、#、$、&、*、|）等，每一个字符均按 ASCII 编码规则转换后存储。例如，根据附录 A 查得大写字母 Q 的 ASCII 码值为十进制 81，转换成二进制表示为 1010001；小写字母 q 的 ASCII 码值为十进制 113，转换成二进制表示为 1110001。小写字母 q 的存储方式为

0	1	1	1	0	0	0	1

即使用一个字节存储，最高位补 0。实际上，ASCII 码表只用低 7 位就可以表示各种常用字符及控制字符，最多可以表示 $2^7-1=127$ 个不同的 ASCII 码值。

字符常量在使用时还要注意一些细节问题，分别说明如下。

1）单个字符常量

单个字符常量是用一对单引号引起来的单个字符，如'a'、'$'、'3'、'9'等，每一个字符均对应一个 ASCII 码。注意，键盘上的数字字符的 ASCII 码并不等于数字本身，例如数字字符 8 的字符常量表示为'8'，其 ASCII 码为 00001000。

ASCII 字符的码值可以用十进制表示，例如，字符'A' 的十进制 ASCII 码值为 65，字符'a'的十进制 ASCII 码值为 97，字符'0'的十进制 ASCII 码值为 48，字符'\n'的十进制 ASCII 码值为 10。

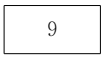
2）字符串常量

字符串常量是用一对双引号引起来的单个字符或字符串，如"a"、"123"、"test"等。字符串常量中的每一个字符均有自己的 ASCII 码值。双引号是字符串常量的标识，如常量"9"是字符串，它与字符常量'9'占用的存储空间不同。

字符串常量"9"占 2 个存储单元：

9	\0

字符常量'9'占1个存储单元：



可见，字符串常量和字符常量是不同的量。字符串常量"abc"在存储器内部存储的字符序列为

其中ASCII字符'\0'(NULL)为字符串结束标志。

3）控制字符常量

在ASCII字符集中，有一些字符不能用符号表示，也不能通过键盘输入，这些字符代表着特定的意义，需用反斜线'\'和特定字符组合表示，这类字符叫作转义字符。C语言中的常用转义字符如表3-3所示。

表3-3　常用转义字符

字符形式	ASCII码值	功　能
\0	0x00	NULL
\a	0x07	响铃
\b	0x08	退格
\t	0x09	水平制表(tab)
\f	0x0c	走纸换页
\n	0x0n	回车换行
\v	0x0b	垂直制表
\r	0x0d	回车(不换行)
\\	0x5c	反斜线
\'	0x27	单引号
\"	0x22	双引号
\?	0x3f	问号
\nnn	0nnn	用3位八进制数表示ASCII字符
\xhh	0xhh	用2位十六进制数表示ASCII字符

实际上，C语言ASCII字符集中的任何一个字符均可用转义字符来表示。表中\nnn是用八进制数表示ASCII字符，\xhh是用十六进制数表示ASCII字符。nnn是3位八进制数，hh是2位十六进制数。例如，'\102'表示字母B，'\134'表示反斜线等。以字符常量A为例，可以有以下几种表示形式：

- 'A'为字符A的字符常量表示形式；
- '\101'为字符A的八进制数表示形式；
- 65为字符A的十进制数表示形式；
- '\x41'为字符A的十六进制数表示形式。

上述4种表示形式均可以表示字母A。

4）符号常量

为了提高C程序设计实现过程的可维护性，可以使用一个标识符来代表一个常量，称为符号常量。符号常量的定义格式为

```
#define 标识符 字符串常量
```

其中，#define是一条预处理命令，称为宏定义命令，该命令的功能是把标识符定义为其后的字符串常量，程序编译时，将程序中所有符号常量均代换为其后的字符串常量。C语言程序设计习惯上将符号常量名用大写标识，普通变量名用小写标识。

例 3-3 输入半径(radius)，计算圆的周长(perimeter)、面积(area)和球的体积(volume)，数值输出格式要求预留宽度为10个字符，小数点后保留4位，四舍五入。

程序源代码：

```
/*L3_3.c*/
#define PI 3.1415926                                    /*定义符号常量*/
main ( )
{
    float perimeter, area, radius, volume;
    printf("input radius=:");                           /*提示输入半径*/
    scanf ("%f", & radius);                             /*输入半径*/
    perimeter=2.0*PI*radius;
    area=PI*radius*radius;
    volume=4/3.0*PI*radius*radius*radius;
    printf("perimeter=%10.4f\n area=%10.4f\n volume=%10.4f\n",perimeter,
    area, volume);
}
```

程序编译、链接后运行3次，输入3个不同的半径值，即输入1.、2.6和3.3，得到3组不同的结果，如图3-6所示。

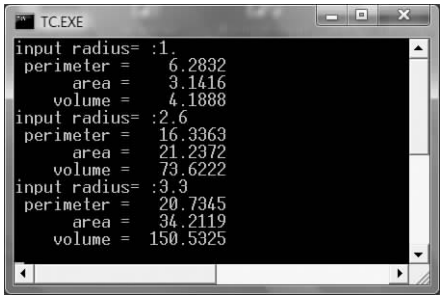


图3-6 输入不同半径值的输出结果

程序中定义的PI代表常量3.1415926，在编译源程序时，遇到PI就用常量3.1415926代替，PI可以和常量一样进行运算。程序中控制符%10.4f表示输出浮点类型变量值，预留输出宽度为10个字符，小数点后保留4位，四舍五入。

C语言规定，每个符号常量的定义式占据一行，而且符号常量不能被再次赋值。

使用符号常量的好处主要有以下两点：

(1) 提高了程序的可读性。本例程序中，PI代表圆周率。因此，定义符号常量时，应该尽量使用见名知意的常量名。

(2) 提高了程序的易修改、易维护性。本例程序中，圆周率用符号常量表示，则程序需要改变运算精度时，不需要逐一查找和修改程序中用到圆周率3.1415926的语句，只需

修改符号常量 PI 定义中的数值，程序中的所有 PI 都会自动代换为新的值。

3.1.5 程序变量定义

程序变量是在程序运行过程中可以被重新赋值，改变其存储内容的量。在 C 程序设计中，任何变量使用前都必须首先定义变量标识符和数据类型，一般定义格式为

```
数据类型  变量标识符
```

变量定义包括两部分：第一部分是数据类型，规定了变量的赋值运算类型和存储空间大小；第二部分是变量标识符，也称变量名，是程序运行时数据存储及操作运算的对象。每个变量都必须有一个合法的变量名，变量命名需遵循 C 语言标识符的命名规则。变量在正确定义后必须赋值才能调用，即使用变量运算之前变量必须有值，在程序运行过程中，变量值存储在内存中，程序通过变量名来引用变量值。

变量定义格式中的变量标识符可以是一个变量，也可以是多个变量，若定义两个以上的变量，各变量名之间用逗号“,”作为分隔符。在类型说明符与变量名之间至少要有一个空格作为分隔符。变量定义语句最后必须以“;”号结束。所有变量定义必须放在执行语句之前，函数变量的定义放在函数体的开始部分。

已经定义的变量在赋值后才能使用。例如，定义一个整型变量 a，对变量赋值，如 a=256，然后对变量 a 与常量 10 求和，再赋值给变量 a。这 3 条命令执行完成后，变量 a 的值为 266，如图 3-7 所示。

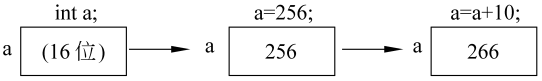


图 3-7　整型变量 a 的定义与使用

变量名标识符实际上是一个符号地址，在编译时可获得系统分配的一个内存地址，作为存放变量值的存储单元。从变量 a 中取值时，程序通过变量名 a 找到相应的内存地址，并从该内存地址对应的存储单元中读取数据 256，与整型常量 10 相加后重新赋给变量 a。

由于大写字母和小写字母按 ASCII 编码规则被认为是两个不同的字符，因此在 C 语言程序中，标识符 PRICE 和 price 代表的是两个不同的量，习惯上变量名用小写字母表示，符号常量用大写字母表示。

C 语言允许定义变量的同时对变量赋初值，称为变量初始化。变量初始化的一般形式为

```
数据类型  变量 1=值 1, 变量 2=值 2, …;
```

变量命名时一般应注意以下几点：

(1) 标准 C 语言变量名通常以小写字母或下画线开始。

(2) 变量名中的字符可以是字母 a～z、数字 0～9、字符@或下画线。

(3) 变量名的有效长度通常为 32 个字符。

(4) C 语言约定的保留关键字不能用作变量名。

例如,以下是正确的变量定义和初始化:

```
int a,_sum=123,b=0x123;
float f_x=9.85,f_y=.9,f_z=190;
char ch1='R',ch2=97;
```

而以下是错误的变量定义:

```
int Dr.liu;
char 3x;
float $f;
```

以上 3 条语句中的变量名不正确。

另外,变量不允许连续赋值,如 int a=b=c=5;是不正确的写法。有些语法规则因系统而异,但考虑到程序设计的兼容性,应尽量使用标准写法。

3.2 数据存储方式与应用

数据类型一经指定,程序运行时系统就会根据定义自动分配内存空间,数据存储方式及取值范围也就限定了。C 语言的数据存储方式与实际应用关系紧密。

3.2.1 数据存储的正负数问题

计算机系统中的正负数问题主要是指数值型数据的负数符号如何表达、负数如何运算等,最终归结到数据存储方式上解决,就是无论正数还是负数,都以二进制补码形式存储和调用。

补码是由机器系统自动转换生成的。正负数的补码转换方式不同,正数的补码就是原码,而负数的补码是该负数去掉负号以后的二进制原码取反后加 1。

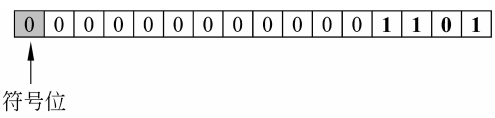
以有符号短整型数据为例,有如下定义:

```
short int valu1=13;
```

或

```
signed short int valu1=13;
```

转换为二进制:$(13)_{10}=(1101)_2$,则有符号短整型数据 13 在内存中的存储方式为



正数 13 的补码就是原码。

C 语言中有符号短整型数据存储的最高位用作符号位，用来表示数据的正负号，符号位置 0 值时表示该数为正数，置 1 值时则表示该数为负数，符号位不用于表示数据值。有符号短整型数据以符号位的值来确定正负。

若执行定义语句

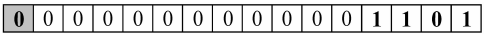
```
short int valu2=-13;
```

或

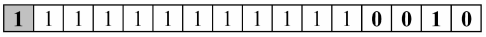
```
signed short int valu2=-13;
```

系统会自动将－13 转换为该负数的补码形式存储。负数转换过程如下：

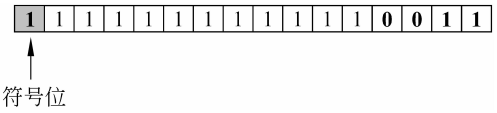
－13 去掉负号后，13 的二进制码为



取反后的二进制码为



加 1 后即转换成－13 的二进制补码，存储方式为



该值即为－13 的补码形式，最高位为 1，而低 15 位则表示该负数的补码值，本例 $(111\ 111\ 111\ 110\ 011)_2$ 就是－13 的二进制补码，即

$$(-13)_{10} = (1111\ 111\ 111\ 110\ 011)_2 = (fff3)_{16}$$

3.2.2 数据变量取值范围

整型变量的存储与字符型变量和浮点型变量不同，有符号变量和无符号变量也不同。数据类型一经定义，其存储方式就随之确定。不同编译系统规定的整型变量数值长度虽有不同，但基本原理是一样的。

1. 符号位问题

C 语言程序设计中，整型变量的基本类型为 int，加上不同的关键字，其存储方式和取值范围就会发生变化。

- 基本型：类型说明符为 int，在内存中占 2B 或 4B。
- 短整型：类型说明符为 short int 或 short，在内存中占 2B。
- 长整型：类型说明符为 long int 或 long，在内存中占 4B。

以上数据类型定义若不加 unsigned 关键字，则数据在内存中存储时以其最高位标识

数的正负号，以 0 表示正数，1 表示负数，该二进制位不作为数据值的表示位。

通常使用关键字 short、long 和 int 等定义整型数据，均默认为有符号存储方式。

C 语言还允许使用无符号整数，最高位不作为符号位，而用来表示数据值。无符号整型的说明符为 unsigned，与上述 3 种关键字配合构成以下类型：

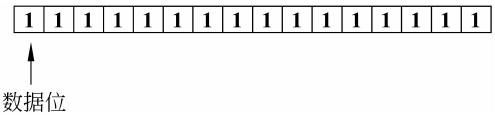
- 无符号基本型：类型说明符为 unsigned int 或 unsigned。
- 无符号短整型：类型说明符为 unsigned short。
- 无符号长整型：类型说明符为 unsigned long。

无符号整型数所占的内存空间与对应的有符号整型数相同，但由于最高位用来表示数据值，因此只能表示正数，不能表示负数，其正数取值范围扩大一倍。

例如，执行以下命令语句：

```
unsigned short int u_int;
u_int=65535;
```

由于$(65535)_{10}=(1111111111111111)_2$，在内存中的存储方式为



用 unsigned 定义的短整型变量，其数据在存储时的最高位不再用作符号位，而是用来表示数据值的数据位，这样正数取值范围为 0～65 535，比有符号位的正数取值范围 0～32 767 大一倍。

2. 数据溢出问题

在程序设计中，当数据值很大或很小时就可能会遇到数据溢出问题。定义变量时首先要注意带符号位的数据存储类型的取值范围。

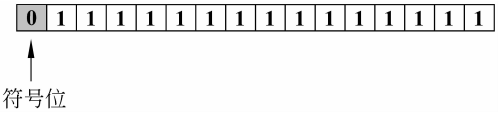
例如，执行变量定义命令语句

```
short int v_max=32767;
```

或

```
signed short int v_max=32767;
```

转换为二进制：$(32767)_{10}=(111111111111111)_2$，则有符号整型数据 32 767 在内存的存储方式为



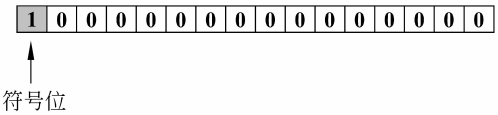
即在以二进制补码值表示有符号短整型的数据存储方式下，能够表示和存放的最大正整数为 $2^{15}-1=32\ 767$，即

$$(32767)_{10}=(0111111111111111)_2=(7fff)_{16}$$

此时若执行命令

```
v_max=32767+1;
```

则计算后的存储方式为



此时变量存储最高位置1，即符号位置1，表示该数是一个负数，这样数据就出现了最大值溢出错误。通常对此类问题系统不会提示出错，需要用户自己甄别。

下面再来看最小值数据溢出问题。

例如，执行变量定义命令语句

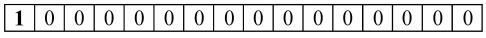
```
short int v_min=-32768;
```

或

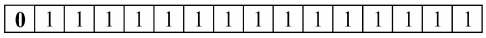
```
signed short int v_min=-32768;
```

系统自动将－32 768转换为该负数的补码形式存储。转换过程如下：

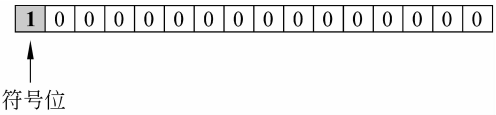
－32 768去掉负号后为32 768，其存储方式为



取反后的二进制反码为



反码加1后即转换成－32 768的二进制补码，存储方式为



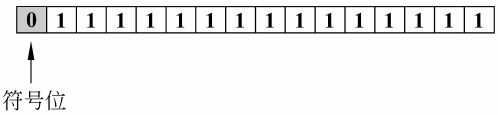
符号位为1表示这是一个负数，数据位均为0。显然，这个负数补码值是有符号短整型数据类型的存储方式下所能够表示和存放的最小负整数。即

$$(-32768)_{10} = (1000000000000000)_2 = (8000)_{16}$$

此时若执行命令

```
v_min=-32767-1;
```

则存储方式为



发生数据溢出，使符号位置0，而符号位置0又表示这是一个正数，数据位均为1显然是

短整型正数所能表示的最大值。

例 3-4 编写程序，验证常用的各种数据类型的存储方式，输出各变量的十进制值和十六进制值。

```
/*L3_4.c*/
#include "stdio"
#include "math.h"
main()
{
    signed short int valu2=-13;                          /*有符号变量初始化*/
    signed short int v_max=32767;
    signed short int v_min=-32768;
    unsigned short int u_int=65535;                      /*无符号变量初始化*/
    printf("valu2=%d, hexadecimal is%x\n",valu2,valu2); /*输出变量值,检验存储方式*/
    printf("v_max=%d, hexadecimal is%x\n",u_int,u_int);
    printf("v_min=%d, hexadecimal is%x\n", v_min, v_min);
    printf("u_int=%u, hexadecimal is%x\n",u_int,u_int);
    printf("v_min-1=%d, hexadecimal is%x\n",v_min-1,v_min-1);
    printf("v_max+1=%d, hexadecimal is%x\n",v_max+1,v_max+1);
    printf("unsigned short is%u,hexadecimal is%x\n",(int)pow(2.,16.)-1,(int)
    pow(2.,16.)-1);
}
```

注意，幂函数 pow()要求参数为浮点类型，输出结果需转换为整数以作比较，因此使用了整型强制类型转换运算(int)，以符合%u 无符号整数类型输出的要求。程序在 Turbo C 2.0 集成环境下编译运行后，输出结果如图 3-8 所示。

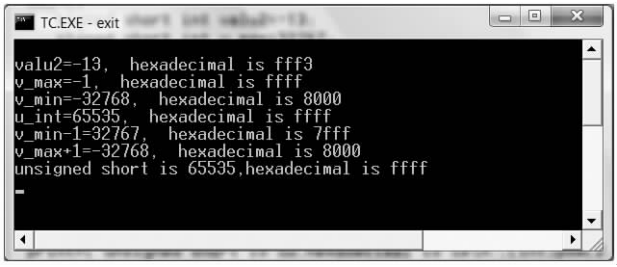


图 3-8 Turbo C 2.0 环境下的运行结果

在 Microsoft Visual C++ 6.0 集成环境下编译运行，输出结果如图 3-9 所示。

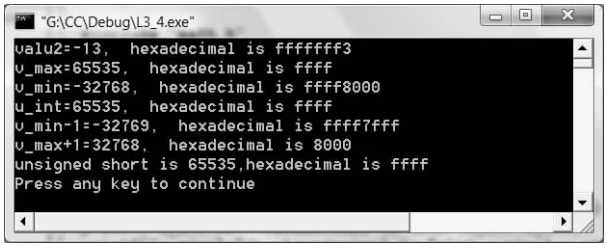
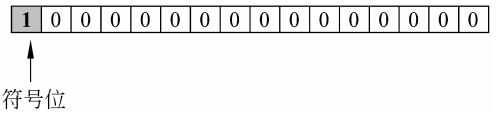


图 3-9 Microsoft Visual C++ 6.0 环境下的运行结果

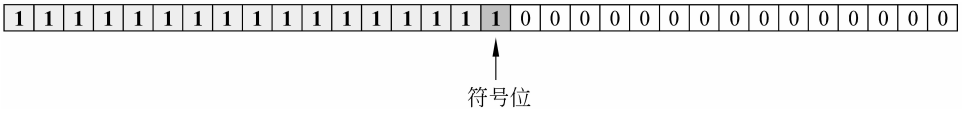
注意，负数以十六进制输出时为 4B，这是因为 Visual C++ 6.0 集成环境中普通整型占 4B。短整型虽然只占 2B，程序运行时仍取 4B。正数的符号位为 0，则 4B 最高位补 0，输出时省略；而负数的符号位为 1，则 4B 最高位补 1，运行后则原样输出。例如：

$$(-32768)_{10} = (1000000000000000)_2 = (8000)_{16}$$

−32 768 的二进制补码的存储方式为



在 Visual C++ 6.0 集成环境中为其分配 4B 存储空间，实际存储方式为



−32 768 的符号位置 1，程序运行时扩展为 4B 后，新扩展的 16 位均为 1，所以 Visual C++ 6.0 集成环境中−32 768 显示为十六进制 ffff8000。

综上所述，在程序设计中应注意定义变量的存储类型和存储方式，以便合理使用各种数据类型变量的取值范围。当赋值超过取值范围时就会出现数据溢出错误，这种错误在运行时并不报错，需要加以注意。

3.2.3 实型数据存储方式

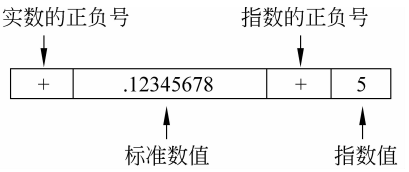
在计算机系统中，实型数据均以指数形式存储，一般占 4B(32b)内存空间，分两大部分，其中 3B 存放数据值部分，1B 存放指数值部分。例如，输入一个数 12345.678，由于计算机系统会自动转换成指数形式存储，这个数可以化成标准指数形式 0.12345678×10^5，包括数据值和指数值两部分：

$$\underbrace{0.12345678}_{\text{数据值}}\times10^{\overbrace{5}^{\text{指数值}}}$$

一个实数可以有多种指数形式，如 0.12345678×10^5 可以表示成 1.2345678×10^4、1234567.8×10^{-2}、1234.5678×10^1、12345.678×10^0 等，但只有 0.12345678×10^5 符合标准指数形式，即数据值部分小数点前第一位数字为 0，小数点后第一位数字不为 0。

1. 实数存储规范

实数在计算机系统中是按标准指数形式存储的。例如，实数 12345.678 在内存中的存储形式为



实数在计算机系统中最终以二进制形式存放，以字节表示。标准数值的小数部分占的位数越多，所表示数据的有效位数越多，精度越高；指数部分占的位数越多，则所能表示的数值范围越大。

2. 实数分类及有效范围

实数分为单精度(float)、双精度(double)和长双精度(long double)3 种类型，如表 3-4 所示。

标准 C 语言中，单精度型占 4B(32b)内存空间，其数值范围为 $3.4\times10^{-38}\sim3.4\times10^{38}$，只能提供 7 位有效数字。双精度型占 8B(64b)内存空间，其数值范围为 $1.7\times10^{-308}\sim1.7\times10^{308}$，可提供 16 位有效数字。长双精度型占 10B(80b)内存空间，有效数字为 18 位或 19 位，实际应用中根据计算机系统而定。

表 3-4　实数的类型及有效范围

类型说明符	字 节 数	有效数字位数	取值范围
float	4	7 位	$3.4\times10^{-38}\sim3.4\times10^{38}$
double	8	16 位	$1.7\times10^{-308}\sim1.7\times10^{308}$
long double	10	18 或 19 位	$1.2\times10^{-4931}\sim1.2\times10^{4932}$

3. 实型数据的舍入误差

由于实型变量存储方式所限，程序运行时所能提供的有效位也是有限的，运算中有效位被舍去就会产生舍入误差。

例 3-5　编写程序检验实型变量运算的数值有效位误差。

程序源代码：

```
/*L3_5.c*/
main()
{
    float a,b;
    a=12345.6789e6;                          /*赋值为较大的实数*/
    b=a+50;
    printf("a=%f\n",a);
    printf("b%lf\n",b);
}
```

程序编译运行后的输出结果如图 3-10 所示。

数学表达式 a+50 的理论值应是 12345678950，但由程序运行结果可以看出，变量 a

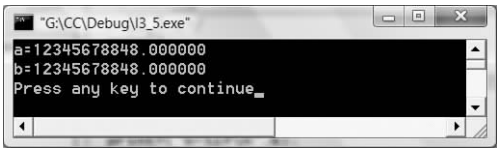


图 3-10　普通实型变量运算误差

的值与整数 50 相加再赋给变量 b 后，b 的输出值与 a 输出值是相同的。这是因为普通实型变量最多只能保证 7 位有效数字，后面的数字无实际操作意义，不能准确地表示和运算，所以运行程序得到的变量 b 值未变。一般应注意避免将一个很大的数和一个很小的数直接相加或相减，否则就会“丢失”较小的数。

若将变量定义 float a，b；改为 double a，b；，则输出结果发生变化，变量 b 输出有效值，如图 3-11 所示。这是因为双精度实型变量能保证 11 位有效数字，所以输出结果是准确的。

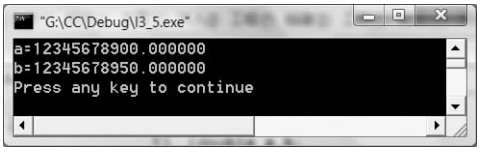


图 3-11　双精度实型变量减少运算误差

例 3-6　编写程序检验单精度和双精度实型变量舍入误差。

程序源代码：

```
/*L3_6.c*/
#include "stdio.h"
main()
{
    float a;
    double b;
    a=666666.666666;
    b=555555.55555555555555;
    printf("a=%f\n",a);
    printf("b=%lf\n",b);
    printf("b=%e\n",b);
 }
```

该程序分别在 Microsoft Visual C++ 6.0 和 Turbo C 2.0 系统环境下编译运行，输出结果如图 3-12 所示。

从本例运行结果可以看出，由于变量 a 是单精度浮点类型，有效位数只有 7 位，而 a 变量值整数已占 6 位，因此小数点后的数为无效数字；变量 b 是双精度类型，有效位为 16 位。C 语言在 Microsoft Visual C++ 6.0 和 Turbo C 2.0 系统环境下对实型数据分配的字节数和有效取值范围相同，规定小数点后最多保留6 位数，其余部分自动四舍五入。

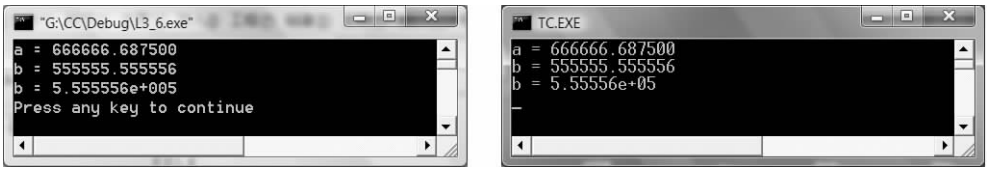


图 3-12　变量舍入误差比较

例 3-7　编写程序，定义各种数据类型变量，使各种类型数据混合运算，输出结果。

程序源代码：

```
/* L3_7.c */
main()
{
    char c='s';                                    /* 定义字符型变量并初始化 */
    int a=3,b=5;                                   /* 定义整型变量并初始化 */
    float x=2.3,y=4.5,z;                           /* 定义浮点型变量并初始化 */
    z=c+a+x+b* (x+y)/3;                            /* 各种类型变量和数据混合运算 */
    printf("c_Character=%c c_ASCII=%d\n",c,c);     /* 字符变量值输出 */
    printf("a+b=%d a+b=%o a+b=%x a+b=%u\n",a+b,a+b,a+b,a+b);
                                                   /* 以各种进制输出 */
    printf("x=%f y=%f\n",x,y);
    printf("z=%f\n",z);                            /* 输出混合运算结果 */
}
```

程序编译运行后的输出结果如图 3-13 所示。

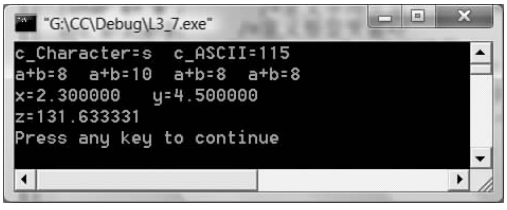


图 3-13　各种数据类型混合运算的结果

其中字符型变量 c 以字符和 ASCII 码值两种方式输出；整型算术运算表达式 a+b 分别以十进制、八进制、十六进制和无符号类型输出；实型变量 x、y 值以浮点类型输出，低位补 0；混合运算结果以浮点类型输出，有效位数为 7 位。

3.2.4　字符型数据存储方式

字符型数据与数值型数据不同，例如字符'5'与数字 5 是两个不同的常量，'5'是字符常量，本身不能作为实际数值参与数学运算。

字符变量的类型定义说明符是 char，用来存储单个 ASCII 字符或转义字符。字符变量在内存中占一个字节，每个字符变量被分配一个字节的内存空间，字符值是以 ASCII