第 3 章 数据类型、运算符和表达式

计算机中所谓的类型，就是对数据分配存储单元的安排，它包括存储单元的长度（占多少字节）及数据的存储形式，不同的类型分配不同的长度和存储形式。

C 语言中数据是有类型的，不同类型的数据具有不同类型的运算性质，其在计算机中的存储也是不同的。本章将详细介绍 C 语言的数据类型及常量与变量的表示方法，各种运算符的功能及其运算规则，表达式和语句的书写方法等基本知识。

3.1 常量与变量

3.1.1 常量

所谓常量，是指在程序运行过程中，其值不能被改变的量。

常量也是数据，所以常量也应该有数据类型。C 语言中常量的类型有 4 种：

（1）整型常量，如 1000、12345、0、−345。

（2）实型常量。

- 十进制小数形式表示，如 0.34、−56.79、0.0。
- 指数形式表示，如 18.34e3（代表 18.34×10^3）。

（3）字符常量。

- 一般的字符常量，如'A'、's'、'8'。
- 转义字符，如'\n'。

（4）字符串常量，如"Boy"、"shaanxi"。

常量分为直接常量和符号常量。从其字面形式即可判断出的常量称为字面常量或直接常量。用一个标识符来代表一个常量，则称为符号常量，如#define、PI、3.1416。

例 3.1 符号常量的定义和使用。

```
#include <stdio.h>
#define PRICE 5
void main( )
{
  int num=20, total;
```

```
    total=num*PRICE;
    printf("total= %d", total);
}
```

运行结果如图 3-1 所示。

图 3-1　例 3.1 运行结果图

在上面这个程序中，用 PRICE 这个标识符代表常量 5。用#define 命令使 PRICE 这个标识符在随后的程序中代表数字 5。定义后的 PRICE 则称为符号常量，即标识符形式的常量，它的值在程序运行期间不能改变。

习惯上，符号常量名用大写字母表示，变量用小写字母表示，以示区别。使用符号常量可以提高程序的可读性，便于修改，具有以下优点。

（1）含义清楚，便于理解。如上面的程序中，看程序时，从 PRICE 就知道它代表价格。因此，定义符号常量名应考虑“见名知义”。

（2）修改方便，一改全改。在需要改变一个常量的值时，能做到“一改全改”。例如，对上面的程序作如下修改：

#define PRICE 10

则在程序中出现的所有 PRICE 都代表 10。

注意：符号常量是常量，不同于变量，它的值在其作用域内不能改变。如再用下面的赋值语句对 PRICE 赋值是不对的。

PRICE=30;

3.1.2　变量

在程序运行过程中，其值可以改变的量，称为变量。一个变量应该有一个名字作为标识，变量名的命名必须遵循标识符规则。在命名时应考虑“见名知义”的原则，如用“sum”代表“总和”。

实际上变量在存在期间，在内存中占据一定的存储单元，在该存储单元中存放变量的值。变量名实际上是一个符号地址，在对程序编译连接时由系统给每一个变量分配一个内存地址，而在程序运行过程中从变量中取值，则实际上是通过变量名找到相应的内存地址，从其存储单元中读取数据的。

变量也分为不同的类型，如整型变量、实型变量、字符型变量等。在 C 语言中要求对所有用到的变量作强制定义，即应“先定义，后使用”。否则，在编译时会指出有关“出错信息”，这样做的目的是：

（1）保证程序中变量的使用不会发生错误，例如，在定义部分有：

int　student;

而在执行语句中错写成了：

statent=30;

在编译时会检查出 statent 没有进行定义，可以避免使用变量名时出错。

（2）指定变量类型后，在编译时就能为其分配相应的存储单元。

（3）一个变量类型确定后，也就确定了该变量所能进行的操作。例如，整型变量 a 和 b，可以进行求余运算：a%b。%表示“求余”，得到 a/b 的余数。如果将 a、b 指定为实型变量，则不允许进行求余运算。在编译时，可以根据其类型来检查该变量所进行的运算是否合法。

（4）变量定义的一般形式为：

类型符　变量名表列;

这里的类型符必须是有效的 C 数据类型，变量名表列可以由一个或多个由逗号分隔的多个标识符构成。例如：

```
int  a, b;
```

表示定义了两个整型变量 a 和 b。

变量的值一般通过赋值运算符或通过调用标准输入函数获取。变量赋值的一般形式为：

变量名=表达式;

例如：

```
int  a;
a=8;
```

则是将整型数据 8 赋给整型变量 a，从而使变量 a 对应的存储单元中存放的数据为 8。

也可以在定义变量时给变量赋值，称为变量的初始化。其一般形式为：

类型符　　变量名=表达式;

例如：

```
int a=3;                  /*指定 a 为整型变量，初值为 3*/
```

相当于：

```
int a;
a=3;
float f=3.56;          /*指定 f 为实型变量，初值为 3.56*/
char c='a';            /*指定 c 为字符型变量，初值为'a'*/
```

也可以给定义的部分变量赋初值。例如：

```
int a, b=6, c;
```

注意，不能有如下的形式：

```
int a=b=c=6;
```

标准输入函数的使用在后面介绍。

因此，设计和运用变量时应注意以下原则：

- 变量必须先定义，后使用。
- 定义变量时指定该变量的名字和类型。
- 变量名和变量值是两个不同的概念。
- 变量名实际上是以一个名字代表一个存储地址的。

• 从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据的。

3.2 C 语言的数据类型

3.2.1 C 语言数据类型概述

程序和算法处理的对象统称为数据。数据以某种特定的形式存在，而且不同的数据还存在着某些联系。数据结构就是指数据的组织形式，它是以数据类型的形式出现的。C 语言数据类型如图 3-2 所示。

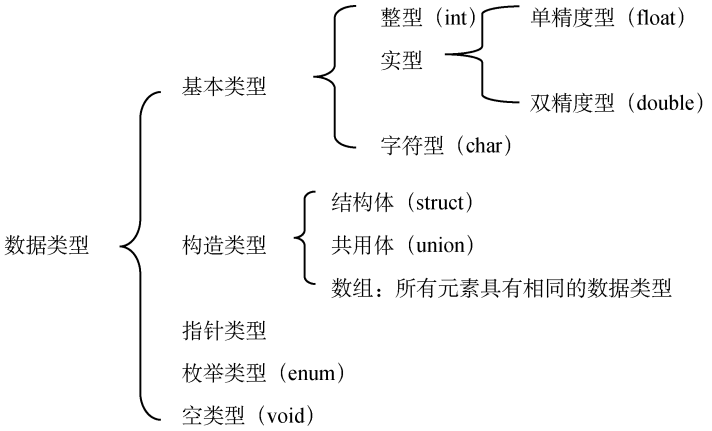

图 3-2 C 语言数据类型

在 C 程序中所用到的数据都必须指定其数据类型。指定了数据类型，也就定义出了数据在计算机内存中所占的空间字节数。

3.2.2 整型数据

1. 整型常量

整型常量用来表示一般数学中的整数，包括正整数、负整数和 0，所以整型常量也称整常数。

在 C 语言中，整型常量可用十进制数、八进制数和十六进制数三种形式表示。为识别不同进制表示的整型常量，C 语言规定，凡是以数字“0”开头的数字序列，一律当作八进制整数；凡是以 0x（数字“0”、字母“x”）开头的数字序列，则是十六进制整数。三种整型常量书写形式举例如下：

（1）十进制整数，如 0、−111、+15、21 等。

（2）八进制整数，如 00、−0111、+015、021，它们分别表示十进制整数 0、−73、+13、17。

（3）十六进制整数，如 0x0、−0x111、+0x15、0x21，它们分别表示十进制整数 0、−273、+21、33。

注意：正整数可以在前面加“+”号，也可以不加。负整数的前面必须加“−”号。

2. 整型变量

在 C 语言中，根据占用内存字节数的不同，整型变量可分为：基本整型、短整型、长整型和无符号整型。

（1）基本整型：以 int 表示。

（2）短整型：以 short int 表示，或以 short 表示。

（3）长整型：以 long int 表示，或以 long 表示。

（4）无符号整型：存储单元中全部二进制位（bit）用做存放数据，而不包括符号。无符号整型又分为无符号基本整型、无符号短整型和无符号长整型，分别以 unsigned int、unsigned short、unsigned long 表示。无符号整型变量只能存放不带符号的整数，不能存放负数。

以上几种类型的整型变量主要区别在于表示的整数范围不同，如表 3-1 所示。

表 3-1　整型数据分类表

数据类型	类型符	内存中所占位数	表示的数值范围
基本型	int	16	−32768～32767，即-2^{15}～($2^{15}-1$)
短整型	short	16	−32768～32767，即-2^{15}～($2^{15}-1$)
长整型	long [int]	32	−2147483648～2147483647，即-2^{31}～($2^{31}-1$)
无符号整型	unsigned [int]	16	0～65535，即 0～($2^{16}-1$)
无符号短整型	unsigned short [int]	16	0～65535，即 0～($2^{16}-1$)
无符号长整型	unsigned long [int]	32	0～4294967295，即 0～($2^{32}-1$)

注：在定义变量时，方括号[ ]内的关键字可以省略。

在将一个整型常量赋值给上述几种类型的整型变量时，应注意以下几点：

（1）一个整型常量，如果其值在−32768～+32767 范围内，认为它是 int 型，它可以赋值给 int 型和 long [int]型变量。

（2）一个整型常量，如果其值超过了上述范围，而在−2147483648～+2147483647 范围内，则认为它是长整型，可以将它赋值给一个 long [int]型变量。

说明：在一个整型常量的后面加一个字母 l 或 L，则认为是 long [int]型常量。例如，123l、432L。

（3）如果某一计算机系统的 C 版本（如 Turbo C）确定 short [int]与 int 型数据在内存中占据的长度相同，则它表示的数的范围与 int 型相同。因此，一个 int 型的常量也可同时赋给 int 型或 short [int]型变量。

（4）常量无 unsigned 型。一个非负值的整数可以赋值给 unsigned 型整型变量，只要它的范围不超过变量的取值范围即可。例如，将 50000 赋给一个 unsigned int 型变量是可以的，而将 70000 赋给它是不行的（溢出）。

例 3.2　整型变量的定义与使用。

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

```
void main( )
{
     int a,b,c,d;          /*指定 a，b，c，d 为整型变量 */
     unsigned  u;          /*指定 u 为无符号整型变量*/
     a=12;b=-24;u=10;
     c=a+u;d=b+u;
     printf（"c=%d,d=%d",c,d）;
}
```

运行结果如图 3-3 所示。

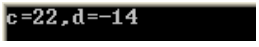

图 3-3　例 3.2 运行结果图

可以看到不同种类的整型数据可以进行算术运算。本例中是 int 型数据与 unsigned int 型数据进行相加相减运算（有关运算的规则在本章后面介绍）。

3. 整型数据的溢出

一个 int 型变量的最大允许值为 32767，如果再加上 1，会出现什么情况？我们来看这样一个例子。

例 3.3　整型数据的溢出。

```
#include <stdio.h>
void main( )
{
   int x,y;
   x=32767;
   y=x+1;
   printf（"%d,%d",x,y）;
}
```

运行结果如图 3-4 所示。

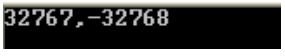

图 3-4　例 3.3 运行结果图

为什么会出现这样的结果呢？我们来看看数据的存储图，如图 3-5 所示。

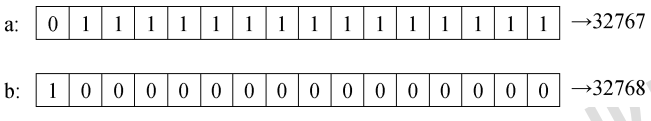

图 3-5　整数溢出示意图

从图 3-5 可以看到：变量 a 的最高位为 0，后 15 位全为 1。加 1 后变成第一位为 1，后面 15 位全为 0。而它是−32768 的补码形式，所以输出变量 b 的值为−32768。

由此可见，一个整型变量只能容纳−32768～32767 范围内的数，无法表示大于 32767

的数，遇此情况就发生溢出，而程序在运行时并不报错。32767 加 1 得不到 32768，却得到 −32768，这可能与程序编制者的原意不同。

C 语言的用法比较灵活，往往会出现一些副作用，而系统又不给出“出错信息”，要靠程序员的细心和经验来保证结果的正确。这里将变量 b 的类型改成 long 型就可得到预期的结果 32768。

3.2.3　实型数据

1. 实型常量

实型也称为浮点型。实型常量也称为实数或浮点数。在 C 语言中，实数只采用十进制。它有小数形式和指数形式两种表示形式：

日常数据：	18.6	0.00186	−186
小数形式：	18.6	0.00186	−186.0
指数形式：	1.86e1	1.86e−3	−1.86e2
或	18.6e0	18.6e−4	−0.186e3

在实型常量的书写中，需要注意以下几点：

（1）小数部分为 0 的实型常量，可以写为 186.0 或 186。

（2）用小数形式书写时，小数点的两边必须有数字，不能写成.86 或 86.等形式。

（3）用指数形式书写时，e（或 E）之前必须有数字，且 e 后面指数必须为整数。例如，e3、2e3.5、e 等都不是合法的指数形式。

2. 实型变量

（1）实型数据在内存中的存放形式。实型数据一般占 4 个字节（32 位）内存空间。与整型数据的存储方式不同，实型数据按指数形式存储。系统把一个实型数据分成小数部分和指数部分分别存放。指数部分采用规范化的指数形式。例如，实数 1.23456 在内存中的存放形式如图 3-6 所示。

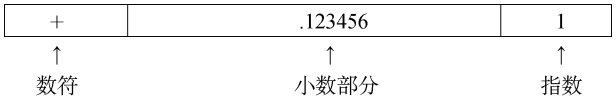

图 3-6　实数存储示意图

图 3-6 中的数字是用十进制数来示意的，实际上在计算机中是用十进制数来表示小数部分及用 2 的幂次来表示指数部分的。

在 4 个字节（32 位）中，C 编译系统以 24 位表示小数部分（包括符号），以 8 位表示指数部分（包括指数的符号）。小数部分占的位（bit）数越多，数的有效数字越多，精度越高。指数部分占的位数越多，则能表示的数值范围越大。

（2）实型变量的分类。C 语言将实型变量分为单精度（float 型）、双精度（double 型）两类。有关规定如表 3-2 所示。

表 3-2　实型数据

类型说明符	比特数（字节数）	有效数字	数的范围
float	32（4）	6～7	10^{-37}～10^{38}
double	64（8）	15～16	10^{-307}～10^{308}

实型变量定义的格式和书写规则与整型相同。例如：

float x, y;　（定义 x、y 为单精度实型）

double a, b, c;　（定义 a、b、c 为双精度实型）

需要说明的是，实型变量是由有限的存储单元组成的，因此能提供的有效数字是有限的。在有效位以外的数字将被舍去，由此可能会产生一些误差。

3. 实型常量的类型

实型常量是不分单、双精度的。一个实型常量可以赋给 float 型变量，也可以赋给 double 型的变量，系统都将按双精度（double）型进行处理。

在对实型常量进行计算时，系统先按双精度数据存储及运算，得到一个双精度的计算结果后，再取前 7 位值。虽然这样可以保证计算结果更精确，但是运算速度将受到影响。当然也可以在数字的后面通过加字母 f 或 F（如 3.45f、56.78F）来使编译系统将其按单精度处理。

值得注意的是，如果实型常量是 double 型的，当把该实型常量赋给一个 float 型变量时，系统会截取相应的有效位数。例如，

```
float  a;
a=123456.789;
```

由于 float 型变量只能接收 7 位有效数字，因此，最后两位小数不起作用，实际存储的 a 值为 123456.7。如果将 a 改为 double 型变量，则能全部接收上述 9 位数字并存储在变量 a 中。

3.2.4　字符型数据

1. 字符常量

C 语言中，把用单引号（撇号）括起来的单个字符看做字符常量。例如，'a'、'1'、'%'、'a'等都是字符常量。字符常量中的单引号只起分隔作用，称为字符常量的分隔符，它并不是字符常量的一部分。注意，'a'和'A'是不同的字符常量。

除了以上形式的字符常量，C 语言中还有一种特殊形式的字符常量，就是以一个“\”开头的字符序列。例如，前面已经遇到过的，printf 函数中的“\n”，它代表一个“换行”符。这种非显示字符难以用一般形式的字符表示，故规定用这种特殊形式表示。C 语言中把这种以“\”开头的字符序列称为“转义字符”。常用的转义字符如表 3-3 所示。

转义字符也就是将反斜杠（\）后面的字符转换成另外的意义。如“\n”中的“n”不代表字母 n 而作为“换行”符。

字符常量中使用单引号和反斜杠及双引号时，都必须使用转义字符表示，即在这些字符前加上反斜杠。

表 3-3　转义字符及其含义

转义字符	含　义	ASCII 码值（十进制）
\a	响铃（BEL）	007
\b	退格（不换行）	008
\f	走纸换页（FF）	012
\n	换行	010
\r	回车（CR）	013
\t	横向跳格（即跳到下一个输出区，占 8 列）	009
\v	竖向跳格（垂直制表）	011
\\	反斜杠字符“\”	092
\?	问号字符	063
\'	单引号（撇号）字符	039
\"	双引号字符	034
\0	空字符（NULL）	000
\ddd	1 到 3 位八进制数所代表的字符	
\xhh	1 到 2 位十六进制数所代表的字符	

在 C 程序中使用转义字符\ddd 或者\xhh 可以方便灵活地表示任意字符。\ddd 为斜杠后面可跟三位八进制数，代表这三位八进制数对应的八进制 ASCII 码值所对应的字符。例如，“\101”代表 ASCII 码值为 65 的字符“A”。“\012”（ASCII 码值为 10）代表“换行”。用“\376”，代表图形字符“■”。请注意“\0”或“\000”是代表 ASCII 码为 0 的控制字符，即“空操作”字符。它将用在字符串中。\x 后面可跟两位十六进制数，代表这两位十六进制数对应的十六进制 ASCII 码值所对应的字符。

使用转义字符时需要注意：

（1）转义字符中只能使用小写字母，每个转义字符只能看做一个字符。

（2）\v 垂直制表和\f 走纸换页符对屏幕没有任何影响，但会影响打印机执行响应操作。

（3）在 C 程序中，使用不可打印字符时，通常用转义字符表示。

例 3.4　转义字符的使用。

```
#include <stdio.h>
void main( )
{
   printf("ab c\t de\rf\tg\n");
   printf("h\ti\b\bj   k");
}
```

程序中没有设字符变量，用 printf 函数直接输出双引号内的各个字符。请注意其中的“转义字符”。第一个 printf 函数先在第一行左端开始输出“ab c”，然后遇到“\t”，它的作用是“跳格”，即跳到下一个输出位置，在我们所用系统中一个输出区占 8 列。下一输出位置从第 9 列开始，故在第 9～11 列上输出“de”。下面遇到“\r”，它代表“回车”（不换行），

返回到本行最左端（第 1 列），输出字符“f”，然后“\t”再使当前输出位置移到第 9 列，输出“g”。下面是“\n”，作用是“换行”。第二个 printf 函数先在第 1 列输出字符“h”，后面的“\t”使当前输出位置移到第 9 列，输出字母“i”，然后输出位置应移到下一列（第 10 列）准备输出下一个字符。下面遇到两个“\b”，“\b”的作用是“退格”，因此“\b\b”的作用是使当前输出位置回退到第 8 列，接着输出字符“j　k”。

程序运行时在打印机上得到以下结果：

f ab c　gde

h　　ji　k

注意：在显示屏上最后看到的结果与上述打印结果不同，如图 3-7 所示。

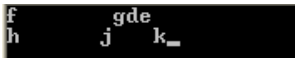

图 3-7　例 3.4 运行结果图

这是由于“\r”使当前输出位置回到本行开头，在此输出的字符（包括空格和跳格所经过的位置）将取代原来屏幕上该位置上显示的字符。所以原有的“ab c ”被新的字符“f　g”代替，其后的“de”未被新字符取代。回车换行后先输出“h　　i”，退两格后再输出“j　k”，j 后面第一个“␣”将原有的字符“i”取而代之。因此屏幕上看不到“i”。实际上，屏幕上完全按程序要求输出了全部的字符，只是因为在输出前面的字符后很快又输出后面的字符，在人们还未看清楚之前，新的已取代了旧的，所以误以为未输出应输出的字符。而在打印机输出时，不会像显示屏那样“抹掉”原字符，它能真正反映输出的过程和结果。

在 C 语言中，字符是按其所对应的 ASCII 码值来存储的，一个字符占一个字节。

注意：字符'9'和数字 9 的区别，前者是字符常量，后者是整型常量，它们的含义和在计算机中的存储方式都截然不同。

2. 字符串常量

C 语言除了允许使用字符常量，还允许使用字符串常量。字符串常量是一对双引号括起来的字符序列。例如："How are you"、"CHINESE"、"$ 123.45"、"COME & GO"、"88834−1234"、"A"都是字符串常量。C 语言可以输出一个字符串，例如：

printf　("How are you ")；

不要将字符常量与字符串常量混淆。字符常量'a'和字符串常量"a"。虽然都只有一个字符，但在内存中的情况是不同的。

'a'在内存中占一个字节，表示为：[a]

"a"在内存中占二个字节，表示为：[a][\0]

C 语言中，字符串常量在内存中存储时，系统自动在字符串的末尾加一个“串结束标志”，即 ASCII 码值为 0 的字符 NULL，常用“\0”表示，以便系统据此判断字符串是否结束。因此在程序中，长度为 *n* 个字符的字符串常量，在内存中占有 *n*+1 个字节的存储空间。例如，字符串"Study"有 5 个字符，作为字符串常量"Study"存储于内存中时，共占 6 个字节，

系统自动在后面加上 NULL 字符。

3. 字符变量

字符变量用来存放字符常量，在内存中占一个字节的存储空间，只能存放一个字符。在一个字符变量中无法放字符串。字符型的类型符为 char。字符变量的定义形式如下：

```
char   c1, c2;
```

它表示 c1 和 c2 为字符变量，各放一个字符。因此可以用下面语句对 c1、c2 赋值：

```
c1='a'; c2 ='b';
```

也可以在定义时赋值，即字符变量的初始化。例如：

```
char   c1='a', c2='b';
```

需要注意的是： C 语言中没有字符串变量，所以不能将一个字符串常量赋给一个字符型变量。例如，“char ch="china"”是错误的。

如果想存放一个字符串，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。相关内容参见后续章节。

4. 字符型数据在内存中的存储形式及其使用方法

将一个字符常量赋值给字符变量，实际上是将该字符的相应的 ASCII 码值以二进制形式放到存储单元中，并没有将字符本身放到变量的内存单元中去。正是由于字符数据的这种特殊存储形式（即与整型数据类似的存储形式），使得字符数据和整型数据之间可以通用，也就是说字符数据和整型数据之间可以进行运算。

对字符型数据进行算术运算，只是相当于对它们的 ASCII 码进行算术运算。同样，在输出时，一个字符数据既可以以字符形式输出，也可以以整数形式输出，以字符形式输出时，需要先将存储单元中的 ASCII 码值转换成相应的字符，然后输出；以整数形式输出时，直接将 ASCII 码作为整数输出。

注意：字符数据只占一个字节，它只能存放 0～255 范围内的整数。

例 3.5　向字符变量赋整数值。

```
#include <stdio.h>
void main( )
{
    char c1,c2;
    c1=97;  c2=98;
    printf("%c%c",c1,c2);
}
```

c1、c2 被指定为字符变量。但在第 4 行中，将整数 97 和 98 分别赋给 c1 和 c2，它的作用相当于以下两个赋值语句：

```
c1='a'; c2='b';
```

因为'a'和'b'的 ASCII 码为 97 和 98。第 5 行将输出两个字符。“%c”是输出字符的格式。程序输出如图 3-8 所示。

图 3-8　例 3.5 运行结果图

例 3.6　大小写字母转换。

```
#include <stdio.h>
void main( )
{
    char c1,c2;
    c1='a'; c2='b';
    c1=c1-32;  c2=c2-32;
    printf("%c%c",c1,c2);
}
```

运行结果如图 3-9 所示。

图 3-9　例 3.6 运行结果图

程序的作用是将两个小写字母 a 和 b 转换为大写字母 A 和 B。因为'a'的 ASCII 码值为 97，而'A'的 ASCII 码值为 65，'b'的 ASCII 码值为 98，'B'的 ASCII 码值为 66。从 ASCII 码表中可以看到每一个小写字母比大写字母的 ASCII 码值大 32。即'a'='A'+32。

3.3　不同类型数据的混合运算

3.3.1　不同数值型数据间的混合运算与类型转换

在 C 语言中，整型、单精度型、双精度型数据可以混合运算。前面我们已经知道，字符型数据可以与整型通用，因此，整型、实型（包括单、双精度）、字符型数据间可以混合运算。

不同类型数据进行混合运算时，首先要把不同类型的数据转换成同一类型，然后进行运算。这种转换由编译系统自动完成。其转换遵循以下规则：

• 在运算中，先进行水平方向上的转换。这种水平方向上的转换是必须要进行的。即使是两个 char 类型的数据进行运算，也要先转换成整型数据再运算。同样的，所有的浮点运算都是以双精度类型进行的。

• 如果在进行了水平方向上的转换后，仍存在不同类型的数据，则按纵向方向进行转换。纵向的箭头表示当运算对象为不同类型时转换的方向，即由下向上，数据类型逐步升高。

例如，int 型与 doub1e 型数据进行运算，先将 int 型的数据转换成 double 型，然后对两个同类型（double 型）数据进行运算，结果为 double 型。

这种纵向方向上的转换要将运算式中低级别的类型向本式中最高级别的类型转换，运算结果的类型与该式子中最高级别的类型相同，如图 3-10 所示。

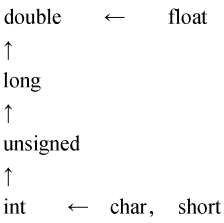

图 3-10　数据自动转换次序

例如，公式 10+'a'+0.3/3−2.1*3L。

计算机执行时从左至右扫描，运算过程为：

（1）先将字母'a'转换为整型数 97，0.3 转换为 double 型，2.1 转换成 double 型。

（2）进行 10+'a'的运算，经过第（1）步的转换，本式中两个数据的类型都是整型，所以不再转换类型，运算结果为 107。

（3）进行 0.3/3 的运算，经过第（1）步的转换，0.3 转换为 double 型，所以整数 3 也必须转换为 double 型。

（4）进行 2.1*3L 的运算，经过第（1）步的转换，2.1 转换为 double 型，长整型数 3L 也必须转换为 double 型。

（5）在第（2）～（4）步得到 3 个量，类别分别是整型、实型，在进行最后运算前，先要将第（2）步的结果转换为 double 型。

（6）这个式子最后结果为 double 型。

上述的类型转换是由系统自动进行的。

3.3.2　赋值运算中的数据类型转换

C 语言规定，在赋值运算中，如果赋值运算符两边表达式的类型不一致，但都是数值型或字符型时，在赋值时将进行类型转换。转换时将赋值号右边表达式的类型转换为左边变量的类型。

（1）将实型数据（包括单、双精度）赋给整型变量时，实数的小数部分将被舍弃。如 i 为整型变量，执行 i=3.68 的结果是使 i 的值为 3。

（2）将整型数据赋给单、双精度变量时，数值不变，但补足有效位数，以浮点数形式存储到变量中。如将 32 赋给 float 型变量 f，即 f=32，先将 32 转换成 32.00000，再存储到 f 中。如将 32 赋给 double 型变量 d，即 d=32，则将 32 补足有效位数字为 32.00000000000000，然后以双精度浮点数形式存储到 d 中。

（3）将一个 float 型数据赋给 double 型变量时，数值不变，有效位数扩展到 16 位，在内存中以 64 位（bit）存储。

（4）将一个 double 型数据赋给 float 型变量时，截取其前面 7 位有效数字，存放到 float 型变量的存储单元（32 位）中。但应注意数值范围不能溢出。如果有以下语句：

```
float  f;
double  d=123.456789e100;
f=d;
```

则会出现溢出错误。

（5）将字符型数据赋给整型变量时，由于字符只占一个字节，而整型变量为 2 个字节，因此将字符数据（8 位）放到整型变量低 8 位中。若字符最高位为 0，则整型变量高 8 位补 0；若字符最高位为 1，则高 8 位全补 1。这称为“符号扩展”。这样做的目的是使数值保持不变，如变量 c（字符'\376'）以整数形式输出为−2，i 的值也是−2。

（6）将一个 int、short、long 型数据赋给一个 char 型变量时，只将其低 8 位原封不动地送到 char 型变量中（即截断）。例如：

```
int  i=289;
char  c='a';
c=i;
```

赋值情况如图 3-11 所示，c 的值为 33，如果用“%c”输出 c，将得到字符“!”（其 ASCII 码为 33）。

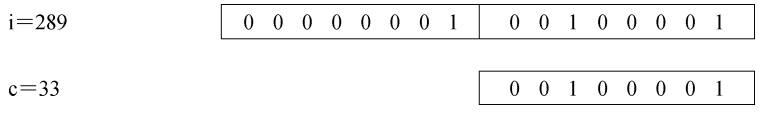

图 3-11　整型数据赋给字符变量示意图

（7）将带符号的整型数据（int 型）赋给 long int 型变量时，要进行符号扩展。将整型数据的 16 位送到 long 型低 16 位中，如果 int 型数据为正值（符号位为 0），则 long int 型变量的高 16 位补 0；如 int 型数据为负值（符号位为 1），则 long 型变量的高 16 位补 1，以保持数值不改变。

反之，若将一个 long int 型数据赋给一个 int 型变量，只将 long int 型数据中低 16 位原封不动送到整型变量中（即截断）。

（8）将 unsigned int 型数据赋给 long int 型变量，不存在符号扩展问题，只需将高位补 0 即可。将一个 unsigned int 型数据赋给一个占字节数相同的整型变量，例如，unsigned int →int，unsigned long→long，unsigned short→short，将 unsigned 型变量的内容原样送到 unsigned 型变量中，但如果数据范围超过相应整型的范围，则会出现数据错误。例如：

```
unsigned int a=65535;
int b;
b=a;
```

将 a 整个送到 b 中，由于 b 是 int 型变量，最高位是符号位，所以 b 成了负数（−1 的补码）。

unsigned a 对应 1111111111111111，int b 对应 1111111111111111。

可以用“printf（"%d", b）；”来验证一下。

（9）将 unsigned 型数据赋给长度相同的 unsigned 型变量，也是原样照赋（连原有的符号位也作为数值一起传送）。

例 3.7　不同类型整型数间的赋值。

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

```
void main( )
{
    unsigned  a;
    int  b=-1;
    a=b;
    printf("%u",a);
}
```

“%u”是输出无符号数时所用的格式符。运行结果如图 3-12 所示。

图 3-12　例 3.7 运行结果图

赋值的规则看起来较复杂，其实，不同类型的整型数间的赋值就是按存储单元中的存储形式直接传送的。

3.3.3　强制类型转换

强制类型转换是利用强制类型转换运算符将一个表达式转换成所需类型，它常被称为显示类型转换，而把自动类型转换称为隐式类型转换。其一般形式为：

(类型符)(表达式)

其功能就是把表达式结果的类型转换为圆括号( )中的数据类型。例如：

(double) a　　/*将 a 转换成 double 类型*/

(int)(x+y)　　/*将 x+y 的值转换成整型*/

(float)(5%3)　　/*将 5%3 的值转换成 float 型*/

注意：表达式一般应该用括号括起来（单个变量可以不加括号）。如果写成：

(int)x+y

则只将 x 转换成整型，然后与 y 相加。

需要说明的是，在强制类型转换时，得到一个所需类型的中间变量，而不改变数据说明时对该变量定义的类型，即原来变量的类型未发生变化。例如：

(int)x　　/*不要写成 int x( )*/

如果 x 原指定为 float 型，进行强制类型转换后得到一个 int 型的中间变量，它的值等于 x 的整数部分，而 x 的类型不变（仍为 float 型）。

例 3.8　强制类型转换示例。

```
#include <stdio.h>
void main( )
{
   float f=5.75;
   printf("(int)f=%d,f=%f\n",(int)f,f);
}
```

程序运行结果如图 3-13 所示。

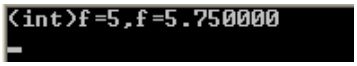

图 3-13　例 3.8 运行结果图

f 虽强制转换为 int 型，但只在运算中起作用，是临时的，而 f 本身的类型并不改变。因此，(int)f 的值为 5（删去了小数部分），而 f 的值仍为 5.75。

从上可知：有两种类型转换，一种是在运算时不必用户指定，系统自动进行类型转换；第二种是强制类型转换，当自动类型转换不能实现目的时，可以用强制类型转换。如“%”运算符要求其两侧均为整型量，若 x 为 float 型，则“x%3”不合法，必须用(int)x%3。

3.4　运算符和表达式

3.4.1　C 语言运算符简介

运算符是告诉编译程序执行特定算术或逻辑操作的符号，C 语言提供了多种运算符，具有很强的运算能力。C 语言的运算符可分为以下几类。

（1）算术运算符：用于各类数值运算，包括加（+）、减（−）、乘（*）、除（/）、求余（或称模运算，%）、自增（++）、自减（−−）共 7 种。

（2）关系运算符：用于比较运算，包括大于（>）、小于（<）、等于（==）、大于等于（>=）、小于等于（<=）和不等于（!=）共 6 种。

（3）逻辑运算符：用于逻辑运算，包括与（&&）、或（||）、非（!）共 3 种。

（4）位操作运算符：参与运算的量，按二进制位进行运算，包括位与（&）、位或（|）、位非（～）、位异或（^）、左移（<<）、右移（>>）共 6 种。

（5）赋值运算符：用于赋值运算，分为简单赋值（=）、复合算术赋值（+=，−=，*=，/=，%=）和复合位运算赋值（&=，|=，^=，>>=，<<=）三类共 11 种。

（6）条件运算符：这是一个三目运算符，用于条件求值（?:）。

（7）逗号运算符：用于把若干表达式组合成一个表达式（,）。

（8）指针运算符：用于取内容（*）和取地址（&）两种运算。

（9）求字节数运算符：用于计算数据类型所占的字节数（sizeof）。

（10）特殊运算符：有括号( )，下标[]，成员（→，.）等几种。

3.4.2　算术运算符和算术表达式

1. 基本的算术运算符

C 语言中算术运算符如表 3-4 所示。

表 3-4　算术运算符

运 算 符	作　用	运 算 符	作　用
−	减法（或取负）	%	求余运算

续表

运 算 符	作　　用	运 算 符	作　　用
+	加法（或取正）	/	除法
*	乘法		

加法运算符“+”：应有两个量参与运算，为双目运算符，如 5+6、b+c 等。

减法运算符“−”：双目运算符，如 6−4、y−1 等。但“−”也可做负值运算符，如−x，−5 等，此时为单目运算符，具有右结合性。

乘法运算符“*”：双目运算符，如 6*8。

除法运算符“/”：双目运算符。进行除法运算的量均为整型变量时，结果也为整型量，舍去小数。如果运算量中有一个是实型量，则结果为双精度实型量，如 4/3 结果为 1，8/3 结果为 2，6.0/4 结果为 1.5。

求余运算符“%”：也称模运算符，双目运算符。求余运算要求参与运算的量均为整型量，运算的结果等于两数相除后的余数，如 6%4 的值为 2。求余运算符“%”不能用于 float 和 double 类型。

2. 算术表达式和运算符的优先级与结合性

算术表达式是指用算术运算符和括号将运算对象（也称操作数）连接起来的符合 C 语言语法规则的式子。这里所说的运算对象包括常量、变量、函数等。a*b / c−1.5+'a'就是一个合法的 C 语言算术表达式。

对运算符的优先级和结合性，C 语言有专门的规定：

（1）在表达式求值时，先按运算符优先级别的高低次序执行。例如，先乘除后加减。如 a−b*c，b 的左侧为减号，右侧为乘号，而乘号优先于减号，因此，相当于 a−（b*c）。

（2）如果在一个运算对象两侧的运算符的优先级别相同，则按规定的“结合方向”处理。

C 语言规定了各种运算符的结合方向（结合性），算术运算符的结合方向为“自左至右”，即先左后右，如：a−b+c，运算时 b 先与减号结合，执行 a−b 的运算，再执行加 c 的运算。自左至右的结合方向又称“左结合性”，即运算对象先与左面的运算符结合。以后可以看到有些运算符的结合方向为“自右至左”，即“右结合性”，例如赋值运算符。

如果一个运算符两侧的数据类型不同，则会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。

3. 自增、自减运算符

自增（++）和自减（−−）运算符是 C 语言中两个很常用的运算符。运算符“++”是操作数加 1，而“−−”是操作数减 1，即：

++x; 等同于 x=x+1;

−−x; 等同于 x=x−1;

自增和自减运算符可放在操作数之前，也可放在其后，但在表达式中这两种用法是有区别的。例如：x=x+1; 可写成++x; 或 x++;。

（1）自增或自减运算符在操作数之前，称为前置，此时 C 语言在引用操作数之前就先执行加 1 或减 1 操作。

（2）自增或自减运算符在操作数之后，称为后置。C 语言先引用操作数的值，而后再进行加 1 或减 1 操作。例如：

①++i, −−i;：表示在使用 i 之前，先使 i 的值加（减）1。

②i++, i−−;：表示在使用 i 之后，使 i 的值加（减）l。

总的来看，++i 和 i++的作用相当于 i=i+1。但++i 和 i++不同之处在于++i 是先执行 i=i+1 后，再使用 i 的值；而 i++是先使用 i 的值后，再执行 i=i+1。如果 i 的原值等于 5，则：

```
j=++i;        /*j 的值为 6*/
j=i++;        /*j 的值为 5，然后 i 变为 6*/
```

又如：

```
i=5;
printf  ("%d",++i) ;
```

输出 6。

若改为：

```
printf ("%d",i++) ;
```

则输出 5。

关于自增运算符和自减运算符需要注意以下两点：

（1）自增运算符（++）和自减运算符（−−）只能用于变量，不能用于常量或表达式。如 5++或(a+b)++都是不合法的。因为 5 是常量，常量的值不能改变。(a+b)++也不可能实现，假如 a+b 的值为 6，那么自增后得到的 7 放在什么地方呢？无变量可供存放。

（2）++和−−和负号运算符（−）的优先级别是一样的，但比正号运算符的优先级别高。如对+i++；先算优先级别高的++，再进行正号运算符的运算。则上式相当于+(i++)，如果 i 的初值为 5，那么整个表达式的值为 5，在得出表达式的值后，i 再增加 1，变成 6。

++和−−的结合方向是“自右至左”。

如−i++;因负号运算符和“++”运算符同优先级，那么表达式的计算就要按结合方向进行。负号运算符和自增运算符的结合方向都是“自右至左”（右结合性），所以整个表达式相当于−(i++)，先算右边的 i++，再与负号运算符结合。如果 i 的初值为 5，那么整个表达式的值为−5，在得出表达式的值后，i 再增加 1，变成 6。

同样地，−(++i);中，如果 i 的初值为 5，那么整个表达式的值为−6，i 的值为 6。

3.4.3 赋值运算符

1. 赋值运算符

赋值符号“=”就是赋值运算符，它的作用是将一个数据赋给一个变量。例如：

a=3;

其作用是执行一次赋值操作（或称赋值运算），把常量 3 赋给变量 a。

r=x/y;

其作用是将表达式 x/y 的值赋给变量 r。

C 语言中赋值运算符“=”和数学中的等号“=”是有区别的，它不是表示“等同”的关系，而是进行“赋值”的操作。

（1）赋值表达式 x=y 的作用是将变量 y 所代表的存储单元中的内容赋给变量 x 所代表的存储单元，x 中原有的数据将被取代；赋值后，y 变量中的内容保持不变。此表达式应理解为“把右边变量中的值赋给左边变量”，而不应理解为“x 等于 y”。

（2）赋值表达式 m=m+1，表示取变量 m 中的值加 1 后再放入到变量 m 中，使变量 m 的值增 1。而在数学中，m=m+1 则是完全不能成立的等式。

（3）赋值运算符的左边只能是变量而不能是常量或表达式。如 a+b=c 这个式子就不是合法的赋值表达式。因为系统已为 a、b、c 分别分配一一对应的存储单元，内存中并无 a+b 这样的存储单元。

2. 复合赋值运算符

在 C 语言中，在赋值运算符“=”之前加上其他运算符，就构成复合赋值运算符。比如在“=”前加一个“+”运算符就成了复合运算符“+=”。

参加算术复合运算的两个运算数，先进行算术运算，然后将其结果赋给第一个运算数，例如：

```
a+=3          /*  等价于   a=a+3 */
x*=y+8        /*  等价于   x=x*(y+8)*/
x%=3          /*  等价于     x=x%3 */
```

以“a+=3”为例来说明，它先使 a 加 3，再赋给 a。同样，“x*=y+8”的作用是使 x 乘以(y+8)，再赋给 x。

C 语言规定，凡是二元（二目）运算符，都可以与赋值运算符一起组合成复合赋值运算符。可以使用的复合赋值运算符有 10 种，分别是：+=，−=，*=，/=，%=，<<=，>>=，&=，^=，|=。其中，后 5 种是有关位运算的。

C 语言采用这种复合赋值运算符，一是为了简化程序，使程序精练；二是为了提高编译效率。

3. 赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为“赋值表达式”。

它的一般形式为：

<变量> <赋值运算符> <表达式>

如“a=5”是一个赋值表达式。对赋值表达式求解的过程是：将赋值运算符右侧的“表达式”的值赋给左侧的变量。赋值表达式的值就是被赋值的变量的值。例如，“a=5”这个赋值表达式的值为 5（变量 a 的值也是 5）。

上述一般形式的赋值表达式中的“表达式”，又可以是一个赋值表达式。例如：

a=(b=6)

括号内的“b=6”是一个赋值表达式，它的值等于 6，因此“a=(b=6)”相当于“a=6”，a 的值等于 6，整个赋值表达式的值也等于 6。

C 语言中，赋值运算符按照“自右至左”的顺序结合，因此，“b=6”外面的括弧可以不要，即“a=(b=6)”和“a=b=6”等价，都是先求“b=6”的值（得 6），然后再赋给 a，下面是赋值表达式的例子：

```
a=b=c=6                 /* 赋值表达式值为 6，a、b、c 值均为 6 */
a=5+(c=6)               /* 表达式值为 11，a 值为 11，c 值为 6 */
a=（b=4）+（c=6）       /* 表达式值为 10，a 值为 10，b 值为 4，c 值为 6 */
a=（b=10）/（c=2）      /* 表达式值为 5，a 等于 5，b 值为 10，c 值为 2 */
```

赋值表达式也可以包含复合的赋值运算符。例如：

```
a+=a-=a*a
```

也是一个赋值表达式。如果 a 的初值为 12，此赋值表达式的求解步骤如下：

①先进行“a-=a*a”的运算，它相当于 a=a-a*a=12-144=-132。

②再进行“a+=-132”的运算，相当于 a=a+(-132)=-132-132=-264。

3.4.4 关系运算符和关系表达式

1. 关系运算符

关系运算符是双目运算符，其功能是将两个运算对象进行比较大小。C 语言中提供了 6 种关系运算符，关系运算实际上是比较运算，表示两个运算分量之间的大小关系，如大于、小于等。关系运算则是对两个运算量的数值进行比较的过程。关系运算符是双目运算符，C 语言提供了 6 种关系运算符，分别是：

```
>     大于
>=    大于等于
<     小于
<=    小于等于
==    等于
!=    不等于
```

说明：

（1）其中前 4 种关系运算符（<，<=，>，>=）的优先级别相同，后两种运算符的优先级也相同，但前 4 种的优先级高于后两种。例如，“>”优于“==”，而“>”与“<”优先级别相同。

（2）关系运算符的优先级低于算术运算符，而高于赋值运算符，如图 3-14 所示。

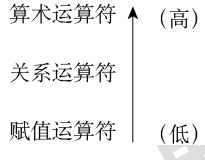

图 3-14 关系运算符优先级

（3）关系运算符都是左结合性的，即自左至右。

2. 关系表达式

用关系运算符将两个表达式（可以是算术表达式、关系表达式、逻辑表达式、赋值表达式、字符表达式）连接起来的有意义的式子称为关系表达式。例如，下面都是合法的关系表达式：a>b，a+b>b+c，(a=3)>(b=5)，'a'<'b'，(a>b)>(b<c)。

关系表达式的值按比较结果分为逻辑真和逻辑假，通常若关系表达式成立，则该表达式的值为“真”，用整数 1 表示；若关系表达式不成立，则该表达式的值为“假”，用整数 0 表示，即“真”或“假”。以 1 代表“真”，以“0”代表“假”。

若关系表达式表达的关系成立，则它的结果值为“1”，否则为“0”。例如：

x==y：x 等于 y 时，关系表达式成立，其值为 1；x 不等于 y 时，关系表达式不成立，其值为 0。

x！=y：x 不等于 y 时，关系表达式成立，其值为 1，否则，关系表达式不成立，其值为 0。

a=x＞y：等价于 a=(x＞y)；若 x＞y，则 a=1，否则 a=0。

关系表达式常用于在流程控制中作为分支或循环的条件，关系运算符的结合方向为自左至右。

3.4.5　逻辑运算符和逻辑表达式

1. 逻辑运算符

C 语言中提供了 3 种逻辑运算符，分别是：

&& 逻辑与　（相当于其他语言中的 AND）

|| 逻辑或　（相当于其他语言中的 OR）

! 逻辑非　（相当于其他语言中的 NOT）

其中运算符“&&”和“||”是双目运算符，要求有两个运算分量，用于连接多个条件，构成更复杂的条件。运算符“!”是单目运算符，用于对给定条件取“反”。逻辑运算产生的结果也是一个逻辑量：真或假，分别用 1 和 0 表示。

逻辑运算符中，“!”的优先级最高，其次是“&&”，“||”的优先级最低。另外“&&”和“||”的优先级低于关系运算符的优先级，但高于算术运算符的优先级；而“!”的优先级高于算术运算符的优先级。

2. 逻辑表达式

逻辑表达式是用逻辑运算符将算术表达式、关系表达式或逻辑量连接起来的式子。

“&&”和“||”构成的逻辑表达式一般形式为：

a 逻辑运算符 b

! 构成的逻辑表达式一般形为：

!a

其中 a、b 为表达式或逻辑量。

逻辑运算符进行运算时的运算规则为：

a&&b：若 a、b 同时为真，则 a&&b 为真；若 a、b 中有一项为假，则 a&&b 为假。

a||b：若 a、b 同时为假，则 a||b 为假；若 a、b 中有一项为真，则 a||b 为真。

!a：若 a 为真，则!a 为假；若 a 为假，则!a 为真。

注意：C 语言编译系统在给出逻辑运算结果时，以数值 1 代表“真”，以 0 代表“假”，但在判断一个量是否为“真”时，以 0 代表“假”，以非 0 代表“真”，即将一个非零的数值认为“真”。例如：

①若 a=4，则!a 的值为 0。因为 a 的值为非 0，被认为“真”，对它进行“非”运算，得“假”，“假”以 0 代表。

②若 a=4，b=5，则 a&&b 的值为 1。因为 a 和 b 均为非 0，被认为“真”，因此 a&&b 的值也为“真”，值为 1。

如果在一个表达式中不同位置上出现数值，应区分哪些作为数值运算或关系运算，哪些作为逻辑运算对象。

实际上，逻辑运算符两侧的运算对象不但可以是 0 和 1，或者是 0 和非 0 的整数，也可以是任何类型的数据，还可以是字符型、实型或指针型等。系统最终以 0 和非 0 来判断它们属于“真”或“假”。例如：

'c'&&'d'的值为 1（因为'c'和'd'的 ASCII 码值都不为 0，按“真”处理）。

我们注意到，“逻辑与”和“逻辑或”运算分别有如下性质：

a&&b　　当 a 为 0 时，不管 b 为何值，结果为 0。

a || b　　当 a 为 1 时，不管 b 为何值，结果为 1。

C 语言利用上述性质，在计算连续的逻辑与运算时，若有运算分量的值为 0，则不再计算后继的逻辑与运算分量，并以 0 作为逻辑与算式的结果；在计算连续的逻辑或运算时，若有运算分量的值为 1，则不再计算后继的逻辑或运算分量，并以 1 作为逻辑或算式的结果。也就是说，对于 a&&b，仅当 a 为非零时，才计算 b；对于 a||b，仅当 a 为 0 时，才计算 b。

例如，设 x=1，y=1，z=0，则执行表达式(y&&x)||(z++)后，z 的值为 0。计算过程是：先求 y&&x 的值为 1，由于接下来进行逻辑或运算，不管第二个分量值如何，整个表达式值为 1，从而不再计算第二个分量 z++，故 z 的值仍为 0。

同算术表达式一样，在关系或逻辑表达式中也使用括号来修改原计算顺序。

因为所有关系和逻辑表达式产生的结果不是 0 就是 1，所以下面的程序段不仅正确而且将在屏幕上打印数值 1。

```
int   x;
x=100;
printf ("%d", x>10);
```

3.4.6　逗号运算符和逗号表达式

C 语言提供一种特殊的运算符——逗号运算符，用它将两个表达式连接起来。例如：

3+4, 5+6

称为逗号表达式。逗号表达式的一般形式为：

表达式 1, 表达式 2

逗号表达式的求解过程是：先求解表达式 1，再求解表达式 2。整个逗号表达式的值是表达式 2 的值。例如，上面的逗号表达式“3+4, 5+6”的值为 11。又如，逗号表达式：

a=3*5, a*4

先求解 a=3*5，得 a 的值为 15，然后求解 a*4 得 60，整个逗号表达式的值为 60。

一个逗号表达式又可以与另一个表达式组成一个新的逗号表达式。例如：

(a=3*5, a*4), a+5

先使 a 的值等于 15，再计算 a*4（但 a 值未变），最后计算 a+5 得 20，即整个表达式的值为 20。

逗号表达式的一般形式可以扩展为：

表达式 1, 表达式 2, 表达式 3, …, 表达式 *n*

它的值为表达式 *n* 的值。

逗号运算符是所有运算符中级别最低的。因此，下面两个表达式的作用是不同的：

① x=(a=3, 6*3)

② x=a=3, 6*a

表达式①是一个赋值表达式，将一个逗号表达式的值赋给 x，x 的值等于 18。表达式②是一个逗号表达式，值为 18。它包括一个赋值表达式和一个算术表达式，x 的值为 3。

通过以下分析，使用逗号表达式时需注意：

（1）逗号表达式无非是把若干个表达式“串联”起来。在许多情况下，使用逗号表达式的目的只是想分别得到各个表达式的值，而并非一定需要得到和使用整个逗号表达式的值，逗号表达式最常用于 for 循环语句中，详见第 4 章。

（2）逗号运算符又称为“顺序求值运算符”，但并不是任何地方出现的逗号都是作为逗号运算符的。函数参数也是用逗号来间隔的。例如：

Printf ("%d, %d, %d", a, b, c);

上一行中的“a, b, c”并不是一个逗号表达式，它是 printf 函数的三个参数，参数间用逗号间隔。如果改写为：

Printf ("%d, %d, %d", (a, b, c), b, c);

则“(a, b, c)”是一个逗号表达式，它的值等于 c 的值。括号内为一个表达式，括号内的逗号不是参数间的分隔符而是逗号运算符。

C 语言表达能力强，其中一个重要方面就在于它的表达式类型丰富，运算符功能强，因而 C 语言使用灵活，适应性强。在后面几章中将会看到这一点。

3.4.7　条件运算符和条件表达式

条件运算符由两个运算符组成，分别是“?”“:”。这是 C 语言提供的唯一的三目运算符，即要求有三个运算对象。条件运算符优先于赋值运算，但低于逻辑运算、关系运算和

算术运算。

由条件运算符构成的表达式称为条件表达式，其形式如下：

表达式 1? 表达式 2: 表达式 3

条件表达式的求解过程是：当“表达式 1”的值为非零时，求出“表达式 2”的值，此时“表达式 2”的值就是整个条件表达式的值；当“表达式 1”的值为零时，去求“表达式 3”的值，这时便把“表达式 3”的值作为整个条件表达式的值。

例如，假设 a=1，b=2，c=3，d=4，求表达式 a<b ? a : c < d ? a : d 的值。

分析：条件表达式嵌套使用时，根据条件运算符自右至左的结合性，应先将最后一个“?”与其最近的“:”配对，即该表达式等价于“a<b ? a : (c < d ? a : d)”。当 a<b 成立时，取 a 的值作为该条件表达式的值。

故该表达式的值为 1。

3.4.8 位运算

前面介绍的各种运算都是以字节作为最基本单位进行的。但是，在很多系统程序中常要求在位（bit）一级进行运算或处理。C 语言提供了位运算的功能，这使得 C 语言也能像汇编语言一样用来编写系统程序。

位运算是 C 语言的一种特殊运算功能，它是以二进制位为单位进行运算的。位运算符只有逻辑运算和移位运算两类。在 C 语言中，提供了以下 6 种位运算符：

& 按位与
| 按位或
^ 按位异或
～ 取反
<< 左移
>> 右移

1. 按位与运算

按位与运算符“&”是双目运算符。其功能是参与运算的两数各对应的二进位相与。只有对应的两个二进位数均为 1，结果位才为 1，否则为 0。参与运算的数以补码方式出现。按位与运算通常用来对某些位清 0 或保留某些位。

例如，3&5 可写算式如下：

```
    00000011    （3 的二进制补码）
&   00000101    （5 的二进制补码）
-----------------------------------
    00000001    （1 的二进制补码）
```

故可得：3&5 的值为 1。

2. 按位或运算

按位或运算符“|”是双目运算符。其功能是参与运算的两数各对应的二进位相或。只

要对应的两个二进位数有一个为 1，结果位就为 1。参与运算的两个数均以补码出现。

例如，3|5 可写算式如下：

```
    00000011    （3 的二进制补码）
|   00000101    （5 的二进制补码）
-----------------------------------
    00000111    （7 的二进制补码）
```

故可得：3|5 的值为 7。

3. 按位异或运算

按位异或运算符“^”是双目运算符。其功能是参与运算的两数各对应的二进位相异或，当两个对应的二进位数相异时，结果为 1，否则为 0。参与运算数均以补码出现。

例如，3^5 可写算式如下：

```
    00000011    （3 的二进制补码）
^   00000101    （5 的二进制补码）
-----------------------------------
    00000110    （6 的二进制补码）
```

故可得：3^5 的值为 6。

4. 求反运算

求反运算符“～”为单目运算符，具有右结合性。其功能是对参与运算的数的各二进位按位求反。

例如，～3 的运算为～(0000000000000011)，结果为 1111111111111100。

5. 左移运算

左移运算符“<<”是双目运算符。其功能把“<<”左边的运算数的各二进位全部左移若干位，由“<<”右边的数指定移动的位数，高位丢弃，低位补 0。

例如，设 a 的值为 3，a<<2 指把 a 的各二进位向左移动 2 位。即 a=00000011（十进制数 3），左移 2 位后为 00001100（十进制数 12）。

6. 右移运算

右移运算符“>>”是双目运算符。其功能是把“>>”左边的运算数的各二进位全部右移若干位，“>>”右边的数指定移动的位数。

例如，设 a 的值为 15，a>>2 表示把 000001111（十进制数 15）右移为 00000011（十进制 3）。

注意：对于有符号数，在右移时，符号位将随之移动。当为正数时，最高位补 0；而为负数时，符号位为 1，最高位是补 0 或是补 1 取决于编译系统的规定。Turbo C 和很多系统规定为补 1。

此外，位运算符还可以与赋值符一起组成复合赋值符，如&=、|=、^=、>>=、<<=等。

习　题

一、选择题

1. 以下选项中，不正确的 C 语言浮点型常量是（　　）。

A. 160　　B. 0.12　　C. 2e4.2　　D. 0.0

2. 以下选项中，（　　）是不正确的 C 语言字符型常量。

A. 'a'　　B. '\x41'　　C. '\101'　　D. "a"

3. 在 C 语言中，字符型数据在计算机内存中，以字符的（　　）形式存储。

A. 原码　　B. 反码　　C. ASCII 码　　D. BCD 码

4. 若 x、i、j 和 k 都是 int 型变量，则计算下面表达式后，x 的值是（　　）。

x=(i=4, j=16, k=32)

A. 4　　B. 16　　C. 32　　D. 52

5. 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

6. 若有代数式 $\frac{3ae}{bc}$，则不正确的 C 语言表达式是（　　）。

A. a/b/c*e*3　　B. 3*a*e/b/c　　C. 3*a*e/b*c　　D. a*e/c/b*3

7. 表达式!x||a==b 等效于（　　）。

A. !((x||a)==b)　　B. !(x||y)==b　　C. !(x||(a==b))　　D. (!x)||(a==b)

8. 假设整型变量 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

9. 假设有语句“int a=3;”，则执行了语句“a+=a−=a*=a;”后，变量 a 的值是（　　）。

A. 3　　B. 0　　C. 9　　D. −12

10. 在以下运算符中，优先级最低的运算符是（　　）。

A. *　　B. !=　　C. +　　D. =

11. 假设整型变量 i 值为 2，表达式(++i)+(++i)+(++i)的结果是（　　）。

A. 6　　B. 12　　C. 15　　D. 表达式出错

12. 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

13. sizeof (double)的结果是（　　）。

A. 8　　B. 4　　C. 2　　D. 出错

14. 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

15. 假设 a 为整型变量，不能正确表达数学关系：10<a<15 的 C 语言表达式是（　　）。

A. 10<a<15　　B. a= =11|| a= =12 || a= =13 || a= =14

C. a>10 && a<15　　D. !(a<=10) && !(a>=15)

16. 假设 f 是实型变量，下列表达式中不是逗号表达式的是（　　）。

A. f= 3.2, 1.0　　B. f>0, f<10　　C. f=2.0, f>0　　D. f=(3.2, 1.0)

17. 表达式 18/4*sqrt(4.0)/8 值的数据类型是（　　）。

A. int　　B. float　　C. double　　D. 不确定

18. 已知字母 A 的 ASCII 码为十进制数 65，且 C2 为字符型，则执行语句“C2='A'+'6'-'3';”后 C2 中的值是（　　）。

A. D　　B. 68　　C. 不确定的值　　D. C

19. 以下用户标识符中，合法的是（　　）。

A. int　　B. nit　　C. 123　　D. a+b

20. C 语言中，要求运算对象只能为整数的运算符是（　　）。

A. %　　B. /　　C. >　　D. *

21. 若有说明语句“char c='\72';”，则变量 c 在内存占用的字节数是（　　）。

A. 1　　B. 2　　C. 3　　D. 4

22. 字符串"ABC"在内存占用的字节数是（　　）。

A. 3　　B. 4　　C. 6　　D. 8

23. 要为字符型变量 a 赋初值，下列语句中哪一个是正确的（　　）。

A. char a="3";　　B. char a='3';　　C. char a=%;　　D. char a=*;

24. 下列不正确的转义字符是（　　）。

A. \\　　B. \'　　C. 074　　D. \0

二、填空题

1. C 语言中的逻辑值“真”是用_____表示的，逻辑值“假”是用_____表示的。

2. 若 x 和 n 都是 int 型变量，且 x 的初值为 12，n 的初值为 5，则计算表达式 x%=(n%=2)后 x 的值为_____。

3. 假设 c='w'，a=1，b=2，d=−5，则表达式 'x'+1>c，'y'!=c+2，−a−5*b<=d+1，b==a=2 的值分别为_____、_____、_____、_____。

4. 假设“float x=2.5,y=4.7; int a=7;”，表达式 x+a%3*(int)(x+y)%2/4 的值为_____。

5. 判断变量 a、b 的值均不为 0 的逻辑表达式为_____。

6. 求解赋值表达式 a=(b=10)%(c=6)，表达式值，a、b、c 的值依次为_____。

7. 求解逗号表达式 x=a=3,6*a 后，表达式值，x、a 的值依次为_____。

8. 数学式 a/(b*c)的 C 语言表达式_____。

三、编程题

1. 假设长方形的高为 1.5，宽为 2.3，编程求该长方形的周长和面积。

2. 编写一个程序，将大写字母 A 转换为小写字母 a。