第 2 章

基本数据类型

本章主要介绍 C 语言的基本数据类型，常量数据的分类、表示方法和存储格式，C 语言中变量的定义和赋值，不同数据类型间的类型转换，以及 C 语言中常用运算符的功能、使用方法、结合性和优先级等。

学习目标

本章要求掌握 C 语言中常量数据的表示方法，了解基本数据类型的存储格式，掌握变量的定义和赋值，理解数据运算中类型的自动转换和强制转换，掌握常用运算符表达式值的判定，并且能在编程中熟练运用这些运算符解决问题。

本章要点

- C 语言的数据类型
- 常量数据的表示
- 变量的定义和赋值
- C 语言类型修饰符
- 表达式的数据类型转换
- C 语言的运算符和表达式

2.1 C 语言的数据类型

程序通常要对数据进行处理，程序设计语言需要支持丰富的数据类型，以满足各种编程需要。C 语言提供了丰富的数据类型，不仅可以表达并处理基本的数据(如整数、实数、字符等)，还可以组织成复杂的数据结构(如表、树、栈等)。

C 语言的数据类型如图 2.1 所示。

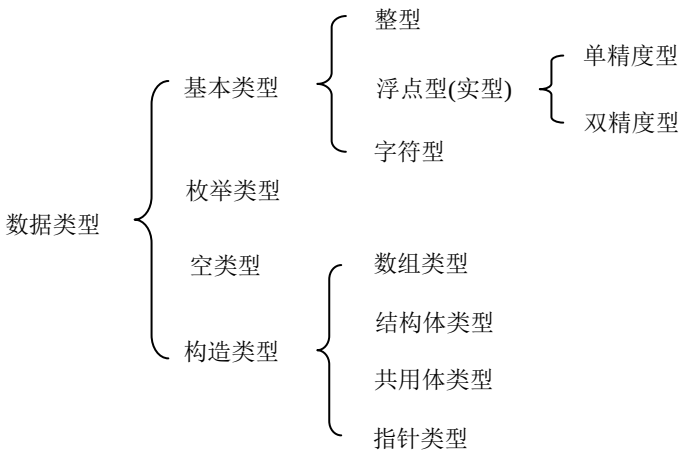

图 2.1 C 语言的数据类型

待处理的数据是被存放在内存中的，了解不同类型的数据各占用多大的内存空间及其在计算机中的存储方式是很重要的。本章主要介绍基本数据类型及其存储形式。

2.2 常量数据的表示

C 语言中的数据有两种表现形式，即**常量**和**变量**。常量的值不能被改变，是具体的值，如 2、6、135、0.23、3.14 都是常量。

常用的常量有以下几类。

1. 整型常量

整型常量可以用以下 3 种形式表示。

(1) 十进制整数，如 15、−378、0。

(2) 八进制整数。以 0 开头，如 0147 表示八进制数 147，−013 表示八进制数−13。

(3) 十六进制整数。以 0x 开头，如 0x11，代表十六进制数 11，−0x33 代表十六进制数−33。

2. 浮点型常量

浮点数也称为实数，它有两种表示形式。

(1) 十进制小数形式。这种浮点数由整数部分、小数点和小数部分组成，如 3.6、123.07、−0.22，注意小数点不能省略。

(2) 指数形式。如 123e3 代表 123×10^3。计算机输入和输出时，无法表示上角或者下角，故以字母 e 或 E 代表以 10 为底的指数，小写的 e 可以用大写字母 E 代替。注意：字母 e 之前必须有数字，e 后边的指数必须为整数，如不能写成 3e4.5。

浮点数可以用多种指数形式表示，如 43.21 可以表示为 43.21e0、4.321e1、0.4321e2 等，把其中的 4.321e1 称为规范化的指数形式，即字母 e 之前的小数部分中，小数点左边只有一位非零的数字。浮点数在用指数形式输出时，是按规范化的指数形式输出的。

3. 字符型常量

C 语言的字符型常量有两种形式，即普通字符和转义字符。

(1) 普通字符。普通字符是用单引号括起来的一个字符，如'A'、'a'、'*'、'？'、'9'都是字符常量，不能写成'ab'或者'13'。注意：单引号是界限符，字符常量只是一个字符，不包括单引号在内。

(2) 转义字符。转义字符是除了以上形式的字符常量外另一类以字符\开头的字符序列。例如，'\n'代表一个换行符，'\’'、'\”'、'\\'则分别代表单引号、双引号和反斜杠。这是一种无法显示的“控制字符”，在程序中无法用一般形式的字符来表示，只能用这样的特殊形式来显示。

常用的以“\”开头的特殊字符见表 2.1。

表 2.1　转义字符及其作用

转义字符	字 符 值	输出结果
\'	一个单引号(')	输出单引号
\"	一个双引号(")	输出双引号
\?	一个问号(?)	输出问号
\\	一个反斜杠(\)	输出反斜杠
\a	警告(alert)	产生声音或视觉信号
\b	退格(backspace)	将当前位置后退一个字符
\f	换页(form feed)	将当前位置移到下一页的开头
\n	换行	将当前位置移到下一行的开头
\r	回车(carriage return)	将当前位置移到本行的开头
\t	水平制表符	将当前位置移到下一个 Tab 位置
\v	垂直制表符	将当前位置移到下一个垂直制表对齐点
\o、\oo 或\ooo(o 为八进制数字)	与该八进制码对应的 ASCII 字符	输出与该八进制码对应的字符
\xh[h…] (h 为十六进制数字)	与该十六进制码对应的 ASCII 字符	输出与该十六进制码对应的字符

4. 字符串常量

字符串常量是一对双引号括起来的字符序列，如"China"、"A"、"$12"、"hello!"。注意：字符串常量是双引号中的全部字符，但不包括双引号本身，而且只能是双引号，不能是单引号，如'China'是错误的。字符串可以用语句输出，如 printf("China");可以输出字符串 China。

组成字符串的字符按照从左到右的顺序依次存放在一段连续的内存空间里，其中每个字符占用 1 字节的内存单元，即 8 个二进制位(或称比特)。其内容为该字符在 ASCII 码表中对应的数值，需要注意的是，C 语言的字符串在实际存储时，将自动在字符串尾部加一个结束标志'\0'(其 ASCII 码值为 0)，所以"China"占用 6 字节存放字符 C、h、i、n、a、\0，但在输出时不输出'\0'。

5. 符号常量

用#define 指令指定用一个符号名称代表一个常量。例如：

```
#define  PI  3.14                    //注意行末没有分号
```

经过上述指定后，本文件中从此行开始所有的 PI 都代表 3.14。程序进行编译之前，预处理器会对 PI 进行预处理，把所有的 PI 都置换为 3.14。在编译后，符号常量全部变成了字面上的 3.14。用一个符号名来代表一个常量，称为符号常量。

使用符号常量的好处有以下两点。

(1) 含义清楚。定义常量名的时候通常遵循“见名知意”原则，看到上面例子中的 PI，可以猜出其代表圆周率。

(2) 需要改变程序中多处用到的一个常量时，可以“一改全改”。例如，在程序中需要多处用到商品的个数，如果个数用常数 100 表示，在部分商品被售出后需要调整个数为 60 时，则需要在程序的多处进行修改，如果用符号常量 AMOUNT 表示个数，就只需要修改一处即可：

```
#define  AMOUNT  60
```

需要注意的是，符号常量不是变量，它不占用内存，只是一个临时的符号，预编译之后这个符号就不存在了，因此也不能给符号常量赋新的值。习惯上符号常量用大写字母表示，如 AMOUNT、PI 等。

2.3 变量的定义

前面介绍了 C 语言中的数据有两种基本形式，即常量和变量。在程序运行过程中，其值会发生改变的数据称为**变量**。每个变量应该有一个名字，即**变量名**。变量在内存中占用一定的存储单元，在该存储单元中存放变量的值，即**变量值**。变量名和变量值的关系如图 2.2 所示。

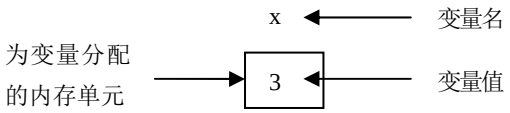

图 2.2　变量名和变量值关系

C 语言所有的变量在使用之前必须先定义，即说明变量的类型，也就是“先定义，再使用”。变量定义的形式如下：

```
类型说明符 变量名;
```

变量在定义时要注意以下几个问题。

(1) 变量的命名要符合 C 语言规定的标识符的命名规则，即只能由字母、数字和下画线 3 种字符组成，且第一个字符不能是数字。C 语言规定的有特殊意义的关键字，如 float、if、else、while、return 等，不能作为变量名。

下面列出的标识符或变量名是正确的：

value、_student1、_123、x、day、Class、total_23

下面是不合法的标识符或变量名：

123、total—23、Y.M.D、c#d、int、if、max@9

需要注意的是，C 语言是严格区分大小写的，即大写字母和小写字母被认为是两个不同的字符。因此，total 和 Total 是两个不同的变量名。通常情况下，变量名用小写字母表示，与人们日常习惯一致，以增加可读性。此外，在选择变量名和其他标识符时，应做到“见名知意”，即选择有含义的英文单词(或其缩写)作标识符，如 sum、name、day、birth、country 等。本书在一些简单的举例中，为方便起见，仍用单字符的变量名(如 x、y、a、b 等)，请读者注意不要在其他代码较长的程序中使用这些简单的变量名。

(2) 变量的数据类型决定了它的存储类型，即该变量占用的存储空间。定义变量类型，就是为了给该变量分配内存空间，以便存放数据。

基本的变量类型及其存储空间见表 2.2。

表 2.2　基本变量类型及其存储空间

类　型	名　称	存储空间	取值范围	举　例
int	整型	4 字节	介于-2^{31}～$2^{31}-1$ 的整数	int i
float	单精度浮点型	4 字节	实数，有效位数 6～7 位	float x
double	双精度浮点型	8 字节	实数，有效位数 15～16 位	double y
char	字符型	1 字节	ASCII 码字符，−128～127 的整数	char c

说明：

- 表 2.2 中列出 C99 标准定义的 int 型数据取值范围是 4 字节，VC++6.0 编译系统就使用该标准。有的 C 编译系统规定一个整型数据占 2 字节。
- 浮点型数据在内存中是按照指数形式存储的，小数部分和指数部分分别存放，在

4 或 8 字节中，究竟多少位表示小数部分，多少位表示整数部分，由各 C 编译系统自定。一般而言，小数部分位数多，数据表示的有效数字多，精度就高；而指数部分位数多，则表示的数据范围更大。VC++6.0 编译系统中，单精度浮点数默认保留 6 位有效数字，双精度浮点数默认保留 15 位有效数字。

- char 型数据在内存中存储的形式为该字符对应的 ASCII 值，即一个整数。这样实现了整型数据和字符型数据之间的转换。一个 char 型变量只能存放一个字符，汉字或字符串的存储需要用字符数组实现。

变量的类型决定了它可以存放的数据范围，所以在定义变量之前，一定要先明确待处理数据的特征和范围，再确定适合的变量类型以便存放数据。

2.4 变量的赋值

定义变量后，系统会自动根据变量的类型分配内存空间。需要对一些变量预先设置初始值，即赋值。赋值操作通过赋值符号“=”把右边的值赋给左边的变量。例如：

```
int a=2;                /* 指定 a 为整型变量，初值为 2 */
```

相当于：

```
int a;                  /* 指定 a 为整型变量 */
a=2;                    /* 赋值语句，将 2 赋给 a */
```

变量在定义的同时可以赋初值，称为**变量的初始化**。

又如：

```
char ch='A'; float total=1.2;
int y; y=2*3-4;
int i=1; i=i+1;
```

说明：

(1) C 语言中的赋值符号“ =”不同于数学中的“ =” 。在 C 语言中，i=i+1 是成立的，它表示将变量 i 的值加 1 后再赋给变量 i。在 C 语言中，判断两个数是否相等时使用符号“ ==” ，它是一个关系运算符，如“ a==b” 这个式子的值要么为 0 要么为 1，将在本章“ 运算符与表达式” 一节中详细介绍。

(2) 如果赋值时等号两侧类型不一致，系统将会做以下处理。

① 将一个浮点数赋给一个整型变量时，如 int a=3.5; 系统自动舍弃小数部分，变量 a 被赋值为 3。可以理解为变量 a 中只能存放整数，3.5 要想存放到 a 中就只能把整数部分的 3 存放进去。

② 将一个整数赋给一个浮点型变量时，如 float f=12; 系统将保持数值不变，以浮点小数形式存储在变量中，此时 f=12.000000。可以理解为变量 f 中只能存放带小数点的浮点数，12 要想存放到 f 中就只能加小数点并补 0。

③　将一个字符赋给一个整型变量时，如 int x='a'; 不同的编译系统实现的情况不同，一般当该字符的 ASCII 值小于 127 时，系统将整型变量的高字节置 0，低字节存放该字符的 ASCII 值，此时变量 x 被赋值为字符 a 的 ASCII 值 97，即变量 x 中存放的是整数 97。

(3)　可以将字符型数据、介于−128～127 的整数或者转义字符赋给字符型变量。

众所周知，计算机存储的是二进制数，所以将一个字符数据存放在一个字符变量中，实际存放的是该字符的 ASCII 码的二进制形式。大写字母 A 的 ASCII 码十进制表示是 65、二进制表示是 01000001、八进制表示是 101、十六进制表示是 41，即这 4 种数据在计算机中的存储形式相同，赋给一个字符型变量的结果也相同。

例 2.1　将字符'A'赋值给字符变量的 4 种方法。

【代码】

```
#include <stdio.h>
int main()
{
   char c1,c2,c3,c4;
   c1='A';
   c2=65;
   c3='\101';
   c4='\x41';
   printf("%c, %c, %c, %c\n",c1, c2, c3, c4);
   printf("%d, %d, %d, %d\n",c1, c2, c3, c4);
   return 0;
}
```

【运行结果】

```
A, A, A, A
65, 65, 65, 65
```

说明：

- 转义字符'\101'表示八进制数 101，其对应的十进制形式为 65。
- 转义字符'\x41'表示十六进制数 41，其对应的十进制形式为 65。
- "%c"表示输出类型为字符形式，系统将存储的二进制数按照 ASCII 码表转化成相应的字符，然后输出。
- "%d"表示输出类型为十进制整数，系统直接将二进制数转换成十进制整数输出。

例 2.2　变量赋值示例。

【代码】

```
#include <stdio.h>
int main()
{
   int x,y;
   float z;
```

```
    x=3.4;
    y='a';
    z=12;
    printf("x = %d, y = %d, y = %c, z = %f\n",x,y,y,z);
    return 0;
}
```

【运行结果】

```
x = 3, y = 97, y = a, z = 12.000000
```

说明：

- 浮点数 3.4 赋给整型变量 x，系统舍弃小数部分保留整数部分。
- 小写字母 a 的 ASCII 码值为 97，小于 127，以十进制整数形式输出时为 97，以字符形式输出时为 a。
- 整数 12 赋给实型变量 z，转换为等值的浮点小数形式。

2.5 C 语言的类型修饰符

以上介绍的基本数据类型可以带修饰性前缀，即类型修饰符，用来适应更多的数据处理需要，可扩大 C 语言基本数据类型的适用范围。C 语言共有 4 种类型修饰符：long——长型；short——短型；signed——有符号型；unsigned——无符号型。具体使用情况和存储空间见表 2.3。

表 2.3 带类型修饰符的数据类型

类 型	存储空间	取值范围
[signed] int	4 字节	−2147483648～2147483647，即 -2^{31}～$(2^{31}-1)$
unsigned int	4 字节	0～4294967295，即 0～$(2^{32}-1)$
long [int]	4 字节	−2147483648～2147483647，即 -2^{31}～$(2^{31}-1)$
unsigned long [int]	4 字节	0～$(2^{32}-1)$，最高位不作为符号位
short	2 字节	−32768～32767，即$(-2^{15}$～$2^{15}-1)$
unsigned short	2 字节	0～65535，即$(0$～$2^{16}-1)$
long double	8 字节	-1.7×10^{-308}～1.7×10^{308}，取值范围同 double 型
[signed] char	1 字节	−128～127，即-2^{7}～$(2^{7}-1)$
unsigned char	1 字节	0～255 ，即 0～$(2^{8}-1)$，最高位不作为符号位

说明：

- 表 2.3 中第一列的数据类型，方括号内的可以省略，如 long [int]可简写为 long。
- short 型不常用，对于不同机型取值范围不同，本书中为 VC++6.0 编译系统的取值。
- long double 根据编译系统的不同分配的存储空间大小不同，VC++6.0 编译系统将

long double 和 double 同样处理，分配 8 字节空间。而 Turbo C 编译系统为 long double 分配 16 字节的空间。注意：在 C 语言中进行浮点数的算术运算时，将 float 型数据都自动转换为 double 型，然后再进行运算。

- 基本类型前加 signed 和前边介绍的基本类型等价。例如，[signed] int 和 int 等价，[signed] char 和 char 等价，一般都不写 signed。

例 2.3　类型修饰符的使用示例。

【代码】

```
#include <stdio.h>
int main()
{
    char a,b;
    unsigned char a1,b1;
    int x,y;
    long x1,y1;
    a=127; b=129;
    a1=127; b1=129;
    x=2147483647; y=2147483649;
    x1=2147483647L; y1=2147483649L;
    printf("a = %d, a1 = %u, b = %d, b1 = %u\n",a, a1, b, b1);
    printf("x = %d, x1 = %ld, y = %d, y1 = %ld\n",x, x1, y, y1);
    return 0;
}
```

【运行结果】

```
a = 127, a1 = 127, b = -127 , b1 = 129
x = 2147483647, x1 = 2147483647, y = -2147483647, y1 = -2147483647
```

说明：

- char 型的取值范围是−128～127，在此范围内，char 和 unsigned char 输出结果一致，即 a 和 a1 输出都是 127。129 超出 char 的取值范围，在 unsigned char 的范围内，而 129 和−127 在计算机中存储内容相同，都为二进制 10000001。当最高位 1 作为符号位表示负数，则 char 型变量 b 输出显示−127。当最高位 1 作为无符号数，代表 129，则 unsigned char 型变量 b1 输出显示 129。
- int 型和 long 的取值范围相同，是−2147483648~2147483647，在此范围内，int 和 long 输出结果一致，即 x 和 x1 输出都是 2147483647。2147483649 超出 int 和 long 的取值范围，当 2147483649 放入 int 型和 long 型变量中，实际占用 4 字节，共 32 个二进制位，最高位为 1 是负数，则 y 和 y1 输出显示−2147483647，已经溢出。
- “ %u” 输出格式符，表示无符号的十进制形式的整数。
- 将整数常量赋值给 long 型变量，在整数常量后加大写字母 L 或小写字母 l。“ %ld” 表示输出 long 型数据。注意：是字母 l 和字母 d，而不是数字 1 和字母 d。

2.6 表达式的数据类型转换

不同的数据类型数据可以进行混合运算，如 1+'a'−2.3、4.5*6/7 等，因为各种数据类型的取值范围不同，需要弄清楚不同数据类型的混合运算结果(表达式的值)是什么类型。

2.6.1 自动类型转换

在程序中常会遇到不同类型的数据进行混合运算，不同类型的数据在参加运算前会自动转换为相同的类型，再进行运算。需要注意的是，这种转换是编译系统自动完成的，不需要用户参与，转换的规则如图 2.3 所示。

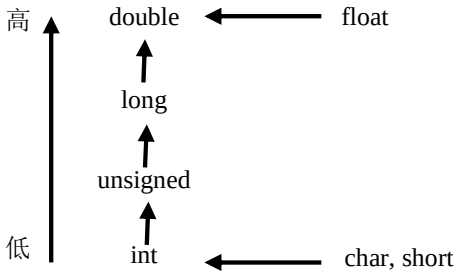

图 2.3 自动类型转换规则

图 2.3 中，横向向左的箭头表示必定的转换。例如，char 型和 int 型进行混合运算，char 型先转换成 int 型再运算，结果为 int 型；float 型数据在运算时一律先转换为 double 型，再参与运算。

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 型和 double 型数据进行运算，先将 int 型转换为 double 型再进行运算，结果为 double 型。

例 2.4 自动类型转换示例。

【代码】

```
#include <stdio.h>
int main()
{
    float a,b;
    float pi=3.1416;
    int s, r=4;
    a=5/2-4/3;
    b=3.0/2+5/3.0;
    s=pi*r*r;
    printf("s = %d\n",s);
    printf("a = %f, b = %f\n",a,b);
    return 0;
}
```

【运行结果】

```
s = 50
a = 1.000000, b = 3.166667
```

说明：

- 整数除整数结果为整数，所以 5/2−4/3 的整数结果赋给浮点变量 a，以浮点小数形式存储。
- 浮点数和整数的运算结果为浮点数，所以 3.0/2+5/3.0 的浮点小数结果赋给浮点型变量 b，以浮点小数形式存储。
- pi 为实型，s、r 为整型。在执行 s=pi*r*r 语句时，pi 和 r 都转换成 double 型计算，结果也为 double 型。但由于最终结果要赋给整型变量 s，故将 double 型的结果舍去了小数部分，赋给整型变量 s。

2.6.2　强制类型转换

可以通过强制类型转换运算符来将一个表达式转换成所需要的类型。其一般形式为：

```
(类型说明符)表达式
```

作用是把表达式的运算结果强制转换成类型说明符所表示的类型。

例如：

(int)7.4%3 表示先把 7.4 转化成整数 7，再对 3 取余。

(int)(x+y)表示把 x+y 的结果转换为整型。

(int)2.3*12 是把 2.3 转换成整型再与 12 相乘。

int x=5,y=2; 则 x/y 结果为 2(注意，整数除整数的结果仍然是整数)，而(float)x/y 是先将 x 强制转换为浮点型 5.000000，然后再除以 2，浮点数除以整数结果为浮点数 2.500000。

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，原来变量的类型并未发生变化，如上例(float)x/y，(float)x 的值为 5.000000，x 的类型仍然为整型，值为 5。

2.7　C 运算符和表达式

运算符是说明各种不同计算法则的符号。C 语言提供了丰富的运算符，如算术运算符、关系运算符、逻辑运算符、赋值运算符等。参加运算的数据和运算符连接起来就构成了运算表达式，即表达式。表达式是由各类运算符将常量或变量数据连接起来，因此需要了解各类运算符的功能、结合性和优先级别。

本节将介绍 C 语言各种运算符的含义、功能、结合方向和优先级别。

2.7.1 算术运算符和算术表达式

1. 基本的算术运算符

正号运算符“+”：正号运算符为单目运算符，即只有一个量参与运算，如+1、+a。

负号运算符“-”：负号运算符为单目运算符，如-5.2、-b。

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、3+2。

减法运算符“-”：减法运算符为双目运算符，如 5-2。

乘法运算符“*”：乘法运算符为双目运算符，如 2*5。

除法运算符“/”：除法运算符为双目运算符，如 7/2。

模运算符(或称取余运算符“%”)：为双目运算符，如 7%3。结果等于两数相除后的余数，如 7%3 结果是 1 而不是 2。需要注意，要求“%”两侧均为整型数据。

注意： 要特别留意除法运算和模运算。整数除以整数结果仍然为整数，浮点数(单精度和双精度)除以整数结果为浮点数。例如，5/3 的结果为 1，舍去小数部分，而 5.0/3 的结果为 1.666667。另外，模运算符要求运算的两侧必须是整型数据，如 7%3 的结果是 1。如果不是整型的数据做模运算，可以采用强制类型转换。例如，变量 a 为 float 型，对 3 取余，可表示为(int)a%3。

2. 自增运算符和自减运算符

自增、自减运算符主要用于给一个变量加 1 或减 1。运算符及其功能如下。

自增运算符“++”：如 i++，++i，等同于 i=i+1。

自减运算符“--”：如 i--，--i，等同于 i=i-1。

自增、自减运算符是单目运算符，“++”“--”可以放在变量前边或者后边。这两种方式都完成了变量的自增或自减，i++和++i 的值是一样的。但当变量的自增运算或是自减运算同其他运算配合构成一个表达式时，在前、在后的不同位置使该变量在参加运算时的值是不同的，i++表示先使用 i 而后 i 再自增 1，而++i 表示 i 先自增 1 再使用 i。

例如：

当 n=5 时，执行语句 i=n++(++在后)，n 的值先赋值给 i，i=5，n 再自增 1，n=6；而执行语句 i=++n(++在前)，n 先自增 1，n=6，再赋值给 i，i=6。

因此自增、自减运算符放在变量之前时，表示变量先变化，后使用；放在变量之后时，表示变量先使用，后变化。

注意： 自增和自减运算符只能用于变量，而不能用于常量或表达式，如 3++或者(x+y)--都是不合法的。按照自增的定义，3++可以等同于 3=3+1，明显不对。而(x+y)--这个式子中 x+y 的结果要是一个常量的话，结果也是不成立的。

自增或自减运算符常被用在循环语句中或者指针变量上，这些将在后续的章节中进行介绍。

3. 算术表达式

用算术运算符将数据对象连接起来的式子称为算术表达式。表达式的运算按照运算符的优先级和结合性来进行。

C 语言规定了运算符的**优先级**和**结合性**。在表达式求值时，先按算术运算符的优先级别高低次序执行。规定负号运算符(“-”)优先级高于乘、除、模运算符，乘、除、模运算符优先级高于加、减运算符。如表达式 a-b*c，乘号优先于减号，因此相当于 a-(b*c)。如果在一个运算对象两侧的运算符优先级别相同，如表达式 a-b+c，减号和加号优先级别相同，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)，算术运算符遵循“左结合性”，即从左往右算。例如，a-b+c，先计算 a-b，再+c。有的运算的结合方向为“从右至左”，称为“右结合性”，即从右往左算。C 语言运算符中有不少为右结合性，应注意区别。

例 2.5　计算并编程验证 x-3*-7/(int)(1.3+y)+2 的结果，其中 x=1.4，y=3.1。

【代码】

```
#include <stdio.h>
int main()
{
    float x=1.4,y=3.1;
    printf("%f", x-3*-7/(int)(1.3+y)+2);
    return 0;
}
```

【运行结果】

```
8.400000
```

2.7.2　关系运算符和关系表达式

1. 关系运算符

关系运算是进行比较运算的，比较两个数据是否符合某个给定的条件。C 语言有以下 6 种关系运算符。

<：小于运算符，如 a<6。

<=：小于等于运算符，如 0<=3。

>：大于运算符，如 a>b。

>=：大于等于运算符，如 a>=5。

==：等于运算符，如 x==(y+1)。

!=：不等于运算符，如 z!=0。

两个数据在进行值的比较时，其结果只有两个，要么成立要么不成立，成立则表达式

的值为“真”，不成立则表达式的值为“假”。在 C 语言中，任何非 0 值为“真”，0 值为“假”。因此，关系运算的结果仅可能产生两个值：1 表示“真”；0 表示“假”。

2. 关系运算符的优先级和结合性

关系运算符中，<、<=、>、>=的优先级相同，==、!=的优先级相同，前 4 种的优先级高于后两种。例如，3!=1>3，“>”优先级高于“!=”，1>3 结果为 0，3!=0 结果为 1，所以这个式子的值为 1。

关系运算符优先级小于算术运算符。例如，4>2−1,“−”优先级高于“>”，所以先计算 2−1，结果为 1，再计算 4>1，最终结果为 1。

关系运算符的结合性为左结合性，即从左至右。例如：3>2>1，先计算 3>2，成立则结果为 1，再计算 1>1，关系不成立，最终结果为 0。这个式子的最终结果为假，即 0。注意：这与我们的常识是不一样的，表面上看 3>2>1 是成立的，结果应该是 1，但是关系运算符是双目运算符，只能两两比较，不能一次比较 3 个以上的数。

3. 关系表达式

关系表达式是用关系运算符连接起来的式子。关系表达式的值是一个逻辑值，即“真”或者“假”。成立就是“真”，值为 1；不成立就是“假”，值为 0。

例 2.6 关系运算符演示示例。

【代码】

```
#include <stdio.h>
int main()
{
    int a=4,b=3,c=1,result=2;
    printf("%d\n",a<=b);
    printf("%d\n",a==b+c);
    result=a>b>c;
    printf("%d\n",result);
    return 0;
}
```

【运行结果】

```
0
1
0
```

2.7.3 逻辑运算符和逻辑表达式

1. 逻辑运算符

逻辑运算表示两个数据或表达式之间的逻辑关系。C 语言提供的逻辑运算符有以下 3 种。

(1) 逻辑与运算符“&&”。这是双目运算符，要求有两个运算对象，可以表示成“条件 1&&条件 2”。只有当条件 1 成立并且条件 2 也成立时，逻辑与成立，结果为真，

即值为 1，其余情况结果都为假，即值为 0。所以条件 1 和条件 2 只要有一个不成立，逻辑与的结果都为 0。例如，(2>4)&&(3>2)，由于 2>4 不成立，即使 3>2 成立，结果仍为 0。

(2) 逻辑或运算符“||”。这是双目运算符，要求有两个运算对象，同样可以表示成“条件 1||条件 2”。当条件 1 成立或者条件 2 成立时，逻辑或成立，结果为真，即值为 1，其余情况结果为假，即值为 0。所以条件 1 和条件 2 只要有一个成立，逻辑或的结果就为 1。只有当条件 1 和条件 2 都不成立时，逻辑或的结果才为假，即值为 0。例如，(x>4)||(x<2)，只要 x 取值大于 4 或者小于 2 这两个条件满足一个时结果为 1；否则结果为 0。

(3) 逻辑非运算符“!”。这是单目运算符，可以表示成“!条件”。当条件成立时，逻辑非的运算结果为假，即值为 0；反之当条件不成立时，运算结果为真，即值为 1。例如，!(1+2)，1+2 结果非 0，再进行逻辑非运算，结果为 0。非运算可以理解成取反，任何非 0 的值非运算之后是 0，0 值非运算之后是 1。

注意： 两个条件进行与运算，只有两个条件都为真，结果才为真，即值是 1。两个条件进行或运算，只要有一个条件为真，结果就为真，即值是 1。一个条件进行非运算，就是取反，非 0 值进行非运算结果是 0，0 进行非运算结果是 1。

2. 逻辑运算符优先级和结合性

逻辑运算符的优先次序如下。

(1) !→&&→||。

(2) 和其他运算符相比，“!”高于算术运算符，“&&”和“||”低于关系运算符，如图 2.4 所示。

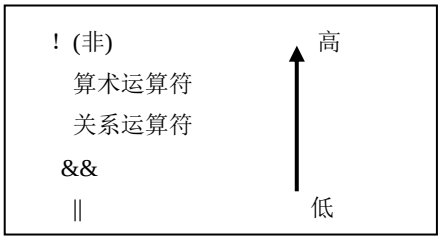

图 2.4　逻辑运算符优先级

注意： 逻辑运算符“!”的结合性为右结合性，即从右往左算，“&&”“||”的结合性为左结合性，即从左往右算。

例 2.7　逻辑运算符演示示例。

【代码】

```
#include <stdio.h>
int main()
{
    int x=9,y=5,z=4;
    printf("%d\n",!2-1||2<x&&3!=y);
```

```
    printf("%d\n",(x+y)&&!z-4&&4%z);
    return 0;
}
```

【运行结果】

```
1
0
```

实际上，逻辑运算符两侧的运算对象不但可以是 0 或非 0 的整数，也可以是任何类型的数据，如字符型、实型或是指针型等。系统最终以 0 或非 0 来判断它们属于“真”还是“假”。例如，'a'&&'b'的值为 1，是因为'a'和'b'的 ASCII 值都不为 0，两个非 0 的数据进行与运算，结果为 1。

注意： 在逻辑表达式的求解中，并不是所有的逻辑运算符都被执行，只是在必须执行下一个逻辑运算符才能求出表达式的值时，才执行该运算符。

举例如下。

(1) a&&b&&c。只有 a 为真(非 0)时，才需要判断 b 的值，只有 a 和 b 都为真时，才需要判断 c 的值。也就是说，只要 a 为假，整个表达式即为假，没必要再判断 b 和 c；只要 a 为真，但 b 为假，就没必要再判断 c。

(2) a||b||c。只要 a 为真(非 0)，整个表达式即为真，就不必判断 b 和 c；如果 a 为假，才判断 b，如果 b 也为假，才需要判断 c。

例 2.8 阅读程序，写出运行结果。

【代码】

```
#include <stdio.h>
int main()
{
    int x=8,y=3;
    printf("%d\n",(x=2>3)&&(y=7));
    printf("x=%d,y=%d",x,y);
    return 0;
}
```

【运行结果】

```
0
x=0,y=3
```

2.7.4 赋值运算符和赋值表达式

1. 基本赋值运算符

C 语言中最常用的赋值运算符是“=”，其作用是将赋值运算符右边的表达式的值赋给左边的变量，如 a=7 是将 7 赋给变量 a。

赋值运算符的优先级比算术运算符、关系运算符和逻辑运算符都要低。赋值运算符的

结合性是右结合性，即从右往左算。

2. 复合赋值运算符

在赋值运算符“=”之前加上其他运算符，如+、−、*、/、%其中的一种，形成复合赋值运算符。如下所示。

加赋值运算符“+=”：如 a+=(1+3)，等价于 a=a+(1+3)。

减赋值运算符“−=”：如 a−=(1+3)，等价于 a=a−(1+3)。

乘赋值运算符“*=”：如 a*=(1+3)，等价于 a=a*(1+3)。

除赋值运算符“/=”：如 a/=(1+3)，等价于 a=a/(1+3)。

取余赋值运算符“%=”：如 a%=(1+3)，等价于 a=a%(1+3)。

复合赋值运算符的作用是先将复合运算符右边表达式的结果与左边的变量进行算术运算，然后再将运算结果赋给左边的变量。

注意： 复合运算符左边一定是变量；复合运算符右边的表达式计算完成后才参与复合赋值运算。使用这种复合运算符，可以简化程序，书写方便，并且能提高编译效率。

复合赋值运算符的优先级和结合性等同于简单的赋值运算符“=”。

例 2.9　赋值运算符演示示例。

【代码】

```
#include <stdio.h>
int main()
{
    int x=5,y=4;
    printf("%d\n",x+=x-=x*x);
    printf("%d\n",y+=y-=y*=y);
    return 0;
}
```

【运行结果】

```
-40
0
```

注意：

- x+=x−=x*x 运算步骤：x*x 结果为 25，x=x−25 结果为 x=−20，x=x+x 结果为−40。
- y+=y−=y*=y 运算步骤：y=y*y 结果为 y=16，y=y−y 结果为 y=0，y=y+y 结果为 y=0。

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。

例如，“a=3”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。注意：赋值表达式的值就是被赋值的变量的值。例如，赋值表达式“a=3”，变量 a 被赋值 3，则表达式 a=3 的值为 3。又如，赋值表达式 a=3+(b=5)，b 值为 5，(b=5)这个式子的值为 5，a 被赋值 8，则整个赋值表达式的值为 8。

赋值表达式可以出现在其他语句中，如输出语句中，这样就实现了在一条语句中同时完成赋值和输出双重功能。例如，printf("%d\n",y=5);作用是 y 被赋值 5，并输出表达式 y=5 的值 5。

2.7.5 逗号运算符和逗号表达式

C 语言提供了一种特殊的运算符用于连接表达式，这种运算符就是逗号运算符。如：

```
1+a,b=5/2
```

用逗号运算符连接的表达式称为逗号表达式。它的一般形式为：

```
表达式 1,表达式 2,…,表达式 n
```

逗号表达式的运算过程是先算表达式 1，再算表达式 2，依次算到表达式 n。例如，上面的举例就是先算 1+a，再算 b=5/2。整个表达式的值是最后一个表达式的值，如上面举例的逗号表达式值为 2。

逗号运算符的优先级别最低，结合性为左结合性，从左往右算。

例 2.10 已知 x 初值为 1，求表达式 x=3,x*5 值为多少？

注意： 该表达式中有逗号运算符、赋值运算符、算术运算符，因为三者中逗号运算符优先级最低，算术运算符最高，所以该表达式是一个逗号表达式，先执行表达式 x=3，则 x 从初值 1 变为 3，再执行 x*5，即 3*5，则整个逗号表达式值为 15。

2.7.6 条件运算符和条件表达式

条件运算符是 C 语言中唯一的三目运算符，它需要 3 个运算数或表达式构成条件表达式。它的一般形式为：

```
表达式 1? 表达式 2:表达式 3
```

执行顺序是：先求解表达式 1，如果成立则整个条件表达式的值就是表达式 2 的值；否则整个表达式的值就是表达式 3 的值。

例如，表达式“max=((a>b)?a:b)”，先判断 a>b 是否成立，成立的话表达式(a>b)?a:b 的值就是 a，不成立的话表达式的值就是 b。执行结果就是最后将条件表达式的值赋给 max，也就是将 a 和 b 中的较大值赋给 max。

条件运算符的优先级别仅高于赋值运算符和逗号运算符，低于关系运算符、算术运算符和逻辑运算符。因此，表达式“max=((a>b)?a:b)”可以写成“max=a>b?a:b”。

条件表达式的结合方向为从右向左。例如：

```
a>b?a:c>d?c:d
```

等价于

```
a>b?a:(c>d?c:d)
```

注意： 条件表达式使用起来很方便、简洁，一个表达式中既有条件又有结果。例如，条件表达式(a>b)?a:b 中，问号前边的 a>b 是判断条件，冒号左右两个式子中必定有一个是整个条件表达式的结果。

例 2.11　求 3 个数中的最大数。

【代码】

```
#include <stdio.h>
int main()
{
    int x,y,z,max;
    scanf("%d%d%d",&x,&y,&z);
    max=x>(y>z?y:z)?x:(y>z?y:z);
    printf("%d,%d,%d,max=%d\n",x,y,z,max);
    return 0;
}
```

【运行结果】

```
1 5 3↙
1,5,3,max=5
```

本节介绍了 C 语言的常用运算符，将这些运算符按优先级别从高到低排序，如图 2.5 所示。

()、！、++、--、负号运算符-	高
算术运算符 *、/、%	
算术运算符 +、-	
关系运算符 <、<=、>、>=	
关系运算符 ==、!=	
逻辑运算符 &&	
逻辑运算符 ||	
条件运算符 ?:	
赋值运算符 =、*=、/=、*=、%=、+=、-=	
逗号运算符 ,	低

图 2.5　常用运算符优先级

习 题 2

一、单项选择题

1．C 语言中编译环境决定了不同类型的变量占用的内存单元大小，VC++6.0 环境下，int 型数据占用的字节数为(　　)。

A．1　　B．2　　C．4　　D．8

2．在 C 语言中，要求参加运算的数必须是整数运算符的是(　　)。

A．/　　B．*　　C．%　　D．=

3．假定 x 和 y 为 int 型，则表达式 x=2，y=x+3/2 的值是(　　)。

A．3.500000　　B．3　　C．2.000000　　D．3.000000

二、判断题

1．算术运算符的优先级高于逻辑运算符。　　(　　)

2．已知 float x=3.5;，则(int)x 强制类型转换后 x 的值变为 3。　　(　　)

3．逗号表达式是从左向右运算的，整个表达式的值是最后一个表达式的值。　　(　　)

三、编程题

1．已知 a=1,b=2,c=3.5，计算(float)(a+b)/3+(int)c 的值。

2．从键盘输入一个小写字母，输出其对应的大写字母。

3．用条件运算符实现：输入一个英文字母，如输入大写字母，输出其对应的小写形式，如输入小写字母，则原样输出。

4．从键盘输入 3 个整数，按照从小到大的顺序输出。