第3章 数据类型基础

程序设计的目的是为了处理数据。数据的类别是各种各样，如数字、文字、图像、声音和视频等。我们将这些数据输入到计算机，希望计算机能够按照我们的要求来处理这些数据。例如，银行需要计算机计算客户需要支付的利息，或者教师期望查看经过排序的学生成绩等。

C语言是通过运算符号来处理各种类型数据的。C数据类型可以分为基本类型、构造类型、指针类型和空类型4大类，同时C语言拥有丰富的运算符号。

本章主要介绍数据在计算机中的存储方式、数据的基本类型、变量和常量以及算术运算符号与表达式。其他数据类型和运算符号将在后面介绍。

3.1 数据在计算机中的存储方式

3.1.1 二进制

数据在计算机中是以二进制的形式存储的，而通常人们书写数字的常用方法是十进制。例如十进制：

$$1234=1\times1000+2\times100+3\times10+4$$

我们知道，$1000=10^3$，$100=10^2$，$10=10^1$，$1=10^0$，所以上面式子也可以表示为：

$$1234=1\times10^3+2\times10^2+3\times10^1+4\times10^0$$

把10称为基数，上述表示方法是基于10的幂。但是计算机只能识别二进制数，原因是它只能被设为0或1，即关闭或者打开。因此以2为基数的系统适应于计算机，它用2的幂代替10的幂。以2为基数表示的数字称为二进制数。数字2对于二进制数的作用和数字10对十进制数的作用是相同的。例如二进制数1101的表示形式如下：

$$1101=1\times2^3+1\times2^2+1\times2^0$$

可以将任何整数表示为1和0的组合。二进制数表示不直观方便，因此有时也将其表示为八进制和十六进制，编译器会自动将其转换为二进制形式存储。关于二进制、八进制、十六进制和十进制的转换等相关知识可以参考其他学习资料，在此不一一介绍。

3.1.2 位与字节

上面提到，计算机中的数据都是以二进制的形式存储的。二进制中的0和1被称为bit（比特，又称为位），bit是binary digit二进制的缩写。各种类型的数据，例如数字、字符和字符串，都被编码为比特序列。CPU要执行的所有指令和数据都以一组比特或者字节的形式

保存在计算机的内存里。内存的存储单元是一个线性地址表，是按字节进行编址的，即每个字节的存储单元都对应着一个唯一的地址。例如图 3.1 中，2000 至 3010 即为存储单元的地址。一般用字节来描述一个数据保存到内存中时所占的空间大小。

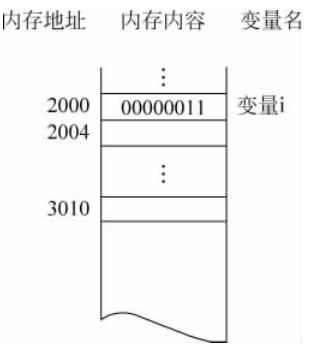

图 3.1　内存地址与存储内容

字节是最小的存储单元，一个字节等于 8 个二进制位，一个二进制位的值只能是 0 或者 1。像 3 这样小的数字就可以存储在单个字节中，它的二进制表示为 00000011，见图 3.1。通常用 8 个二进制位来表示一个数据，可以表示的整数最小值是 0，最大值是 255。为了表示更大的数字，需要将更多的字节组合起来使用，两个字节可以表示 65 536 个不同的值，4 个字节可以表示超过 40 亿个不同的值。

内存中字节的内容永远为非空，但是它的原始内容可能对于你的程序来说是毫无意义的。一旦新的信息被放入内存字节，该字节的当前内容就会丢失。内存保存数据快，但具有挥发性，即掉电即失。

一般来说，用户无法也没有必要直接看到计算机上的位和字节，因为数据是被自动转换为用户能够识别的数字和字符了。

3.1.3　数据的存储方式

程序及其所需数据必须在它们被执行之前放入内存。不同类型数据的存储方式是不同的。数据处理中常用的数据类型是数值型，包括整型和浮点型。下面简单介绍这两种类型数据在内存中的存放形式。

1. 整型数据在内存中的存放形式

C 语言中的整型数据以数据的补码形式在内存中存放。C 标准没有具体规定各整型数据所占内存字节数，为简单起见，下面以 2 个字节为例讲解。

如果定义了一个整型变量 i 如下：

```
int  i ;    i = 10 ;
```

它在内存中存储形式如图 3.2 所示。

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

图 3.2　正整数 10 在内存中的存放

因为正整数的补码和原码相同。

负数的补码：将该数的绝对值的二进制形式按位取反再加 1。－10 在内存中的存放形式如图 3.3 所示。

1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0

图 3.3　负整数 10 在内存中的存放

求－10 的补码过程如图 3.4 所示。

图 3.4　求－10 的补码过程演示图

由此可知,左面的第一位是表示符号的,其为 0 表示正数；其为 1 表示负数。

2. 浮点型数据在内存中的存放形式

浮点型数据,又称实型数据,可借鉴科学记数法的形式表示为小数和指数两部分。系统会把一个浮点型数据分成小数部分和指数部分分别存放,并采用规范化的指数形式。至于在内存中究竟用多少位来表示小数部分,多少位来表示指数部分,C 标准并无具体规定,由各 C 语言编译系统自定。通常使用 IEEE 浮点数标准(Floating-Point Standard)754 格式,如图 3.5 所示。

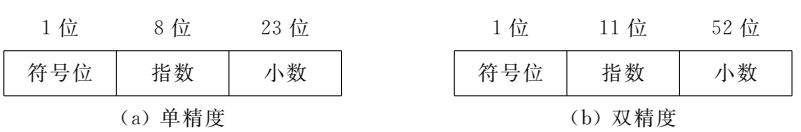

图 3.5　IEEE 浮点数格式

精度及范围定义了能够被浮点型数据类型表示的值的集合。精度是一个值的小数部分的准确性,以位的数目来衡量,通常用有效数字个数表示。范围是小数范围和指数范围的组合,对浮点型数据的范围来说,指数范围更重要。小数部分占的位(bit)数愈多,数的有效数字愈多,精度也就愈高。指数部分占的位数愈多,则能表示的数值范围愈大,详细内容见表 3.1。

表 3.1　浮点型数据的相关数据

类　　型	位　　数	有 效 数 字	数 值 范 围
Float	32	6～7	$-3.4\times10^{38}\sim3.4\times10^{38}$
double	64	15～16	$-1.7\times10^{308}\sim1.7\times10^{308}$

3.2　常量与变量

3.2.1　基本概念

首先,来看一个计算圆的面积的简单问题。该如何编写程序解决这个问题呢?

【例 3.1】 程序的算法描述如下：

(1) 读入半径；

(2) 利用求面积的公式：面积＝半径×半径×π；

(3) 显示面积。

显然，在上述例子的算法转换为程序后，需要用户从键盘输入圆的半径，并计算存储圆的面积。这就产生了三个重要的问题：

(1) 读取半径；

(2) 存储半径；

(3) 计算并存储圆的面积。

半径的值是不确定的，为了存储半径，在程序中需要声明一个称作变量的符号。变量指定在程序中用于存储数据和计算结果的内存位置。

程序如下：

```
#include <stdio.h>
int main()
{
  double radius,area;                          //变量的声明
  scanf("%lf",&radius);                        //从用户处获得输入
  area = radius * radius * 3.1415926;          //求圆的面积
  printf("圆的面积 = %f\n",area);
  return 0;
}
```

有了设计好的程序，计算机就可以按照程序的指示帮助人们做很多事，比如成绩统计、播放语音或视频、计算航天飞机的轨道、发送邮件、画图以及任何你能想象到的事情。当然，要完成这些任务，程序需要处理与此有关的数据。

数据是信息的载体，任何数据其呈现方式都有两种：**常量和变量**。在程序运行之前预先设定并在整个程序运行期间其值不变的量称为常量；还有一些在程序运行过程中其值可以变化的量称为**变量**。比如在例 3.1 中，radius 和 area 都是双精度浮点型变量，可以将任意值赋给它们。而常量圆周率 π 的值是不变的，近似于 3.14159265375。

3.2.2 定义常量的名字(预处理命令#define)

C 语言中，无论是数字常量、字符常量，还是字符串常量，都可以用便于记忆的符号来表示。用符号表示的常量被称为符号常量。符号常量的符号可以是任意的合法标识符。习惯上往往使用由大写字母和下划线组成的标识符。使用命令 #define 可以定义一个符号和它所代表的常量。其一般形式如下：

```
#define 标识符 常量
```

例如：

```
#define PI 3.14159265375
```

上例定义了一个符号常量 PI，它代表的是圆周率 π 的近似值 3.141 592 653 75。在此后的表达式中用到 π 的近似值的地方，就都可以用符号常量 PI 来表示，而不必写 3.141 592 653 75

了。前面的程序改写如下：

```
#include <stdio.h>
#define PI 3.14159265375
int main()
{
  double radius,area;                          //变量的声明
  scanf("%lf",&radius);                        //从用户处获得输入
  area = radius * radius * PI;                 //求圆的面积
  printf("圆的面积 = %f\n",area);
  return 0;
}
```

符号常量的定义也可以是常量表达式，在表达式中可以包含已定义过的符号常量。例如：

```
#define NUMBER 100
#define  RESULTS  (NUMBER/5)
```

这个符号常量 RESULTS 的值就等于 20。这样定义具有关联关系的符号常量可以提高程序的可维护性。在上面这个例子中，只要 RESULTS 和 NUMBER 之间的关系不变，当由于程序的修改而需要改变 NUMBER 和 RESULTS 的值时，只需要改变 NUMBER 的定义即可。

#define 是个编译预处理命令，它所定义的符号常量在预编译阶段被替换为对应字符串。定义符号常量时，对其所对应的值的类型和格式没有任何特殊要求。因此，符号常量不仅可以表示数值，也可以表示字符或字符串等其他类型的常量。

使用符号常量有助于提高程序的可读性，便于记忆和使用。在符号常量定义时，要根据常量的用途和含义对常量命名。良好设计的符号常量名，可以提示常量的用途，避免误用。这样不但便于编程人员记忆，而且便于其他人员阅读和理解程序。同时，也有助于发现和减少错误。符号常量和变量名字的命名规则与第一章讲到的 C 语言中表示符号的命名规则是一样的，但习惯上字母都是大写。

3.2.3 变量的声明和赋值

每一个变量都必须有一个数据类型。类型用来说明变量所存储的数据的种类。在使用变量之前必须对其进行声明。声明的一般形式如下：

类型说明符 变量名表;

例如，例 3.1 程序代码中对圆的半径(radius)和面积(area)的声明：

```
double radius,area;
```

其中 double 是指定义的数据类型属于浮点数据类型。关于基本数据类型将在 3.3 小节里介绍。

使一个变量保存一个确定的数值的操作称为向这个变量赋值。赋值的操作符是“=”。操作符的左侧是变量名，右侧是一个表达式。赋值操作将表达式的值赋给左侧的变量。例如，例 3.1 程序代码中，area 的值是 radius * radius * 3.1415926。

```
area = radius * radius * 3.1415926;                    //求圆的面积
```

当然，也可以给 radius 赋一个常量，比如：

```
radius = 12.2;
```

也可以在变量定义的同时给变量赋以初值，称为变量的初始化。在变量定义中赋初值的一般形式如下：

```
类型说明符 变量1 = 值1,变量2 = 值2,…;
```

例如：

```
int a = 3;
int b,c = 5;
float x = 3.2,y = 3f,z = 0.75;
char ch1 = 'K',ch2 = 'P';
```

在C语言中，如果在变量定义的时候没有为其初始化，那么变量所表示的存储空间也并为空，而是一些无用信息，也称为垃圾数据。在为其赋值后，新的数据会覆盖原来的垃圾数据。

变量是程序设计语言中的重要概念。通常，一个变量可以用 6 种属性来刻画，即名字、地址、值、类型、生存期、作用域，也就是说可以从这 6 个方面来理解变量。其中生存期、作用域在后续章节讲解。

首先C语言中的变量有它的名字和它所代表的值，这一点与代数中的情况有点相似。变量的名字称为变量名或变量标识符，是以字母或下划线开头的，由字母、数字、下划线组成的字符串。用标识符给变量命名时，应遵循“见名知义”的原则。a、ab、_cb、_6、y8 等都是合法的变量名，而 3m、7b、l+k、a.3 等都不是合法的变量名。同时要注意，C语言是严格区分大小写的，A3 和 a3 表示不同的标识符。此外，C语言中保留了一些有特殊用途的字符串，如 auto、else 等，也不能用作变量名。ANSI C 一共定义了 32 个关键字，见表 3.2。

表 3.2　C语言的关键字

auto	break	case	Char	const	Continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	static	sizeof	struct	switch	typedef	union
unsigned	void	volatile	While			

其次，变量是按其名字读写的数据存储空间，实际上是一个符号地址，同时变量是具有类型的。因此在定义变量时，不但需要指定变量的名字，还要指定变量的类型。不同类型的数据在计算机中的表达方式不同，所占用的内存空间大小不同，所表示的数值范围和精度不同，不同类型的变量所能进行的操作也不相同。编译器在碰到变量定义语句时，根据数据类型，给变量分配内存。变量值是内存单元中存储的数据。图 3.1 清楚地表示了变量地址、变量的值与变量的名称之间的关系。

使用变量时需要注意的问题是，在C语言中，变量必须说明在先，使用在后，而且所有变量的说明必须在使用该变量的第一条语句之前。例 3.1 中，如果语句 double radius，area

放置在scanf("%lf",&radius)后面,那程序执行是错误的。

3.2.4 常量的分类

根据数据的类型,C语言中的常量可以分为数值常量、字符常量、字符串常量和符号常量等。数值常量又包括整型数值常量和实型数值常量。

1. 整型数值常量

整型数值常量指整常数,即没有小数部分的数。对于整数,在C语言中可以用十进制方式表示,也可以用十六进制或八进制方式表示。当没有任何其他说明时,一个整数是按十进制方式解释的。如果一个整数前有前缀0x,则说明这是一个十六进制的数。十六进制数中用a、b、c、d、e和f(大小写均可)分别表示十进制的10～15。如果一个整数以0开头,则说明这是一个八进制的数。八进制数中只允许出现数字0～7。下面是几个整数常量的例子,它们所代表的都是同一个数值,只是使用不同的数制来表示:

```
90                                  //十进制数90
0x5a                                //十六进制数5a,等于十进制数90
0132                                //八进制数132,等于十进制数90
```

任何一个整数都可以用上述3种形式表示,表示形式只是它的外观,并不改变它的数值。在计算机系统内部,都是以二进制形式存储的。如:以1个字节为例,十进制整数26、八进制整数032、十六进制整数0xlA在计算机内部都是二进制数00011010。

对于负数,在第一个数字前面应加负号(-),如-123等。当使用十六进制或八进制方式表示时,其所直接表示的是数据的二进制位,其中也包括数据的符号位,因此一般情况下在十六进制或八进制数前不直接使用负号。

此外,在描述一个整数时,也可以同时说明该数在计算机中的保存格式。一个整数常量在计算机中可以保存为普通整数和长整数。长整数需要在数字后面加上后缀L,例如,56L、0x12345L、0L等都是长整数。普通整数则不加后缀。普通整数和长整数的区别取决于具体的计算平台。在有些计算平台上,这两种保存方式是相同的,而在有些计算平台上,表示长整数所使用的二进制位要多于普通整数。

2. 实型数值常量

实型数值常量是指存在小数点的数据,即实数,也称浮点数。对于实数,在C语言中有两种基本的表示方法。

第一种方法和日常算术中的书写方式相同,即直接使用十进制数字表示数据的整数和小数,并使用小数点分隔数字的整数部分和小数部分。与常规方法略有不同的是,当整数或小数部分为0时,相应的部分可以不写数字0,而只写上小数点。下面是这种表示方法的几个例子:

```
0.12
.26
1.23
6.
```

第二种方法被称为科学记数法。这种方法把一个数据表示为有效数字部分和指数部分。有效数字部分可以是整数,也可以是用常规方式表示的小数。指数部分是以字母e或

E 开头的整数，可以带正负号，表示有效数字所要乘以的 10 的幂。下面是几个用科学记数法表示的小数：

```
0.12E3                                  //120.0
5.6E-6                                  //0.0000056
-6.9E10                                 //-69000000000
```

整数和实数在计算机中的保存方式不同，因此尽管没有小数部分的实数表示的是一个整数，但是它在数据类型上并不等同于整数。

此外，同一个实数，在 C 语言中也有两种不同的保存类型：一种是单精度实数，另一种是双精度实数。两种数据类型在数值的表示范围、数据的有效数字位数，以及所占用的存储空间等方面都不相同。C 语言中默认的实数类型是双精度类型，上面的几个例子所表示的都是双精度实数。当需要描述单精度实数时，需要在数据后面加上后缀 f 或 F，下面是几个单精度实数的例子：

```
4.5f
6.8F
0.356E3f
5.6E-4F
```

3. 字符常量

字符常量是用一对单引号引起来的 1 个字符，表示引号中的单个的字符。字符常量既可以是能够显示和打印的可见字符，如字母、数字、标点等，也可以是不可见的特殊字符，例如控制光标在终端屏幕上移动位置、使终端发出振铃声音以及控制打印机走纸等的各类字符。可见字符可以直接用其自身表示，如'A'、'='等。

C 语言还允许使用一种特殊形式的字符常量，就是以一个字符“\”开头的字符序列，称之为“转义字符”，如表 3.3 所示。

表 3.3　转义字符及其含义

转义字符	含　义	ASCII 码
\n	回车换行，将光标移到下一行开头	10
\t	水平制表(将光标移到下一个 Tab 位置)	9
\b	退格，将光标移到前一列	8
\r	回车，将光标移到本行开头	13
\f	换页，打印时将光标移到下一页开头	12
\\	代表反斜杠符号“\”	92
\’	代表单引号字符	39
\”	代表双引号字符	34
\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	

无论是否是可见字符，都可以用它的内部编码值构成转义序列来表示。由 ASCII 码值构成的转义序列可以用十六进制或八进制两种表示方法。当使用十六进制方式时，转义序列形式是'\xhh'，其中 hh 是一个或两个十六进制数(0～9，a～f，A～F)。当使用八进制方式时，转义序列的形式是'\ddd'，其中 ddd 是一个、两个或三个八进制数(0～7)。

这样,一个字符可以有多种表示方式。例如,'a'、'\x61'、'\141'、0x61 等都表示字符 a。对于可见字符,一般使用单引号将字符直接引起来的方式;对于不可见字符,一般使用单引号引起字符转义序列的方式。下面是几个字符常量的例子:

```
'0'                                    //数字字符 0
'Z'                                    //十六进制大写字母 Z
'\n'                                   //换行符
'\31'                                  //数字字符 0
'\071'                                 //数字字符 9
```

4. 字符串常量

字符串常量是一对双引号引起来 0 个或多个连续的字符序列。

C 语言对字符串的长度没有限制,字符串在计算机中以一个空字符'\0'结束,但是在字符串常量中不需要直接表示这个空字符。字符串两端的双引号不是字符串的组成部分,它只是字符串的界限符。当字符串中包含双引号时,需要用"\"来表示。字符串中可以包含转义序列。下面是几个字符串常量的例子:

```
"This is a string\n"                   //This is a string 加换行符
"\" is  \x61 double quote"             //" is a double quote
```

当多个字符串连续出现时,编译系统自动地把它们连接在一起。这样,如果字符串比较长的话,可以把它分成若干段,分别写在若干连续的行中。例如,一个很长的字符串,就可以把它分成几部分,分行书写:

```
printf ("This is a long string so "
"we divided it into three lines\n"
"and output it in two lines\n");
```

需要注意的问题是,不要将字符常量与字符串常量混淆。因为 C 语言规定以字符'\0'作为"字符串结束标志",即在每一个字符串常量的结尾加一个字符串结束标志'\0',并以此判断字符串是否结束。'\0'是一个 ASCII 码为 0 的字符,从 ASCII 码表中可以看到 ASCII 码为 0 的字符是"空操作字符",即它不引起任何控制动作,也不是一个可显示的字符。例如'a'是字符常量,在内存中需要一个字节存储;"a"是字符串常量,在内存中除了需要一个字节存储字符'a'外,还需要一个字节存储其字符串结束标志'\0',即存储"a"需要两个字节。二者在内存中的存储方式和存储空间不一样。

设 ch1 被指定为字符变量:

```
char ch1;
ch1 = 'a';
```

是正确的。而

```
ch1 = "a";
```

则是错误的。不能把一个字符串常量赋给一个字符变量。

如果有一个字符串常量"China",实际上它占内存单元 6 个字节,而不是 5 个字节,最后一个字节中存放字符串结束标志'\0',如图 3.6 所示。但在输出时不输出'\0'。字符串输出时,从第一个字

C	h	i	n	a	\0

图 3.6　字符串的存储

符开始逐个输出字符，直到遇到空操作字符'\0'，才停止输出。

注意：在写字符串时不必加'\0'，因为'\0'字符是系统自动加上的。在C语言中没有专门的字符串变量，如果想将一个字符串存放在变量中以便保存，必须使用字符数组，即用一个字符数组来存放一个字符串，数组中每一个元素存放一个字符。这在后面内容中介绍。

3.3 基本数据类型

无论常量还是变量，都必须属于某种数据类型。前面我们讲过，数据的类别各种各样，计算机需要一种方法来区分和使用这些不同的类型。C通过一些基本的数据类型来做到这一点。对于常量数据，一般是编译器通过其书写来辨认其类型，比如10是整数，10.000是浮点数。在C语言中，数据类型可以分为基本类型、构造类型、指针类型和空类型4大类，如图3.7所示。

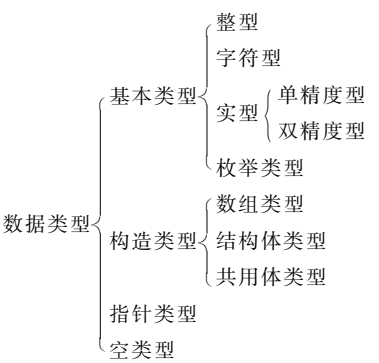

图3.7 C语言中的数据类型

具有相同类型的数据具有相同的性质：具有相同的编码方式、相同的取值范围以及能够进行的操作。

下面详细介绍C使用的基本数据类型。C的基本数据类型包括整型、浮点型、字符型和枚举类型。本章将介绍前面三种基本数据类型，枚举类型将在11章进行介绍。对于每一种数据类型将介绍常量的定义方法、变量的声明以及典型的用法。

3.3.1 整型

最常用的基本数值数据类型是整型，用关键字int来表示。

这和数学上的整数有点类似，整型是没有小数点的数值数据，如5、123、－7892等都是整型数据。C标准没有具体规定整型数据的存储空间大小，不同的编译器对整型数据分配不同大小的存储空间。

1. 整型数据的分类

根据所占存储空间的长度，整型数据细分为短整型、整型(普通整型)、长整型，分别用关键字short、int、long表示。

根据数据有无符号，整型数据可划分为有符号整型数据和无符号整型数据，分别用关键字signed和unsigned表示。通常省略时默认为有符号数据。如此组合起来，整型数据就有表3.4所示的6种组合形式。

表3.4 整数类型的分类

类型名称	类　型	变量声明实例
基本整型	[signed] int	int i;
无符号基本整型	unsigned int	unsigned int i;
短整型	short [int]	short int i;或 short i;
无符号短整型	unsigned short	unsigned short i;
长整型	long [int]	long int i;或 long i;
无符号长整型	unsigned long	unsigned long i;

2. 使用整型数据的注意问题——数据的溢出

【例 3.2】 有符号整型数据的溢出。

程序如下：

```
#include<stdio.h>
int main()
{
   short int a,b;
   a=32767;
   b=a+1;
   printf("%d,%d\n",a,b);
   return 0;
}
```

运行结果：

```
32767,-32768
```

【例 3.3】 无符号整型数据的溢出。

程序如下：

```
#include<stdio.h>
int main()
{
   unsigned short int a,b;
   a=65535;
   b=a+1;
   printf("%d,%d\n",a,b);
   return 0;
 }
```

运行结果：

```
65535,0
```

请思考为什么出现这样的运行结果？

程序分析：由程序的执行结果可见，例3.2代码声明有符号短整型数据a和b，且为a赋该类型数据最大值(32 767)，a+1后超出了有符号数的范围，回到该类型数据最小值(−32 768)。例3.3代码说明无符号短整型数据a和b，a赋值无符号数据最大值(65 535)，a+1后超出了无符号数的范围，超出部分归0，回到无符号数最小值0。

3.3.2 实型

实型也称为浮点型，表示带小数点的数值数据。如2.5、12.0、−56.12等都是实型数据。

1. 实型数据的分类

浮点型数据通常分为单精度浮点型数据、双精度浮点型数据和长双精度数据，分别用关键字float、double和long double来表示，如表3.5所示。

表 3.5　浮点型数据的相关数据

类 型 名 称	类　　型	变量声明实例
单精度实型	float	float f;
双精度实型	double	double d;
长双精度实型	long double	long double d;

2. 使用实型数据类型注意问题

浮点型数据被存储为位数有限的二进制数，这使得浮点型数据的表达只是一种近似。例如，即使连十进制的 0.1 都不能用有限的二进制数来表示。这同时导致了浮点型数据的另一个问题——算术运算之后丢失精确度。因此，使用实型数据类型注意以下问题：

(1) 两个实型数据间不能进行相等比较。

(2) 实数范围内，应当避免将一个很大的数和一个很小的数直接相加或相减，否则就可能表达不准确，“丢失”小的数。因为数据位数超出有效数字个数时，超出部分将舍弃，由此产生一定的误差。当然，实型数据超出范围，超出部分将丢失。

【例 3.4】 实型数据的精度示例。

程序如下：

```
#include<stdio.h>
int  main()
{
   float a,b;
   a=123456789.0;
   b=a+20;
   printf("a= %f,b= %f\n",a,b);
   return 0;
 }
```

运行结果：

```
a=123456792.000000,b=123456812.000000
```

请思考为什么出现这样的运行结果？

3.3.3　字符型

字母 A 到 Z、数字 0 到 9、%、& 等都可以认为是一个字符型数据，C 语言中用关键字 char 来表示字符型数据，用 1 个字节来存储一个字符型数据。

字符型数据以数值编码的形式存储于计算机中。目前，普遍使用的是 ASCII 码。

以 ASCII 码形式存储的字符型数据，每个字符都对应一个 ASCII 码，如字母 a 的 ASCII 码为 97。要存储该字符时，系统用 1 个字节来存储该字符的 ASCII 码，就像存储整型数据 97 一样。显示时，如果以字符形式显示，则显示字母 a，如果是以整型数据显示，则显示 97。

【例 3.5】 分别以字符形式和整数形式输出字符'5'。

程序如下：

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

```
int main()
{
   char c1;
   c1 = '5';
   printf("字符型输出 c1 = %c\n",c1);
   printf("整型输出 c1 = %d\n",c1);
   return 0;
 }
```

运行结果:

```
字符型输出 c1 = 5
整型输出 c1 = 53
```

以 ASCII 码的形式存储字符,与整型数据在计算机系统中的存储形式一致,故 C 语言允许字符型和整型数据通用。请看以下示例。

【例 3.6】 字符数据和整型数据通用示例。

程序如下:

```
#include<stdio.h>
int main()
{
   char c1,c2 = ' ';
   c1 = 'A';
   c2 = c1 + 32;
   printf("c1 = %c, %d, %o, %x\n",c1,c1,c1,c1);
   printf("c2 = %c, %d, %o, %x\n",c2,c2,c2,c2);
   return 0;
 }
```

运行结果:

```
c1 = A,65,101,41
c2 = a,97,141,61
```

3.3.4 sizeof()求类型大小

C 标准并未规定各种不同的类型数据在内存中所占的字节数,只是简单地要求长整型数据的长度不短于基本整型,短整型数据的长度不长于基本整型。另外,同种类型的数据在不同的编译器和计算机系统中所占的字节数也不尽相同,因此,绝不能对变量所占的字节数想当然。

如果要准确计算某种类型数据所占的字节数,需要使用 sizeof()运算符。sizeof()是 C 语言提供的专门用于计算指定数据类型字节数的运算符,它有 2 种形式:

```
sizeof(类型名)
sizeof(表达式)
```

功能: 测量某个数据或某种数据类型在计算机系统内部占用的字节长度,属于单目运算符,其运算结果是整型数据。

说明：表达式尽量加括号，以避免误会，同时也培养良好的程序设计风格。如：

(1) sizeof (float)表示测量 float 类型在计算机系统内部占用的字节长度；

(2) sizeof(a+b)表示测量 a+b 的数据类型在计算机系统内部占用的字节长度；

(3) sizeof(a) + b 表示测量 a 在计算机系统内部占用的字节长度，再将此值与 b 相加。

3.4 数据类型转换

3.4.1 自动转换

在 C 语言中，整型(包括 int、short、long)数据和实型(包括 float、double)数据都是数值型数据，字符型数据又与整型通用，因此整型、实型、字符型数据间可以进行混合运算。要实现混合运算，就必须进行类型转换，转换的目的是：

(1) 将短的数扩展成机器处理的长度。

(2) 使得运算符两侧的数据类型相同。

例如：下面式子在 C 语言中是能够通过类型转换进行计算的。

```
63 + 'a' - 16 * 8 + 15.95
```

在进行运算时，不同类型的数据要先转换成同一类型，然后进行运算。转换的规则如图 3.8 所示。

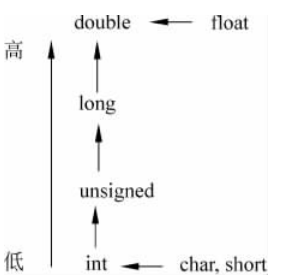

图 3.8 各种数值类型数据间的转换规则

(1) 图 3.8 中横向向左的箭头表示必定的转换。char 型与 short 型必定先自动转换为相应的 int 型，float 型数据在运算时一律先转换成 double 型。

(2) 纵向的箭头表示当运算对象为不同类型时转换的方向。例如，int 型与 double 型数据进行运算，先将 int 型的数据转换成 double 型，然后在两个同类型(double 型)数据间进行运算，结果为 double 型。

(3) 箭头方向只表示数据类型级别的高低，由低向高转换。不要理解为 int 型先转换成 unsigned 型，再转换成 long 型，再转换成 double 型。如果一个 int 型数据与一个 double 型数据运算，是直接将 int 型转换成 double 型。同理，一个 int 型与一个 long 型数据运算，先将 int 型转换成 long 型。换言之，如果有一个数据是 float 型或 double 型，则另一数据要先转换为 double 型，运算结果为 double 型。如果参加运算的两个数据中最高级别为 long 型，则另一数据先转换为 long 型，运算结果为 long 型。其他依此类推。

(4) 如果一个运算符的两侧的数据类型不同，先自动进行类型转换，使二者具有同一种类型，然后进行运算。

由此可知，运算式“63+'a'-16 * 8+15.95”中 15.95 为 double 型，'a'为 char 型，63、8 和 16 为 int 型。

在计算机执行时从左至右扫描，运算次序为：先进行 63+'a'的运算，先将'a'转换成整数 97，运算结果为整数 160。由于乘法优先于减法，先进行 16 * 8 的运算，由于 16 和 8 类型相同且为 int 型，不需转换，计算结果为整数 128。整数 160 与 16 * 8 的积 128 相减，结果为整数 32。计算 32 和 double 型数据 15.95 相加，先将整数 32 转换为 double 型(其值为 32.0，

小数点后加有若干个 0),结果为 double 型数据。

这是一般算术运算转换,这种类型的转换通常是由系统自动进行的。同样属于自动转换的还有赋值转换和输出转换等。

3.4.2 强制类型转换

强制类型转换运算是指由编程者按照需要将一种类型数据显式地转换成所需类型数据的方式。强制类型转换运算的一般形式如下:

```
(类型名)(表达式)
```

例如:

(double)('A')表示将'A'转换成 double 类型。

(int)(3.256)表示将 3.256 的值转换成整型。

(int)(x1+x2)表示将 x1+x2 的和转换成整型。

(float)(8+3)表示将 8+3 的值转换成 float 型。

需要转换的表达式应该用括号括起来。在强制类型转换时,得到一个所需类型的中间变量,原来变量的类型未发生变化。如:

```
float  x=3.6;
int a;
a=(int)x;
```

已定义 x 为 float 型,其值为 3.6,进行强制类型运算后得到一个 int 型的中间变量,它的值等于整数部分 3,将其赋值给变量 a,a 的值为整数 3。而变量 x 的类型仍为 float 型,值仍等于 3.6。

从上可知,有两种类型转换,一种是在运算时不必用户指定,系统自动进行的类型转换,如"3+6.3;",另一种是强制类型转换。当自动类型转换不能实现目的时,可以用强制类型转换。当数据由高级类型转换为低级类型时,很容易损失精度,例如上述语句 a=(int)x。

注意:强制类型转换实际上是一种单目运算。各种数据类型名都可以用来作强制类型转换的运算符,如(char)、(int)、(float)等。

3.5 运算符与表达式

前面我们已经熟悉了表示数据的方式,下面介绍如何处理数据。C 语言的一个特点是它更多地强调表达式而不是语句,表达式是表示如何计算值的公式。最简单的表达式是变量和常量。更加复杂的表达式把运算符用于操作数。

只需一个操作数的运算符称为单目运算符;需要两个操作数的运算符称为双目运算符;需要三个操作数的运算符称为三目运算符。

C 语言的运算符不仅具有不同的优先级,而且还有一个显著的特点,就是它的结合性。在表达式中,各运算量参与运算的先后顺序不仅要遵守运算符优先级别的规定,还要受运算符结合性的制约,以便确定是自左向右进行运算还是自右向左进行运算。这种结合性是其他高级语言的运算符所没有的,这在一定程度上也增加了 C 语言的复杂性。

C语言拥有丰富的运算符，运算符是构建表达式的基本工具。本章主要学习算术运算符号和自增自减运算符号。

3.5.1 算术运算符

(1) 加法运算符＋：加法运算符为双目运算符，即应有两个量参与加法运算。如a＋b、4＋8等，具有左结合性(详见3.5.4节)。

(2) 减法运算符－：减法运算符为双目运算符，如a-b，4-8等，具有左结合性。但“－”也可作负值运算符，此时为单目运算，如－x、－5等具有右结合性(详见3.5.4节)。

(3) 乘法运算符＊：双目运算符，具有左结合性。

(4) 除法运算符/：双目运算符，具有左结合性。参与运算量均为整型时，结果也为整型，舍去小数。如果运算量中有一个是实型，则结果为双精度实型。

(5) 求余运算符(模运算符)“％”：双目运算符，具有左结合性。要求参与运算的量均为整型。求余运算的结果等于两数相除后的余数。

【例3.7】 检验整型数的各种运算。

程序如下：

```
#include<stdio.h>
int main()
{
    int x,y;
    x = 7;
    y = 3;
    printf("x+y=%d,x-y=%d\n",x+y,x-y);
    printf("x*y=%d,x/y=%d\n",x*y,x/y);
    printf("x%%y=%d\n",x%y);
    return 0 ;
 }
```

运行结果：

```
x+y=10,x-y=4
x*y=21,x/y=2
x%y=1
```

注意：由于“％”是格式符的标志，所以在输出该符号时，必须利用“％％”。

3.5.2 自增运算符和自减运算符

自增和自减运算符是两个特殊的算术运算符，是单目运算，其优先级高于所有双目运算符。

自增运算符记为“＋＋”，其功能是使变量的值自增1。

自减运算符记为“－－”，其功能是使变量值自减1。

自增和自减运算符均为单目运算符，只有一个运算对象，且该运算对象只能是变量；具有右结合性。自增和自减运算符均可前置或后置，共有以下4种形式：

```
++i;                                    //前缀模式,i自增1后再参与其他运算
```

```
--i;                                    //前缀模式,i自减1后再参与其他运算
i++;                                    //后缀模式,i参与运算后,i的值再自增1
i--;                                    //后缀模式,i参与运算后,i的值再自减1
```


前缀模式和后缀模式的区别在于值的增加或者减少这一动作发生的准确时间是不同的。

【例3.8】 写出下列程序的结果。

程序如下:

```
#include<stdio.h>
int main()
{
    int i = 10,j = 8,m = 11,n = 20,k;
    printf("i = %3d\tj = %3d\t",i++,++j);
    printf("i = %3d\tj = %3d\t",i,j);
    k = m++;
    printf("k = %3d\t",k);
    printf("m = %3d\n",m);
    k = 3 * (++n);
    printf("k = %3d\t",k);
    printf("n = %3d\n",n);
    return 0;
 }
```

运行结果:

```
i = 10  j =  9  i = 11  j =  9  k = 11  m = 12
k = 63  n = 21
```

程序分析: 一要搞清变量除了参加自增或自减运算外,还进行了其他什么运算;二要注意前置的首先自增或自减,再参与其他运算;后置的先参与其他运算,再自增或自减。

如果企图一次使用太多的自增、自减运算符,可能连自己都会糊涂。自增和自减运算符常用于循环语句中,使循环变量自动加1。也用于以后要学习的指针变量,使指针指向下一个元素的地址,详见第9章。

3.5.3 算术表达式

表达式是由常量、变量、函数和运算符组合起来的式子。一个表达式有一个值及其类型,它们等于计算表达式所得结果的值和类型。表达式求值按运算符的优先级和结合性规定的顺序进行。单个的常量、变量、函数可以看作是表达式的特例。

由算术运算符连接起来的式子称为算术表达式,以下是算术表达式的例子。

```
a+b
(a*2)/c
(x+r)*8-(a+b)/7
++i
sin(x)+sin(y)
(++i)-(j++)+(k--)
```

在计算每个表达式的时候,一定要区分表达式的值和表达式中变量的值是两个不同的概念。

3.5.4 运算符的优先级和结合性

1. 运算符的优先级

C 语言中,运算符的运算优先级共分为 15 级。1 级最高,15 级最低。算术运算符的优先级是先乘除后加减,乘除和求模(%)的优先级一样。

在表达式中,优先级较高的先于优先级较低的进行运算。当两个运算符共享一个操作数时,优先级规定了求值的顺序。例如,在表达式 20-12/3 里,/优先级高于-,因此对于 12 而言,根据优先级的规定除法先进行。

2. 运算符的结合性

当两个共享一个操作数的运算符优先级相同时,结合性决定了求值的顺序。例如,在表达式 20-12/3 * 2 里,/与 * 优先级相同,对于 3 而言是根据算术运算符的左结合性由左至右进行求值。

C 语言中结合性分为两种,即左结合性(自左至右)和右结合性(自右至左)。

例如,算术运算符的结合性是自左至右,即先左后右。如有表达式 x-y+z 则 y 应先与"-"号结合,执行 x-y 运算,然后再执行+z 的运算。这种自左至右的结合方向就称为"左结合性"。而自右至左的结合方向称为"右结合性"。最典型的右结合性运算符是赋值运算符。如 x=y=z,由于"="的右结合性,应先执行 y=z,再执行 x=(y=z)运算。

在以后的学习中可以观察到只有部分单目运算符(第 2 级)、三目运算符和赋值运算符具有"右结合性"。

由于有关优先级和结合性的规则随着语言的不同而有所变化,所以对于要使用多种语言工作的程序员来讲,最明智的做法就是尽量使用括号来明确计算的顺序。

表 3.6 总结了所有运算符的优先级与结合性的规则。同一行的各个运算符具有相同的优先级,纵向往下优先级越低。例如 * 、/和%具有相同的优先级,比二元的+与-的优先级高。还有很多运算符没有介绍,将在后面的章节中结合实际应用进行学习。

表 3.6 运算符优先级与结合性

运 算 符	优 先 级	结 合 性
() [] -> .	1	从左至右
! - ++ -- + - &.(类型)sizeof	2	从右至左
* / %	3	从左至右
+ -	4	从左至右
<< >>	5	从左至右
< <= > >=	6	从左至右
== !=	7	从左至右
&.	8	从左至右
^	9	从左至右
|	10	从左至右
&.&.	11	从左至右
||	12	从左至右
?:	13	从右至左
= += -= *= /= %= &.= ^= |= <<= >>=	14	从右至左
,	15	从左至右

注:一元+ -优先级比二元+ -优先级高。

3.6 本章小结

在C语言中,数据类型可以分为基本类型、构造类型、指针类型和空类型4大类。在数据的操作上,C具有丰富的运算符号来进行处理。本章介绍了基本类型、数据类型转换、算术运算符与表达式。

1. 基本类型

本章介绍了基本类型中的整型、实型和字符型。对于基本类型数据,按其是否可改变又分为常量和变量两种。它们可以和数据类型结合起来分类。如可分为整型常量、整型变量、实型常量、实型变量、字符常量、字符变量等。求字节运算符(sizeof)用于计算数据类型所占的字节数。

2. 数据类型转换

变量的数据类型转换方式有两种:自动转换和强制转换。

3. 算术的运算符和表达式

(1) 运算符的优先级和结合性。

(2) 算术运算符。用于各类数值运算,包括+、-、*、/、%、自增++、自减--等。

通过本章的学习,要理解数据类型的意义,掌握常用的基本类型及其数据类型转换,掌握常见的运算符及其优先级。