第2章 数据类型与运算符

学习目标

- ◆ 掌握进制及进制转换
- ◆ 掌握 C 语言中的关键字和标识符
- ◆ 掌握不同数据类型间的转换
- ◆ 掌握运算符的运用

通过上一章的学习，相信大家对 C 语言已经有了一个初步认知。接下来的第 2 章将针对 C 语言开发中必须要掌握的进制、常量、变量、运算符等基础知识进行讲解。

2.1 进制

进制是一种计数机制，它可以使用有限的数字符号代表所有的数值。对于任何一种进制——X 进制，就表示某一位置上的数运算时逢 X 进一位。实际生活中也有很多进制的应用场景，例如时间每过 60 秒，分钟就会加 1，这就是六十进制。再比如对学生进行分组时，假设 8 人一组，可以让学生进行报数，报满 8 个数就多了一个小组，这就是八进制。在 C 语言程序中常用的有二进制、八进制、十进制和十六进制，本节将针对这些进制及进制间的转换进行详细地讲解。

2.1.1 什么是二进制

在绝大多数计算机系统中，数据都是通过二进制的形式存在的。二进制是一种“逢二进一”的机制，它用 0 和 1 两个符号来描述。为了帮助大家更好地理解二进制，接下来通过二进制和十进制的对比来描述二进制的表示方式，如表 2-1 所示。

表2-1　十进制与二进制对照表

十进制	二进制	十进制	二进制
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

从表 2-1 可以看出，当用二进制表示十进制数字 2 时，由于二进制的数码只有 0 和 1，所以根据“逢二进一”的规则，需要向高位进一位，表示为 0010。同理，使用二进制表示十进制数字 4 时，继续向高位进一位，表示为 0100。

需要注意的是，十进制和二进制只是针对数字的不同表示方式，无论采用哪种方式表示一个数，它的

值都是一样的。以十进制数 7 为例，十进制由符号“7”组成 ，而二进制由符号 “0”、“1”、“1”和“1”组成。

2.1.2 八进制与十六进制

在 C 语言中，除了上面讲到的二进制，八进制和十六进制在程序中也非常重要，下面分别进行介绍。

1、八进制

八进制是一种“逢八进一”的进制，它由 0~7 八个符号来描述。同样地，此处通过十进制和八进制的对比来描述八进制的表示方式，如表 2-2 所示。

表2-2 十进制数和八进制数

十进制	八进制	十进制	八进制
0	0	9	11
1	1	10	12
2	2	11	13
3	3	12	14
4	4	13	15
5	5	14	16
6	6	15	17
7	7	16	20
8	10	17	21

从表 2-2 中可以看出，当使用八进制表示十进制数字 8 时，由于表示八进制的符号只有 0~7，因此，根据逢八进一的规则，需要向高位进一位，表示为 10。同理，使用八进制表示十进制数字 16 时，继续向高位进一位，表示为 20。

2、十六进制

十六进制是一种“逢十六进一”的进制，它由 0~9、A~F 十六个符号来描述。下面通过十进制和十六进制的对比来描述十六进制的表示方式，如表 2-3 所示。

表2-3 十进制数和十六进制数

十进制	十六进制	十进制	十六进制
0	0	17	11
1	1	18	12
2	2	19	13
3	3	20	14
4	4	21	15
5	5	22	16
6	6	23	17
7	7	24	18
8	8	25	19
9	9	26	1A
10	A	27	1B
11	B	28	1C
12	C	29	1D

13	D	30	1E
14	E	31	1F
15	F	32	20
16	10	33	21

从表 2-3 中可以看出，当使用十六进制表示十进制数字 16 时，由于表示十六进制的符号只有 0~9、A~F，因此，根据逢“逢十六进一”的规则，需要向高位进一位，表示为 10。同理，使用十六进制表示十进制数字 32 时，继续向高位进一位，表示为 20。

2.1.3 进制转换

通过前面内容的学习，读者应该知道在计算机中一个数值可以用不同的进制形式来表示，但实际上，不管是哪种进制形式来表示，数值本身是不会发生变化的。因此，各种进制之间可以轻松地实现转换，下面就以前面提到的十进制、二进制、八进制、十六进制为例来讲解进制如何实现转换。

一、十进制与二进制之间的转换

十进制与二进制之间的转换是最常见也是必须掌握的一种进制转换方式，下列针对十进制转二进制和二进制转十进制的方式分别进行讲解。

1、十进制转二进制

十进制转换成二进制可以采用除 2 取余的方式。也就是说将要转换的数，先除以 2，得到商和余数，将商继续除以 2，获得商和余数，此过程一直重复直到商为 0。最后将所有得到的余数倒序排列，即可得到转换结果。

接下来就以十进制的 6 转换为二进制为例进行说明，其演算过程如图 2-1 所示。

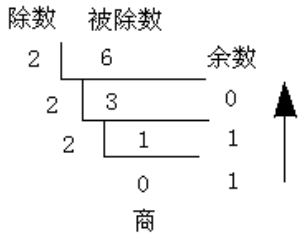

图2-1　十进制转二进制

从图 2-1 中可以看出，十进制的 6 连续三次除以 2 后，得到的余数依次是：0、1、1。将所有余数倒序排列后为 110，因此，十进制的 6 转换成二进制后的结果是 110。

2、二进制转十进制

二进制转化成十进制要从右到左用二进制位上的每个数去乘以 2 的相应次方，例如，将最右边第一位的数乘以 2 的 0 次方，第二位的数乘以 2 的 1 次方，第 n 位的数乘以 2 的 n-1 次方，然后把所有乘的结果相加，得到的结果就是转换后的十进制。

例如，把一个二进制数 0110 0100 转换为 10 进制，转换方式如下：

$$0 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3 + 0 * 2^4 + 1 * 2^5 + 1 * 2^6 + 0 * 2^7 = 100$$

由于 0 乘以多少都是 0，所以上述表达式也可以简写为：

$$1 * 2^2 + 1 * 2^5 + 1 * 2^6 = 100$$

得到的结果 100 就是二进制数 0110 0100 转化后的十进制表现形式。

二、八进制与二进制之间的转换

八进制与二进制之间的转换比较常见的操作就是将一个二进制数转为八进制。在转换的过程中有一个技巧，就是将二进制数自右向左每三位分成一段（若不足三位，用 0 补齐），然后将二进制每段的三位转为八进制的一位，转换过程中数值的对应关系如表 2-4 所示。

表2-4　二进制和八进制数值对应表

二进制	八进制
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

接下来，就以将二进制数 00101010 为例来演示如何转为八进制，具体演算过程如下：

（1） 每三位分成一段，结果为：000 101 010

（2）将每段的数值分别查表替换，结果如下：

```
010 → 2
101 → 5
000 → 0
```

（3）将替换的结果进行组合，组合后的八进制为 0052(注意八进制必须以 0 开头)。

三、十六进制与二进制之间的转换

将二进制转十六进制时，与转八进制类似，不同的是要将二进制数每四位分成一段（若不足 4 位用 0 补齐），查表转换即可。二进制转十六进制过程中数值的对应关系如表 2-5 所示。

表2-5　二进制和十六进制数值对应表

二进制	十六进制	二进制	十六进制
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

接下来，二进制数 01010110 转为十六进制，具体步骤如下：

（1）每四位分成一段,结果为：0101 0110

（2）将每段的数值分别查表替换，结果如下：

```
0110 → 6
```

```
0101 → 5
```

（3）将替换的结果进行组合，转换的结果为:0x56 或 0X56(注意十六进制必须以 0x 或者 0X 开头)。

上述讲解了二进制与其他进制的转换，除二进制外，其他进制之间的转换也很简单，只需将它们转换成二进制数，然后将二进制转为其他进制即可。

2.1.4 ASCII 码表

计算机使用特定的整数编码来表示对应的字符。我们通常使用的英文字符编码是 ASCII（American Standard Code for Information Interchange 美国信息交换标准编码）。ASCII 编码是一个标准，其内容规定了把英文字母、数字、标点、字符转换成计算机能识别的二进制数的规则，并且得到了广泛认可和遵守。下表为 ASCII 码表的可打印字符部分（0～127），供大家查阅使用，如表 2-6 所示。

表2-6　ASCII 码表

代码	字符	代码	字符	代码	字符	代码	字符
0		32	[空格]	64	@	96	`
1		33	!	65	A	97	a
2		34	"	66	B	98	b
3		35	#	67	C	99	c
4		36	$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	'	71	G	103	g
8	退格	40	(	72	H	104	h
9	Tab	41	)	73	I	105	i
10	换行	42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	l
13	回车	45	-	77	M	109	m
14		46	.	78	N	110	n
15		47	/	79	O	111	o
16		48	0	80	P	112	p
17		49	1	81	Q	113	q
18		50	2	82	R	114	r
19		51	3	83	S	115	s
20		52	4	84	T	116	t
21		53	5	85	U	117	u
22		54	6	86	V	118	v
23		55	7	87	W	119	w
24		56	8	88	X	120	x
25		57	9	89	Y	121	y
26		58	:	90	Z	122	z
27		59	;	91	[	123	{

28		60	<	92	\	124	|
29		61	=	93	]	125	}
30	-	62	>	94	^	126	~
31		63	?	95	_	127	

ASCII 码大致由以下两部分组成。

1、ASCII 非打印控制字符：ASCII 表上的数字 0-31 分配给了控制字符，用于控制像打印机等一些外围设备。（参详 ASCII 码表中 0-31）

2、ASCII 打印字符：数字 32-126 分配给了能在键盘上找到的字符，当查看或打印文档时就会出现。数字 127 代表 DELETE 命令。（参详 ASCII 码表中 32-127）

2.2 关键字和标识符

2.2.1 关键字

所谓关键字是指在编程语言里事先定义好并赋予了特殊含义的单词，也称作保留字。关键字在程序中用于表示特殊含义，不能被随便用作变量名、函数名等，在 C 语言中，C89 标准中共定义了 32 个关键字，具体如下：

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

上面列举的关键字中，每个关键字都有特殊的作用，例如，int 关键字用于声明一个整型的变量，sizeof 关键字用于获取指定类型数据的长度，char 关键字用于声明一个字符类型的变量。在本书后面的章节中将逐步对这些关键字进行讲解，这里只需了解即可。

2.2.2 标识符

在编程过程中，经常需要定义一些符号来标记一些名称，如变量名、方法名、参数名、数组名等，这些符号被称为标识符。在 C 语言中标识符的命名需要遵循一些规范，具体如下：

- 标识符只能由字母、数字和下划线组成。
- 标识符不能以数字作为第一个字符。
- 标识符不能使用关键字。
- 标识符区分大小写字母，如 add、Add 和 ADD 是不同的标识符。
- 尽量做到“见名知意”，以增加程序的可读性，如用 age 表示年龄，用 length 表示长度等。
- 虽然 ANSIC 中没有规定标识符的长度，但建议标识符的长度不超过 8 个字符。

在上面的规范中，除了最后两条外，其他的命名规范都是必须要遵守的，否则程序就会出错。为了让读者对标识符的命名规范有更深刻地理解，接下来列举一些合法与不合法的标识符，具体如下：

下面是一些合法的标识符：

```
area
DATE
```

```
_name
lesson_1
```

下面是一些不合法的标识符：

```
3a
ab.c
long
abc#
```

2.3 常量与变量

2.3.1 常量

生活中有些事物需要用数值来表示，例如人民币、时间等。在程序中，同样也会出现一些数值，例如123、1.5、'a'等，这些值是不可变的，通常将它们称之为常量。在 C 语言中，常量包括整型常量、浮点数常量、字符常量等，下面将分别进行详细讲解。

1、整型常量

整型常量是整数类型的数据，又被称为整常数。整常数可用以下三种形式表示，具体如下：

- 十进制整数，如 123，-456，0；
- 八进制整数，如 0123，-011；
- 十六进制整数，如 0x123，-0x12。

需要注意的是，由于生活中普遍使用十进制的方式来表示数字，那么在程序中为了符合生活习惯，通常也使用十进制来表示数字，在没有特定标识的情况下，都可以认为是十进制。

2、 实型常量

实型常量也称为浮点数常量，也就是在数学中用到的小数，可以分为 float 单精度浮点数和 double 双精度浮点数两种类型。其中，单精度浮点数后面以 F 或 f 结尾，而双精度浮点数则以 D 或 d 结尾。当然，在使用浮点数时也可以在结尾处不加任何的后缀。浮点数常量还可以通过指数形式来表示，具体示例如下：

```
2e3f  3.6d  0f  3.84d  5.022e+23f
```

上面列出的浮点数常量中用到了 e 和 f，在后面的小节中会进行详细讲解，这里了解即可。

3、字符常量

字符常量用于表示一个字符，一个字符常量要用一对英文半角格式的单引号（' '）引起来，它可以是英文字母、数字、标点符号以及由转义序列来表示的特殊字符。具体示例如下：

```
'a'  '1'  '&'  '\r' '0x20'
```

上面的示例中，'0x20'表示一个空白字符，即在单引号之间只有一个表示空白的空格。之所以能这样表示是因为 C 语言采用的是 ASCII 字符集，空格字符在 ASCII 码表中对应的值为'0x20'。

2.3.2 变量

在程序运行期间，随时可能产生一些临时数据，应用程序会将这些数据保存在一些内存单元中，每个内存单元都用一个标识符来标识。这些内存单元我们称之为变量，定义的标识符就是变量名，内存单元中存储的数据就是变量的值。

接下来，通过一段代码来学习变量的定义，具体如下：

```
int x = 0,y;
y = x+3;
```

上面的代码中，第一行代码的作用是定义了两个变量 x 和 y，也就相当于分配了两块内存单元，在定义变量的同时为变量 x 分配了一个初始值 0，而变量 y 没有分配初始值，变量 x 和 y 在内存中的状态如图 2-2 所示。

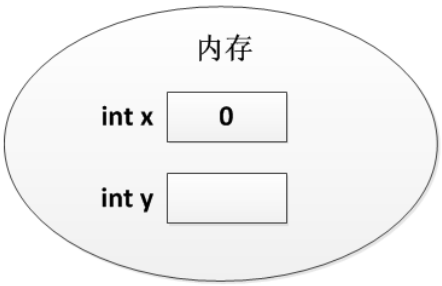

图2-2 x、y 变量在内存中的状态

第二行代码的作用是为变量赋值，在执行第二行代码时，程序首先取出变量 x 的值，与 3 相加后，将结果赋值给变量 y，此时变量 x 和 y 在内存中的状态发生了变化，如图 2-3 所示。

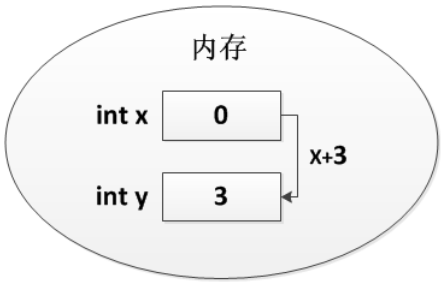

图2-3 x、y 变量在内存中的状态

从图 2-2、图 2-3 以及上面的描述不难发现，变量实际上就是一个临时存放数据的地方。在程序中，可以将指定的数据存放到变量中，方便随时取出来再次进行使用。变量对于一段程序的运行是至关重要的，读者在后续的学习中会逐步地了解变量的作用。

2.3.3 变量的数据类型

在应用程序中，由于数据存储时所需要的容量各不相同，因此，为了区分不同的数据，需要将数据划分为不同的数据类型。C 语言中的数据类型有很多种，具体如图 2-4 所示。

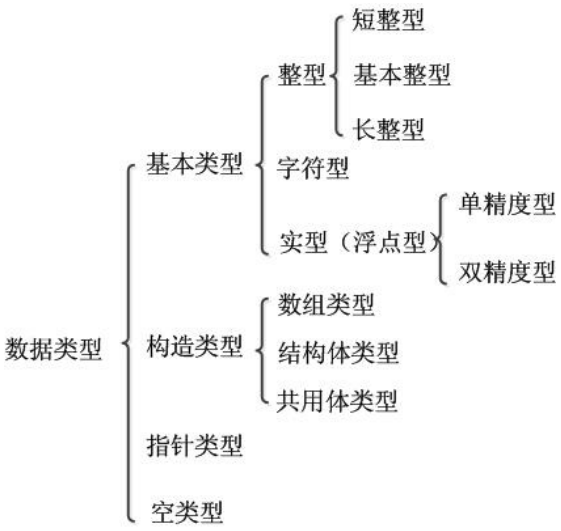

图2-4　C 语言的数据类型

从图 2-4 中可以看出，C 语言中的数据类型可分为 4 种，分别是基本类型、构造类型、指针类型、空类型。接下来将针对基本数据类型进行详细地讲解，关于其他数据类型将在后面章节中讲解。

1、整型变量

在程序开发中，经常会遇到 0、-100、1024 等数字，这些数字都可称为整型。整型就是一个不包含小数部分的数。在 C 语言中，根据数值的取值范围，可以将整型定义为短整型（short int）、基本整型（int）和长整型（long int）。表 2-7 列举了整数类型的长度及其取值范围。

表2-7　　整数类型的长度及其取值范围

修饰符	数据类型	占用空间	取值范围
[signed]	short [int]	16 位（2 个字节）	-32768 到 32767 ($-2^{15} \sim 2^{15}-1$)
	int	32 位（4 个字节）	-2147483648 到 2147483647 ($-2^{31} \sim 2^{31}-1$)
	long [int]	32 位（4 个字节）	-2147483648 到 2147483647 ($-2^{31} \sim 2^{31}-1$)
unsigned	short [int]	16 位（2 个字节）	0 到 65535 ($0 \sim 2^{16}-1$)
	int	32 位（4 个字节）	0 到 4294967295 ($0 \sim 2^{32}-1$)
	long [int]	32 位（4 个字节）	0 到 4294967295 ($0 \sim 2^{32}-1$)

从表 2-7 中可以看出，整数类型可分为 short、int 和 long，这三种类型可以被 signed 和 unsigned 修饰。其中，被 signed 修饰的整数类型称为有符号的整数类型，被 unsigned 修饰的称为无符号的整数类型。它们之间最大的区别是无符号类型可以存放的正数范围比有符号类型中的范围大一倍。例如，int 的取值范围是 -2^{31}~2^{31}-1，而 unsigned int 的取值范围是 0~2^{32}-1。默认情况下，整型数据都是有符号的，此时 signed 修饰符可以不用写。

需要注意的是，整型数据在内存中占的字节数与所选择的操作系统有关。虽然 C 语言标准中没有明确规定整型数据的长度，但 long 类型整数的长度不能短于 int 类型，short 类型整数的长度不能短于 int 类型。

通过对表 2-7 的学习，了解了不同整数类型数据的取值范围，接下来来看一个案例，具体代码如例 2-1 所示。

例2-1

```
1  #include <stdio.h>
2  void main()
3  {
4      int a = 12345;
5      long b = -23456 ,sum1;
```

```
6       unsigned int  c = 32800, sum2;
7       sum1 = b - a;
8       sum2 = c - b;
9       printf("sum1=%ld,sum2=%u\n",sum1,sum2);
10  }
```

运行结果如图 2-5 所示。

图2-5 运行结果

在图 2-5 中可以看出，有符号整数类型 sum1 的结果是-35801，无符号整数类型 sum2 的结果是 56256。

2、实型变量

实型变量也可以称为浮点型变量，浮点型变量是用来存储小数数值的。在 C 语言中，浮点型变量分为两种：单精度浮点数（float）、双精度浮点数（double），但是 double 型变量所表示的浮点数比 float 型变量更精确。表 2-8 列举了两种不同浮点型数所占用的存储空间大小及取值范围。

表2-8 浮点类型长度及其取值范围

类型名	占用空间	取值范围
float	32 位（4 个字节）	1.4E-45 ～ 3.4E+38，-1.4E-45 ～ -3.4E+38
double	64 位（8 个字节）	4.9E-324 ～ 1.7E+308, -4.9E-324 ～ -1.7E+308

表 2-8 中，列出了两种浮点数类型变量所占的空间大小和取值范围。在取值范围中，E 表示以 10 为底的指数，E 后面的“+”号和“-”号代表正指数和负指数，例如，1.4E-45 表示 $1.4*10^{-45}$。为了让读者更好地理解浮点型数据在内存中的存储方式，接下来以单精度浮点数为例进行详细讲解，如图 2-6 所示。

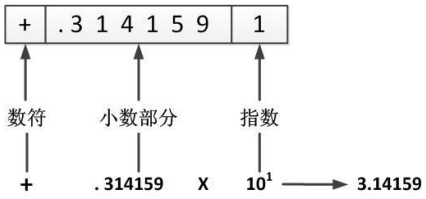

图2-6 单精度浮点数存储方式

在图 2-6 中，浮点数包含符号位、小数位和指数位三部分。例如，小数 3.14159 在内存中的符号位为“+”，小数部分为.31415，指数位为 1，连接在一起即为“$+0.314159 * 10^1 = 3.14159$”。

在 C 语言中，一个小数会被默认为 double 类型的值，因此在为一个 float 类型的变量赋值时需要注意一点，所赋值的后面一定要加上字母“F”（或者小写“f”），而为 double 类型的变量赋值时，其所赋值后面的字符“D”（或小写“d”），可以省略。具体示例如下：

```
float f = 123.4f;       //为一个 float 类型的变量赋值，后面必须加上字母 f
double d1 = 100.1;      //为一个 double 类型的变量赋值，后面可以省略字母 d
double d2 = 199.3d;     //为一个 double 类型的变量赋值，后面可以加上字母 d
```

另外，在程序中也可以为一个浮点数类型变量赋予一个整数数值，示例如下：

```
float f = 100;   //声明一个 float 类型的变量并赋整数值
double d = 100;  //声明一个 double 类型的变量并赋整数值
```

脚下留心：float 和 double 之间的数据转换

由于浮点型变量是由有限的存储单元组成的，因此只能提供有限的有效数字。在有效位以外的数字将被舍去，这样可能会产生一些误差，例如，将 3.1415926 赋给一个 float 型变量，但它只能保证前 7 位是有

效的，如例 2-2 所示。

例2-2

```
1  #include <stdio.h>
2  void  main()
3  {
4      float a ;
5      a = 3.141592612;
6      printf("a=%f\n",a);
7  }
```

运行结果如图 2-7 所示。

图2-7　运行结果

从图 2-7 中可以看出，程序运行结果为 3.141593。我们会发现该输出的值与给定的值之间有一些误差。这是由于 a 是单精度浮点型变量，它只能提供 7 位有效数字，而 3.141592612 已经超出了其取值范围，所以后面的几位被舍去了。

3、字符型变量

字符型变量用于存储一个单一字符，在 C 语言中用 char 表示，其中每个字符变量都会占用 1 个字节。在给字符型变量赋值时，需要用一对英文半角格式的单引号（' '）把字符括起来，例如，'A'的声明方式如下所示：

```
char ch = 'A';   //为一个 char 类型的变量赋值字符'a'
```

上述代码中，将字符常量'A'放到字符变量 ch 中，实际上并不是把该字符本身放到变量的内存单元中去，而是将该字符对应的 ASCII 编码放到变量的存储单元中。例如：ASCII 使用编号 65 来对应大写字母“A”，因此变量 ch 存储的是整数 65，而不是字母“A”本身。接下来通过一个案例来说明，如例 2-3 所示。

例2-3

```
1  #include <stdio.h>
2  void main()
3  {
4      char ch1 = 'A';
5      char ch2 = 65;
6      printf("%c\n", ch1);
7      printf("%c\n", ch2);
8  }
```

运行结果如图 2-8 所示。

图2-8　ASCII 字符

例 2-3 中，定义了两个 char 类型变量，分别赋值为字符'A'和数字 65，然后通过 printf 函数把两个变量的内容以字符形式打印到屏幕上。从图 2-8 中可以看出，两个变量输出的结果是一样的，这说明对于字符型来说，A 和 65 其实没什么区别。严格来说，字符类型也是整型类型。

需要注意的是，除了可以直接从键盘上输入的字符（如英文字母，标点符号，数字，数学运算符等）以外，还有一些字符是无法用键盘直接输入的，比如，“回车”，此时需要采用一种新的定义方式——转义字符，它以反斜杠\开头，随后接特定的字符。表 2-9 列举了一些常见的转义字符。

表2-9　部分常见转义字符表

转义字符	对应字符	ASCII 码表中的值
'\t'	制表符（Tab 键）	9
'\n'	换行	10
'\r'	回车	13
'\"'	双引号	34
'\''	单引号	39
'\\'	反斜杠	92

接下来，通过一个具体的案例来演示转义符的用法，如例 2-4 所示。

例2-4

```
1   #include <stdio.h>
2   void main()
3   {
4       char ch1 = 'A';
5       char ch2 = '\n';
6       char ch3 = 'B';
7       char ch4 = '\\';
8       printf("%c", ch1);
9       printf("%c", ch2);
10      printf("%c", ch3);
11      printf("%c", ch4);
12      printf("%c", ch2);
13  }
```

运行结果如图 2-9 所示。

图2-9　ASCII 字符的输出结果

在例 2-4 中定义了四个字符型变量，其中 ch2 被赋值为转义字符 '\n' 即换行符序，ch4 被赋值为转义字符'\\' 即反斜杠。从第 8 行到第 11 行按照顺序分别将四个变量的值输出到屏幕上，会发现输出字符 A 之后另起一行输出字符 B，转义字符'\n'的作用就是控制输出结果另起一行。字符 B 后输出的是字符 '\' 。第 12 行为了使输出结果的格式清晰一些又输出了一个换行符，防止程序结束后命令行提示符紧跟在输出结果的后面。

4、枚举类型变量

在日常生活中有许多对象的值是有限的，可以一一列举出来。例如一个星期内只有七天、一年只有十二个月等等。如果把这些量说明为整型，字符型或其它类型显然是不妥当的。为此，C 语言提供了一种称为“枚举”的类型。枚举类型就是其值可以被一一列举出来，并且变量的取值不能超过定义的范围。

枚举类型的声明方式比较特殊，具体格式如下：

```
enum 枚举名 {标识符 1 = 整型常量 1, 标识符 2 = 整型常量 2, ...};
```

在上述代码中，enum 表示声明枚举的关键字，枚举名表示枚举对象的名称。为了让读者更好的理解枚举类型的使用，接下来通过定义一个枚举类型来进行详细讲解。示例如下：

```
enum month { JAN=1, FEB=2, MAR=3, APR=4, MAY=5, JUN=6,
             JUL=7, AUG=8, SEP=9, OCT=10, NOV=11, DEC=12 };
```

上面的代码中，声明了一个枚举类型对象 month，然后就可以使用此类型来定义变量。接下来，通过一个具体案例来学习枚举的具体用法，如例 2-5 所示。

例2-5

```
1  #include <stdio.h>
2  enum month {JAN=1, FEB=2, MAR=3, APR=4, MAY=5, JUN=6,
3              JUL=7, AUG=8, SEP=9, OCT=10, NOV=11, DEC=12};
4  void main()
5  {
6      enum month lastmonth, thismonth, nextmonth;
7      lastmonth = APR;
8      thismonth = MAY;
9      nextmonth = JUN;
10     printf("%d %d %d \n", lastmonth, thismonth, nextmonth);
11 }
```

运行结果如图 2-10 所示。

图2-10 运行结果

在例 2-5 中，第 3 行代码定义了一个枚举类型，其中，枚举名 month 是一个标识符，大括号中的内容被称为枚举值表；枚举值表内的标识符 JAN、FEB、MAR 等被称作枚举元素，枚举元素是被命名的整型常量，枚举值表中罗列出所有可用值，枚举成员对应的值是枚举值。枚举类型的定义以分号结束。第 6 行定义了三个枚举变量，在后面的代码中分别进行了赋值，最后依次将这三个枚举变量的值输出到显示器。可以看出输出结果为枚举元素对应的值。

需要注意的是，枚举值是常量，不是变量，在程序中不能赋值。例如在主函数中，对 MAR 再次赋值是错误的。

多学一招：枚举变量的快速定义

在枚举中规定，如果不给标识符指定具体的值，会默认该标识符的值等于前一标识符的值加 1。因此可以将上面的定义简化成：

```
enum month{JAN=1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC};
```

如果不指定第一个标识符对应的常量，则它的默认值是 0，如例 2-6 所示：

例2-6

```
1  #include <stdio.h>
2  //定义一组常量
3  enum Constants {C1, C2, C3 = 4, C4, C5 = 3, C6, C7, C8 = '0', C9};
4  void main()
5  {
6      printf("C1=%d\n", C1);
```

```
7   }
```

运行结果如图 2-11 所示。

图2-11 运行结果

在例 2-6 中，首先用枚举定义了一组常量 C1 到 C9，并向显示器输出常量 C1 的数值。作为读者，在编译和运行该例程之前，最好先试着猜测 C1 到 C9 这九个常量的数值。

2.3.4 类型转换

在 C 语言程序中，经常需要对不同类型的数据进行运算，为了解决数据类型不一致的问题，需要对数据的类型进行转换。例如一个浮点数和一个整数相加，必须先将两个数转换成同一类型。C 语言程序中的类型转换可分为隐式和强制类型转换两种，具体如下。

1、隐式类型转换

所谓隐式类型转换指的是，系统自动将取值范围小的数据类型转换为数据取值范围大的数据类型，它是由系统自动转换完成的。例如，将 int 类型和 double 类型的数据相加，系统会将 int 类型的数据转换为 double 类型的数据，再进行相加操作，具体示例如下：

```
int num1=12;
double num2=10.5;
num1+num2;
```

在上述示例代码中，由于 double 类型的取值范围大于 int 类型，因此，将 int 类型的 num1 与 double 类型的 num2 相加时，系统会自动将 num1 的数据类型由 int 转换为 double 类型，从而保证数据的精度不会丢失。

2、强制类型转换

所谓强制类型转换指的是使用强制类型转换运算符，将一个变量或表达式转化成所需的类型，其基本语法格式如下所示：

```
（类型名）（表达式）
```

在上述格式中，类型名和表达式都需要用括号括起来，具体示例如下：

```
double x;
int y;
(double)(x+y)   //将表达式 x+y 的值转换成 double 类型
(int)x+y        //将变量 x 的值转换成 int 后,在与 y 相加
```

上述讲解的两种类型转换，看起来很简单，但在使用时有许多细节需要注意，具体如下：

（1）浮点型与整型

将浮点数(单双精度)转换为整数时，将舍弃浮点数的小数部分，只保留整数部分。将整型值赋给浮点型变量，数值不变，只将形式改为浮点形式，即小数点后带若干个 0。需要注意的是，赋值时的类型转换实际上是强制的。

（2）单、双精度浮点型

由于 C 语言中的浮点值总是用双精度表示的，所以 float 型数据参与运算时只需要在尾部加 0 延长为 double 型数据。double 型数据转换为 float 型时，会造成数据精度丢失，有效位以外的数据将会进行四舍五入。

（3）char 型与 int 型

将 int 型数值赋给 char 型变量时，只保留其最低 8 位，高位部分舍弃。

将 char 型数值赋给 int 型变量时， 一些编译程序不管其值大小都作正数处理，而另一些编译程序在转换时会根据 char 型数据值的大小进行判断，若值大于 127，就作为负数处理。对于使用者来讲，如果原来 char 型数据取正值，转换后仍为正值。如果原来 char 型值可正可负，则转换后也仍然保持原值，只是数据的内部表示形式有所不同。

（4）int 型与 long 型

long 型数据赋给 int 型变量时，将低 16 位值送给 int 型变量，而将高 16 位截断舍弃。(这里假定 int 型占两个字节)。 将 int 型数据送给 long 型变量时，其外部值保持不变，而内部形式有所改变。

（5）无符号整数

将一个 unsigned 型数据赋给一个长度相同的整型变量时(如：unsigned→int、unsigned long→long，unsigned short→short) ，内部的存储方式不变，但外部值却可能改变。

将一个非 unsigned 整型数据赋给一个长度相同的 unsigned 型变量时， 内部存储形式不变，但外部表示时总是无符号的。

2.4 运算符

运算符是编程语言中不可或缺的一部分，用于对一个或多个值(表达式)进行运算。本节将针对 C 语言中的常见运算符进行详细地讲解。

2.4.1 运算符与表达式

在应用程序中，经常会对数据进行运算，为此，C 语言提供了多种类型的运算符，即专门用于告诉程序执行特定运算或逻辑操作的符号。根据运算符的作用，可以将 C 语言中常见的运算符分为六大类，具体如表 2-10 所示。

表2-10　常见的运算符类型及其作用

运算符类型	作用
算术运算符	用于处理四则运算
赋值运算符	用于将表达式的值赋给变量
比较运算符	用于表达式的比较，并返回一个真值或假值
逻辑运算符	用于根据表达式的值返回真值或假值
位运算符	用于处理数据的位运算
sizeof 运算符	用于求字节数长度

表 2-10 列举了 C 语言中常用的运算符类型，并且每种类型运算符的作用都不同。运算符是用来操作数据的，因此，这些数据也被称为操作数，使用运算符将操作数连接而成的式子称为表达式。表达式具有如下特点：

1、 常量和变量都是表达式，例如，常量 3.14、变量 i。

2、 运算符的类型对应表达式的类型，例如，算术运算符对应算术表达式。

3、每一个表达式都有自己的值，即表达式都有运算结果。

2.4.2 算术运算符

在数学运算中最常见的就是加减乘除四则运算。C 语言中的算术运算符就是用来处理四则运算的符号，这是最简单、最常用的运算符号。表 2-11 列出了 C 语言中的算术运算符及其用法。

表2-11 算术运算符

运算符	运算	范例	结果
+	正号	+3	3
-	负号	b=4;-b;	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取模（即算术中的求余数）	7%5	2
++	自增（前）	a=2;b=++a;	a=3;b=3;
++	自增（后）	a=2;b=a++;	a=3;b=2;
--	自减（前）	a=2;b=--a;	a=1;b=1;
--	自减（后）	a=2;b=a--;	a=1;b=2;

算术运算符看上去都比较简单，也很容易理解，但在实际使用时还有很多需要注意的问题，接下来就针对其中比较重要的几点进行详细地讲解，具体如下：

1、进行四则混合运算时，运算顺序遵循数学中“先乘除后加减”的原则。

2、在进行自增（++）和自减（--）的运算时，如果运算符（++或--）放在操作数的前面则是先进行自增或自减运算，再进行其他运算。反之，如果运算符放在操作数的后面则是先进行其他运算再进行自增或自减运算。

请仔细阅读下面的代码块，思考运行的结果。

```
int num1 = 1;
int num2 = 2;
int res = num1 + num2++;
printf("num2=%d" + num2);
printf("res=%d" + res);
```

上面的代码块运行结果为：num2=3，res=3，具体分析如下：

第一步：运算 mum1+mun2++的结果，此时变量 mun1，mum2 的值不变。

第二步：将第一步的运算结果赋值给变量 res，此时 res 值为 3。

第三步：mum2 进行自增，此时其值为 3。

3、在进行除法运算时，当除数和被除数都为整数时，得到的结果也是一个整数。如果除法运算有浮点数参与运算，系统会将整数数据隐形类型转换为浮点类型，最终得到的结果会是一个浮点数。例如，2510/1000 属于整数之间相除，会忽略小数部分，得到的结果是 2，而 2.5/10 的实际结果为 0.25。

请思考一下下面表达式的结果：

```
3500/1000*1000
```

结果为 3000。由于表达式的执行顺序是从左到右，所以先执行除法运算 3500/1000，得到结果为 3，然后再乘以 1000，最终得到的结果自然就是 3000。

4、取模运算在程序设计中都有着广泛的应用，例如判断奇偶数的方法就是求一个数字除以 2 的余数是 1 还是 0。在进行取模运算时，运算结果的正负取决于被模数(%左边的数)的符号，与模数(%右边的数)的

符号无关。如：(-5)%3=-2，而 5%(-3)=2。

2.4.3 赋值运算符

赋值运算符的作用就是将常量、变量或表达式的值赋给某一个变量。表 2-12 列举了 C 语言中的赋值运算符及其用法。

表2-12　赋值运算符

运算符	运算	范例	结果
=	赋值	a=3;b=2;	a=3;b=2;
+=	加等于	a=3;b=2;a+=b;	a=5;b=2;
-=	减等于	a=3;b=2;a-=b;	a=1;b=2;
*=	乘等于	a=3;b=2;a*=b;	a=6;b=2;
/=	除等于	a=3;b=2;a/=b;	a=1;b=2;
%=	模等于	a=3;b=2;a%=b;	a=1;b=2;

在表 2-12 中，“=”的作用不是表示相等关系，而是赋值运算符，即将等号右侧的值赋给等号左侧的变量。在赋值运算符的使用中，需要注意以下几个问题：

1、在 C 语言中可以通过一条赋值语句对多个变量进行赋值，具体示例如下：

```
int  x, y, z;
x = y = z = 5;              //为三个变量同时赋值
```

在上述代码中，一条赋值语句可以同时为变量 x、y、z 赋值，这是由于赋值运算符的结合性为“从右向左”，即先将 5 赋值给变量 z，然后再把变量 z 的值赋值给变量 y，最后把变量 y 的值赋值变量 x，表达式赋值完成。需要注意的是，下面的这种写法在 C 语言中是不可取的。

```
int  x = y = z = 5;      //这样写是错误的
```

2、在表 2-12 中，除了“=”，其他的都是特殊的赋值运算符，接下来以“+=”为例，学习特殊赋值运算符的用法，示例代码如下：

```
int x=2;
x+=3;
```

上述代码中，执行代码 x += 3 后，x 的值为 5。这是因为在表达式 x+=3 中的执行过程为：

（1）将 x 的值和 3 的执行相加。

（2）将相加的结果赋值给变量 x。

所以，表达式 x=+3 就相当于 x = x + 3，先进行相加运算，在进行赋值。-=、*=、/=、%=赋值运算符都可依此类推。

多学一招：运算符的结合性

运算符的结合性指同一优先级的运算符在表达式中操作的结合方向,即当一个运算对象两侧运算符的优先级别相同时，运算对象与运算符的结合顺序。大多数运算符结合方向是“自左至右”。示代码如下：

```
a-b+c;
```

上述代码中表达式 a- b+c，b 两侧有-和+两种运算符的优先级相同，按先左后右的结合方向，b 先与减号结合，执行 a- b 的运算，然后再执行加 c 的运算。除了自左至右的结合性外，C 语言还有三类运算符，它们分别是单目运算符、条件运算符和赋值运算符。以赋值运算符为例，具体代码如下所示：

```
a=b+c;
```

2.4.4 比较运算符

比较运算符用于对两个数值或变量进行比较，其结果是一个逻辑值（“真”或“假”），如“5>3”，其值为“真”。

C 语言的比较运算中，“真”用数字“1”来表示，“假”用数字“0”来表示。表 2-13 列出了 C 中的比较运算符及其用法。

表2-13 比较运算符

运算符	运算	范例	结果
==	相等于	4 == 3	0
!=	不等于	4 != 3	1
<	小于	4 < 3	0
>	大于	4 > 3	1
<=	小于等于	4 <= 3	0
>=	大于等于	4 >= 3	1

需要注意的是，在使用比较运算符时，不能将比较运算符“==”误写成赋值运算符“=”。

2.4.5 逻辑运算符

逻辑运算符用于判断数据的真假，其结果仍为“真”或“假”。表 2-14 列举了 C 语言中的逻辑运算符及其范例。

表2-14 逻辑运算符

运算符	运算	范例	结果
!	非	!a	如果 a 为假，则!a 为真； 如果 a 为真，则!a 为假
&&	与	a&&b	如果 a 和 b 都为真，则结果为真否则为假
||	或	a|| b	如果 a 和 b 有一个或以上为真，则结果为真，二者都为假时，结果为假

当使用逻辑运算符时，有一些细节需要注意，具体如下：

1、逻辑表达式中可以包含多个逻辑运算符，例如，!a||a>b

2、三种逻辑运算符的优先级从高到低依次为：!、&&、||

3、运算符“&&”表示与操作，当且仅当运算符两边的表达式结果都为真时，其结果才为真，否则结果为假。如果左边为假，那么右边表达式是不会进行运算的，具体示例如下：

```
a+b<c&&c==d
```

若 a=5，b=4，c=3，d=3，由于 a+b 的结果大于 c，表达式 a+b<c 的结果为假，因此，右边表达式 c==d 不会进行运算，表达式 a+b<c&&c==d 的结果为假。

4、运算符“||”表示或操作，当且仅当运算符两边的表达式结果都为假时，其结果为假。同“&&”运算符类似，如果运算符“||”左边操作数的结果为真，右边表达式是不会进行运算的，具体示例如下：

```
a+b<c||c==d
```

若 a=1，b=2，c=4，d=5，由于 a+b 的结果小于 c，表达式 a+b<c 的结果为真，因此，右边表达式 c==d 不会进行运算，表达式 a+b<c||c==d 的结果为真。

2.4.6 位运算符

位运算符是针对二进制数的每一位进行运算的符号，它是专门针对数字 0 和 1 进行操作的。C 语言中的位运算符及其范例如表 2-1515 所示。

表2-15　位运算符

运算符	运算	范例	结果
&	按位与	0 & 0	0
		0 & 1	0
		1 & 1	1
		1 & 0	0
|	按位或	0 | 0	0
		0 | 1	1
		1 | 1	1
		1 | 0	1
~	取反	~0	1
		~1	0
^	按位异或	0 ^ 0	0
		0 ^ 1	1
		1 ^ 1	0
		1 ^ 0	1
<<	左移	00000010<<2	00001000
		10010011<<2	01001100
>>	右移	01100010>>2	00011000
		11100010>>2	11111000

接下来通过一些具体示例，对表 2-15 中描述的位运算符进行详细介绍，为了方便描述，下面的运算都是针对 byte 类型的数，也就是 1 个字节大小的数。

1、与运算符“&”是将参与运算的两个二进制数进行“与”运算，如果两个二进制位都为 1，则该位的运算结果为 1，否则为 0。

例如将 6 和 11 进行与运算， 6 对应的二进制数为 00000110，11 对应的二进制数为 00001011，具体演算过程如下所示：

```
    00000110
&
    00001011
————————————
    00000010
```

运算结果为 00000010，对应数值 2。

2、位运算符“|”是将参与运算的两个二进制数进行“或”运算，如果二进制位上有一个值为 1，则该位的运行结果为 1，否则为 0。具体示例如下：

例如将 6 与 11 进行或运算，具体演算过程如下：

```
    00000110
|
    00001011
————————————
    00001111
```

运算结果为 00001111，对应数值 15。

3、 位运算符“~”只针对一个操作数进行操作，如果二进制位是 0，则取反值为 1；如果是 1，则取反值为 0。

例如将 6 进行取反运算，具体演算过程如下：

```
   ~    00000110
  ————————————————
       11111001
```

运算结果为 11111001，对应数值-7。

4、 位运算符“^”是将参与运算的两个二进制数进行“异或”运算，如果二进制位相同，则值为 0，否则为 1。

例如将 6 与 11 进行异或运算，具体演算过程如下：

```
       00000110
   ^
       00001011
  ————————————————
       00001101
```

运算结果为 00001101，对应数值 13。

5、 位运算符“<<”就是将操作数所有二进制位向左移动一位。运算时，右边的空位补 0。左边移走的部分舍去。

例如一个 byte 类型的数字 11 用二进制表示为 00001011，将它左移一位，具体演算过程如下：

```
       00001011        <<1
  ————————————————
       00010110
```

运算结果为 00010110，对应数值 22。

6、 位运算符“>>”就是将操作数所有二进制位向右移动一位。运算时，左边的空位根据原数的符号位补 0 或者 1（原来是负数就补 1，是正数就补 0）。

例如一个 byte 的数字 11 用二进制表示为 00001011，将它右移一位，具体演算过程如下：

```
       00001011        >>1
  ————————————————
       00000101
```

运算结果为 00000101，对应数值 5。

2.4.7 sizeof 运算符

同一种数据类型在不同的编译系统中所占空间不一定相同，例如，在基于 16 位的编译系统中，int 类型占用 2 个字节，而在 32 位的编译系统中，int 类型占用 4 个字节。为了获取某一数据或数据类型在内存中所占的字节数，C 语言提供了 sizeof 运算符，使用 sizeof 运算符获取数据字节数，其基本语法规则如下所示：

```
sizeof(数据类型名称);
或
sizeof(变量名称);
```

为了帮助大家更好地学习 sizeof 运算符，接下来，通过一个案例来计算每种数据类型所占内存的大小，具体如例 2-7 所示。

例2-7

```
#include <stdio.h>
void main()
{
    //通过类型名称计算各基本数据类型所占内存大小
    printf("char:   %d\n", sizeof(char));
    printf("short:  %d\n", sizeof(short));
    printf("long:   %d\n", sizeof(long));
    printf("float:  %d\n", sizeof(float));
    printf("double: %d\n", sizeof(double));
    printf("unsigned char:  %d\n", sizeof(unsigned char));
    printf("unsigned short: %d\n", sizeof(unsigned short));
    printf("unsigned int:   %d\n", sizeof(unsigned int));
    printf("unsigned long:  %d\n", sizeof(unsigned long));
    //通过变量名称计算变量所属数据类型占用内存大小
    int val_int = 100;
    double val_double = 100000.0;
    printf("val_int:    %d\n", sizeof(val_int));
    printf("val_double: %d\n", sizeof(val_double));
}
```

运行结果如图 2-12 所示。

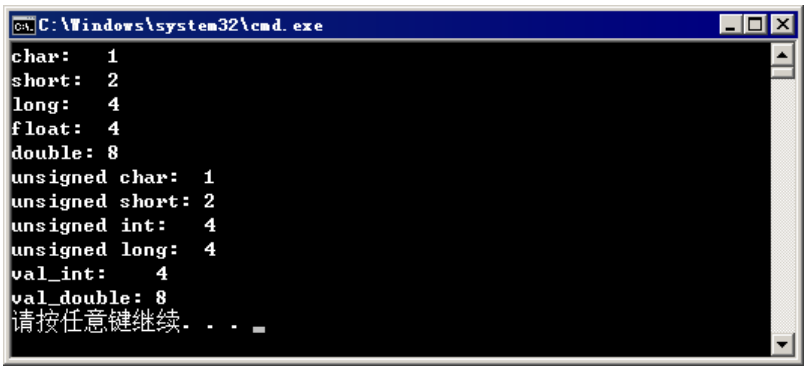

图2-12 运行结果

从图 2-12 中可以看出，不同数据类型在内存中所占字节数都打印了出来，由此可见，使用 sizeof 关键字可以很方便地获取到数据或数据类型在内存中所占的字节数。

2.4.8 运算符的优先级

在对一些比较复杂的表达式进行运算时，要明确表达式中所有运算符参与运算的先后顺序，我们把这种顺序称作运算符的优先级。表 2-16 列出了 C 语言中运算符的优先级，数字越小优先级越高。

表2-16 运算符优先级

优先级	运算符
1	. [] ()
2	++ -- ~ ! (数据类型)
3	* / %

4	+ -
5	<< >> >>>
6	< > <= >=
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:（三目运算符）
14	= *= /= %= += -= <<= >>= >>>= &= ^= |=

根据表 2-16 所示的运算符优先级，分析下面代码的运行结果。

```
int a =2;
int b = a + 3*a;
printf ("%d",b);
```

以上代码的运行结果为 8，这是由于运算符“*”的优先级高于运算符“+”，因此先运算 3*a，得到的结果是 6，再将 6 与 a 相加，得到最后的结果 8。

```
int a =2;
int b = (a+3) * a;
printf ("%d",b);
```

以上代码的运行结果为 10，这是由于运算符“()”的优先级最高，因此先运算括号内的 a+3，得到的结果是 5，再将 5 与 a 相乘，得到最后的结果 10。

其实没有必要去刻意记忆运算符的优先级。编写程序时，尽量使用括号“()”来实现想要的运算顺序，以免产生歧义。

多学一招：单目、双目、三目运算符

在 C 语言中根据运算符进行运算需根据变量的个数可以将运算符分为单目运算符、双目运算符和三目运算符，具体如表 2-17 所示。

表2-17 单目、双目、三目运算符的比较

名称	运算所需变量个数	范例
单目运算符	1 个	++、--、!、sizeof、~、…
双目运算符	2 个	+、-、*、/、%、<<、==、…
三目运算符	3 个	?:

在表 2-17 中单目运算符的优先级高于双目运算符和三目运算符。三目运算符“?:”用于条件判断的情况，读者在这只需了解即可，在后面的章节中将会进行详细讲解。

2.5 本章小结

本章主要讲解了 C 语言中的数据类型以及运算符。其中包括进制、基本数据类型、类型转换、运算符与表达式等。通过本章的学习，读者可以掌握 C 语言中数据类型及其运算的一些相关知识。熟练掌握本章的内容，可以为后面的学习打下坚实的基础。

2.6 习题

一、填空题

1、八进制必须以_____开头，十六进制必须以_____开头。

2、标识符只能由字母、数字和_____组成。

3、在计算机中的二进制表现形式有三种，分别是_____、_____、_____。

4、C 语言中的数据类型可分为 4 种，分别是基本类型、_____、指针类型、_____。

5、C 语言提供了 sizeof 运算符，该运算符主要用于_____。

二、判断题

1、执行语句++i; i=3; 后变量 i 的值为 4。

2、隐式类型转换是指将取值范围大的数据类型转换为数据取值范围小的数据类型。

3、C 语言中的逻辑值“真”是用 1 表示的，逻辑值“假”是用 0 表示的。

4、C 语言中赋值运算符比关系运算符的优先级高。

5、位运算符是专门针对数字 0 和 1 进行操作的。

三、选择题

1、下列选项中，可以正确定义 C 语言整型常量是?

A、32L　　B、51000　　C、-1.00　　D、567

2、算术运算符、赋值运算符和关系运算符的运算优先级按从高到低依次为?

A、算术运算、赋值运算、关系运算　　B、算术运算、关系运算、赋值运算

C、关系运算、赋值运算、算术运算　　D、关系运算、算术运算、赋值运算

3、设整型变量 m,n,a,b,c,d 均为 1，执行 (m=a>b)&&(n=c>d)后, m,n 的值是?

A、0,0　　B、0,1　　C、1,0　　D、1,1

4、若已定义 x 和 y 为 double 类型，则表达式 x=1;y=x+3/2 的值是?

A、1　　B、2　　C、2.0　　D、2.5

5、假设 a=1,b=2,c=3,d=4 则表达式 a<b? a : c<d? a : d 的结果为?

A、4　　B、3　　C、2　　D、1

四、简答题

1、请列举几个单目运算符、双目运算符、三目运算符并说明其区别?

2、请简述自增运算符放在变量前面和后面的区别?

五、编程题

1、已知梯形的上底为 a，下底为 b，高为 h，请用程序实现求梯形的面积。

2、请使用位运算符实现交换两个变量值的功能。