2.3 数据运算

数据运算由数据和运算符组成，C语言的内部运算符很丰富，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语言有三大运算符：算术、关系与逻辑、位操作。

另外，C还有一些特殊的运算符，用于完成一些特殊的任务。

1

在C 语言中，提供了以下各种运算：

赋值运算
算术运算
关系运算
逻辑运算
条件运算
求字节数运算
逗号运算
混合运算

2

2.3.1 赋值运算

最简单形式： 变量 = 表达式

其功能是将一个表达式的值赋给变量。

其中，"="称为赋值运算符

以下赋值表达式是正确的：

```
a = a + 1;
```

该式读作将表达式a+1的值赋给a。其本意是改写变量a的值，而不是判断a+1与a是否相等.

```
k = j + 3 * ( k + j );
```

以下表达式是非法的：

```
3 = j (j + k) = 5;
```

3

赋值运算的执行过程：

(1)计算表达式

(2)如表达式类型与变量类型不一致，将表达式值的类型转换成变量的类型。

(3)将值赋给变量。

例1: int x; double y;

```
x = y = 3.5;
```

结果： y 的值为3.5 ， x的值为3

4

连续赋值

在C语言中，赋值运算符的级别较低，并满足右结合规则。因此表达式： x=y=z=1

是连续赋值表达式，其功能相当于如下表达式的功能

x= (y= (z=1))

它是先执行表达式z=1，即将1赋给z，表达式值也为1，然后将表达式值1赋给y，即执行y=1，表达式值也仍为1，再将表达式值1赋给x，即执行x=1。

例2: i=4+(j=7)，使j值为7，i值为11

例3: i=(j=3)+(k=8)，使j值为3，k为8，i为11

说明：赋值运算符的结合性是"自右至左"。

5

复合赋值运算

在赋值运算符"="之前加上其他运算符，可构成复合赋值运算符。

复合赋值运算符：

+=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=

例:x + = 5.0 → 等效于 x = x + 5.0

x * = u + v → 等效于 x = x * (u + v)

a + = a-=b+2 → 等效于 a= a+(a=a-(b+2))

6

记θ为某个双目运算符，复合赋值运算

x θ= e

的等效表达式为

x = x θ (e)

当e是一个复杂表达式时，等效表达式的括号是必需的。

赋值运算符和所有复合赋值运算符的优先级全部相同，并且都是"自右至左"结合，它们的优先级高于逗号运算符，低于其它所有运算符。

7

复合赋值运算示例

```
#include <stdio.h>
void  main()
{
    int a=12;
    a += a -= a * a;
printf("a=%d\n", a);
}
```

运行过程分析：

step1：计算括号内的赋值表达式

a = a – a * a;

即：a = 12 – 12 * 12 = -132

step2：计算括号外的赋值表达式

a = a + a;

即：a = -132 + (-132) = -264

8

2.3.2算术运算

C语言中提供的算术运算包括：

+(取正)、-(取负)

+(加)、-(减)、*(乘)、/(除)、%(求余)、

++(自增)、--(自减)

9

算术运算符及其功能

- +(加)、-(减)、*(乘)运算与数学运算的习惯相同。
- +(取正)和-(取负)运算与数学运算的习惯相同。
- /(除)除法运算：整除运算

需要注意的是：两个整数相除结果也是整数，而且将会把除不尽的小数部分舍去。

例如：7/4 的结果为1，1/2 的结果为0。

1/2.的结果为0.5

10

- (%)：求余运算

求余运算又可称为求模运算。

求余运算符(%)要求参与运算的两个运算分量(操作数)均为整型数据。

如： 5 % 3 的值为 2。 5.5%2 (×)

一般来说，求余运算所得结果的符号与被除数的符号相同。

如： -5 % 3 = -2 ， 5 % -3 = 2 。

取正(+)、取负(-)是单目运算符，结合性是从右至左，优先级高于 +、-、*、/、% 等双目运算符。

11

- ++(自增)运算和--(自减)运算

C语言中有两个很有用的运算符，通常在其它计算机语言中是找不到它们的—自增和自减运算符， + + 和- -。运算符“ ++”是操作数加1，而“--”是操作数减1，

1) 前缀++和后缀++(++变量，变量++)

++变量运算规则：先使变量值增加1个单位，再引用该变量，即以增1后的变量值为结果。

变量++运算规则：先引用变量，然后使该变量增加1个单位。

单独使用时，++i和i++都等价于i = i + 1

在和其它运算符混合运算时，效果不同，

12

例如：j＝++i; 表示先将i加1，再取i赋给j。
即等价于 i＝i＋1； j＝i;

j = i++; 表示先取i赋给j，再将i加1。

即等价于

```
j = i;
i = i + 1;
```

例如：i = 5; j = i++; 则i的值为6，j的值为5。

而对于i = 5; j = ++i; 则i的值为6，j的值为6。

13

2）前缀--和后缀--

++变量运算规则：先使变量值减少1个单位，再引用该变量.即以减1后的变量值为结果。

变量--运算规则：先引用变量，然后使该变量减1个单位。

单独使用时，--i和i--都等价于i = i - 1

在和其它运算符混合运算时，效果不同，例如：

j = --i; 表示先将i减1，再取i赋给j。

即等价于 i = i - 1; j = i;

14

j = i--; 表示先取i赋给j，再将i减1。即等价于

```
j = i;
i = i - 1;
```

例如：i = 5; j = i--; 则i的值为4，j的值为5。

而对于i = 5; j = --i; 则i的值为4，j的值为4。

```
例 -i++;
i=3;  printf("%d\n",-i++);
```

15

3） 单目运算的性质

++(自增运算符)和-- ，表示只能对一个运算变量(运算操作符)进行运算，使变量的值增1或减1。而不能对常量或者一个表达式进行自增或自减运算。

例如以下运算是非法的：

3++, --5, ++(i+j), (k*5)--

16

与加法运算符+以及减法运算符-相比，++和--的运算优先级要高。

也就是说，对于i+++j，将被理解为

(i++) + j，而不是 i + (++j)。

对于这种可能产生误解的操作，希望在编程时加括号，得到明确无误的表达以避免产生错误。

17

//自增和自减运算程序示例二

```
#include <stdio.h>
void main()
{
    int i, j, k, m;
    i = j = k = 2;
    m = (i++) + (j++) + (k++);
    printf("i=%d, j=%d, k=%d, m=%d\n", i, j, k, m);
    i = j = k = 2;
    m = (i++) + (++j) + (++k);
    printf("i=%d, j=%d, k=%d, m=%d\n", i, j, k, m);
}
```

运行结果：

```
i=3, j=3, k=3, m=6
i=3, j=3, k=3, m=8
```

18

4) 自增和自减的多次运算

自增和自减是带有副作用的运算符，如果在一个表达式中对同一个变量多次使用++或者--运算，在不同的计算机系统中可能产生不同的结果。

❖特别提示：慎用++,--运算符！！！

例：
```
int a=3;  printf("%d,%d\n",a,a++);
printf("%d\n",a);
a=3;  printf("%d,%d,%d,%d\n",++a,a++,a++,--a);
```

TC	4,3 4 5,3,2,2
VC	3,3 4 3,2,2,2

从例中可分析出：
1、函数参数均按从右到左的顺序求值；
2、对于后置运算，都遵循"先用后加"规则，但"加"的时机不同，TC中是用后马上加，这样下个左边的参数用到的值是加后的值，而在VC中，用后不马上加，而是在函数调用（printf）完成后才加。

因此，以上写法不宜提倡，最好改写成：

a=3; b=a++;

或：a=3; b=a; a++;

2.3.3关系运算和逻辑运算

1.关系运算

关系运算符：<、<=、>、>=、==、!=

关系运算符用于对两个值进行关系比较，判定比较条件是否满足。

关系运算表达式的形式为：$a\ \theta\ b$

其中，θ表示以上6个关系运算符中的某一个。运算分量a和b可以是算术常量、变量或者表达式。

a θ b的含义是判定对a和b的比较（条件）是否成立。如果条件成立，则比较的结果为逻辑值"真(TRUE)"，否则结果为逻辑值"假(FALSE)"。

在C语言中，约定非0表示逻辑值"真"；0表示逻辑值"假"。

如果a θ b成立，得到结果为1(TRUE)。

否则，表示a θ b不成立，得到结果为0(FALSE)。


```
short a = 0, b = 1, c = -1, d;
d = a >= b;          0
d = (a * c) != b;    1
a <= b;              1
b < c;               0
a <= b < c;          0
a <= (b < c);        1
```

说明在a <= b < c 中，先计算a <= b，得1，再计算1 < c，得0。

其运算法则是自左向右进行计算的。

优先级：<、<=、>、>= 高于 ==，!=

如：表达式　x > y == c < d

等价于　(x > y) == (c < d)

关系运算符的优先级低于算术运算符的优先级

如：x > u+v　等价于　x > (u+v)

2. 逻辑运算

逻辑运算符

&&(逻辑与)、||(逻辑或)、!(逻辑非)

其中：运算符 && 和 || 是双目运算符，要求有两个运算分量；运算符 ！是单目运算符，只要求一个运算分量。

优先级： !、&&、||

说明： 逻辑运算结果也是一个逻辑量，即真(用1表示)或假(用0表示)。判定一个运算分量的值为真或假时，以运算分量的值不等于零为真，值等于0 为假。

25

逻辑运算表达式

1） 逻辑与运算表达式的形式为

a && *b*

如果*a*为真且*b*为真，则*a* && *b*的结果等于真。

如果*a*为假或*b*为假，则*a* && *b*的结果等于假。

2） 逻辑或运算表达式的形式为

a || b

如果*a*为真或*b*为真，则*a* || *b*的结果等于真。

如果*a*为假且*b*为假，则*a* || *b*的结果等于假。

26

3） 逻辑非运算表达式的形式为

!*a*

如果*a*为"真(TRUE)"，则!*a*的结果为逻辑值"假(FALSE)"。

如果*a*为"假(FALSE)"，则!*a*的结果为逻辑值"真(TRUE)"。

27

逻辑运算真值表

a	b	!a	a&&b	a||b
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

28

连续比较

数学上的连续比较 5>3>2

在数学上是恒成立的。但在C语言中，上式却不成立，因为首先计算第一个大于号，其值为1，而后计算第二个大于号时，成为计算1>2，显然不成立，其值为0。实际上，连续比较大小时，表示几个条件同时满足，因此若将上式改写为条件表达式

5>3 && 3>2

后，则与数学上的的连续比较含意相符，表达式也是成立的。

29

优先级： ！ (非)

算术运算符

关系运算符

&& 和 ||

赋值运算符

！的优先级高于算术运算符的优先级。 && 和 || 的优先级低于关系运算符的优先级；逻辑运算符||和&&的结合方向是自左至右，而逻辑运算符!的结合方向是自右至左。

30

如：　a > b && x > y　⇔ (a > b) && (x > y)

a != b || x != y ⇔(a != b) || (x != y)

x == 0 || x < y && z > y ⇔

(x == 0) || ((x < y) && (z > y))

!b || x > y || a > b ⇔

((!b) || (x > y)) || (a > b)

!a && b || x > y && z < y ⇔

((! a) && b) || ((x>y) &&(z<y))

在算术、关系、逻辑混合运算的表达式中，不同位置上出现的运算分量，应区分哪些是算术运算分量、哪些是关系运算分量和哪些是逻辑运算分量。

例　如：　'2' > 1 && 4 && 7 < 3 + !0

等效于：　(('2' > 1) && 4) && (7 < (3 + (!0)))

1　1　1　4　0　0

逻辑运算的分量也可以是字符型、指针型。以0或非0判定它们的"假"或"真"。

逻辑优化

"逻辑与"和"逻辑或"运算符的性质：

1) a && b，当 a 为 0 时，不管 b 为何值（不再计算 b），结果为 0。只要计算出a值为0，则不需计算b值，这种情况，我们称之为逻辑与优化。

2) a || b，当 a 为 1 时，不管 b 为何值（不再计算b），结果为 1。只要计算出a值为1，则不需计算b值，这种情况，我们称之为逻辑或优化。

例1：a=1, b=2, c=3, c=4, m=5, n=6

(m=a>b) && (n=c>d)

∵ m = 0, ∴ n=c>d不计算，n仍为6

例2：　int a = 1, b = 1, c = 1;

计算 ++a || ++b && ++c，因++a非0，不再计算逻辑或右边表达式++b&&++c。该表达式计算后，变量a的值变为2，而变量b和c的值不变，依旧为1。

3. 注意书写顺序：

例如：　y/x > 2 且 x != 0,

应写成：　x != 0 && y/x > 2

因为，当 x 为0时，不会计算 y/x。

而写成：　y/x > 2 && x != 0　是不正确的，因为当 x 为0时，不能计算 y/x。

2.3.4 条件运算

条件运算需要有三个运算分量参加计算，称为三目运算

一般形式：　e1 ? e2 : e3

其中：　1. ?:　为条件运算符

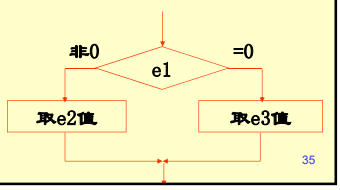
2.　三个运算分量e1、e2、e3 均为表达式。

条件运算的表述为：

如果e1 为"真"（条件成立），其运算结果为e2

如果e1 为"假"（条件不成立），其运算结果为e3

执行过程：



例如，最常用的是绝对值的求取。以下是宏定义ABS(X)的用法：

```
# define ABS(X) (X>0) ? (X) : -(X) /* 定义 */
a = ABS(b); /* 使用宏 */
a = (b>0) ? (b) : -(b) /* 等价 */
```

以下是输出a,b中的大者.

```
例　if (a>b)
      printf("%d",a);
    else
      printf("%d",b);
```

⇔

```
printf("%d",a>b?a:b);
```

结合方向：自右向左

```
max = x > y ? x : y + 1
⇔ max = ((x > y) ? x : (y + 1))
```

条件运算符可嵌套：

```
x > y ? x : u > v ? u : v
⇔ x > y ? x : (u > v ? u : v)
```

条件运算三个运算分量的数据类型可以各不相同。C语言约定，类型低的向类型高的转换。

如： i > j ? 2 : 3.5

当i>j时，条件表达式的值为2.0；否则为3.5。

【例2.6】表达式求值示例

```
#include <stdio.h>
int main()
{ int i = 1, j = 2, k = 3;
  i += j += k;
  printf("i=%d\tj=%d\tk=%d\n", i, j, k);
  /* i = 6, j = 5, k = 3 */
  printf("(i<j?i++:j++)=%d\n", i < j ? i++ : j++);
  /*(i<j?i++:j++)=5*/
  printf("i=%d\tj=%d\n", i, j); /* i = 6, j = 6 */
  printf("(k+=i>j?i++:j++)=%d\n", k += i>j ? i++ : j++);
  /*(k+=i>j?i++:j++)=9*/
  printf("i=%d\tj=%d\tk=%d\n", i, j, k);
   /* i=6, j=7, k=9 */
```



```
                    /* k>=j结果为1，而1<i */
 i = 3; j = k = 4;
printf("(k>=j>=i)=%d", k >= j >= i);
printf("\t(k>=j&&j>=i)=%d\n", k >= j && j >= i);
/* (k>=j>=i)=0     (k>=j&&j>=i)=1*/
   i = j = 2;   k = i++ - 1;
   printf("i=%d\tj=%d\tk=%d\n", i, j, k);
/* i=3 , j=2, k=1 */
   k += -i++ + ++j;    /* k=1+(-3+(2+1)) */
   printf("i=%d\tj=%d\tk=%d\n", i, j, k);
/* i=4, j=3, k=1 */
   return 0;
}
```


说明

(1) C表达式 k>=j>=i与数学式子k>=j>=i的区别。

C 表达式 k>=j>=i因运算符 >= 自左向右结合，可写成(k>=j)>=i。数学式子 k>=j>=i写成C表达式应该是 k>=j && j>=i。

【例2.7】表达式求值示例

```
#include <stdio.h>
int main()
{  int a, b, c;  a = b = c = 1;     ++a || ++b && ++c;
   printf("a=%d\tb=%d\tc=%d\n", a, b, c); /* a=2,b=1,c=1 */
   a = b = c = 1;     ++a && ++b || ++c;
   printf("a=%d\tb=%d\tc=%d\n", a, b, c); /* a=2,b=2,c=1 */
   a = b = c = 1;     ++a && ++b && ++c;
   printf("a=%d\tb=%d\tc=%d\n", a, b, c); /* a=2,b=2,c=2 */
   a = b = c = 1;     --a && --b || --c;
   printf("a=%d\tb=%d\tc=%d\n", a, b, c); /* a=0,b=1,c=0 */
   a = b = c = 1;     --a || --b && --c;
   printf("a=%d\tb=%d\tc=%d\n", a, b, c); /* a=0,b=0,c=1 */
   a = b = c = 1;     --a && --b && --c;
   printf("a=%d\tb=%d\tc=%d\n", a, b, c); /* a=0,b=1,c=1 */
   return 0;
}
```


2.3.5其他运算

1. 逗号运算

形式： 表达式1，表达式2，…，表达式n

结合性：从左向右

计算过程： 从前往后，依次计算各表达式

逗号表达式的值： 等于表达式n的值

例：

```
#include <stdio.h>
void main()
{    int x,y=7;
     float z=4;
     x=(y=y+6,y/z);
     printf("x=%d\n",x);
}
```

程序运行结果为：

```
X=3
```

通常将逗号运算用在将若干表达式连接起来顺序地逐个计算的场合。例如在循环控制结构
for 中，常常会用到逗号运算。
如：

```
for(i=0, j=1; i<5; i++, j++)
{
    ......
}
```

43

2. 求字节数运算： sizeof

功能： 给出运算对象在内存中所占用的字节数。
形式： sizeof(变量名) 或者 sizeof 变量名
sizeof(类型名) 或者 sizeof 类型名

```
#include <stdio.h>
void main()
{
    short x;
    printf("bytes of x =%d\n", sizeof(x));
    printf("bytes of double =%d\n",
    sizeof(double));
}
```

运行结果：

```
bytes of x = 2
bytes of double = 8
```

44

3. 按位运算（只要求了解）

位运算把运算对象看作是由二进位组成的位串信息，按位运算，得到位串信息的结果。
位运算的运算分量只能是整型或字符型数据。

位运算符：

- 按位与 &　　a & b
- 按位或 |　　a | b
- 按位异或 ^　　a ^ b
- 按位非 ~　　~ a

45

a	b	~a	a&b	a|b	a^b
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

说明： 按位取反是单目运算符，其余均为双目运算符。
优先级： ~ 、 & 、 ^ 、 |

46

(1) 按位与运算示例： 53 & 22的结果为20。

```
  0 000 000 000 110 101   (十进制53，八进制为65)
& 0 000 000 000 010 110   (十进制22，八进制为26)
  0 000 000 000 010 100   (十进制20，八进制为24)
```

位与运算主要典型用法：取位串的某几位。
例如，截取x的最低7位的方法是，x & 0177
0177是八进制的写法，对应的二进制为001 111 111。
保留变量的某几位，其余位置0。
例如，保留变量x最低6位，代码x = x & 077
可实现这个要求。

47

(2) 按位或运算示例： 53 | 22的结果为55。

```
  0 000 000 000 110 101   (十进制53，八进制为65)
| 0 000 000 000 010 110   (十进制22，八进制为26)
  0 000 000 000 110 111   (十进制55，八进制为67)
```

位或运算的典型用法：
是将一个位串的某几位置成1。
例如，将变量x的最低4位设置为1，
其余位不变，代码
x = 017 | x 可实现这个要求。

48

(3) 位异或运算示例： 53 ^ 22的结果为35。

	0 000 000 000 110 101	(十进制53，八进制为65)
^	0 000 000 000 010 110	(十进制22，八进制为26)
	0 000 000 000 100 011	(十进制35，八进制为43)

位或运算的典型用法是将一个位串某几位取反。

例如，将变量x的最低4位取反，其余位不变，即在最低4位中，原来是1的位变为0，原来是0的位变为1。代码

x = 017 ^ x 就可实现这个要求。

(4) 位取反运算示例：

~53

~	0 000 000 000 110 101	(十进制53，八进制为65)
	1 111 111 111 001 010	(十进制-54，八进制为177712)

负数在计算机内部是用补码表示的。一个负数首先应将原码取反变成反码，然后再加1才能变成补码形式。同样，将上述补码的结果

1 111 111 111 001 010变为原码，应先将补码减1变为反码1 111 111 111 001 001，再将每位取反（除符号位外）变为原码1 000 000 000 110 110。对应的八进制数是 -66，十进制数就是6×8+6=54。由于它是负值，结果为-54。

4. 移位运算

移位运算是指对移位变量 a 按照二进制向左或者向右移动n.

移位运算符有两个：

(1) 左移运算符： a<<n

(2) 右移运算符： a>>n

移位运算有两个操作数，左操作数a为移位的数据对象，右操作数的值n为移位位数。

移位运算符的优先级低于算术运算符，高于关系运算符，它们的结合方向是自左至右。

例如：char k, m; k = 14, m=64;

m=64	01000000	k = 14	00001110
m << 1 => -128	10000000	k << 1 => 28	00011100
m << 2 => 0	00000000	k << 2 => 56	00111000
		k >> 1 => 7	00000111
		k >> 2 => 3	00000011

对于无符号整数的左移和右移运算

a << n 等价于a * 2^n

a >> n 等价于a / 2^n

2.4 表达式

表达式：由运算符和运算分量构成。

从表达式的构成形式区分，可分以下几类：

2.4.1表达式分类

- 算术表达式：如x+1.0/y-z%5
- 关系表达式：如 x>y+z
- 逻辑表达式：如x>y&&x<z
- 赋值表达式：如x=(y=z+5)
- 条件表达式：如x>y?x:y
- 逗号表达式：如x=1, y++, z+=2

2.4.2 表达式的类型转换

◆混合运算的问题

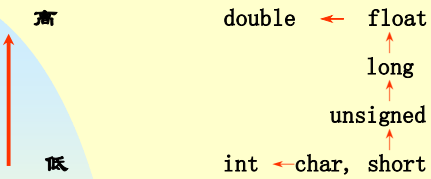
在一个算术表达式(+、-、*、/、%)中，如果不同计算分量的数据类型不同，称为混合运算。混合运算过程中必须要将不同类型的数据转换成同一个数据类型，才能够进行计算。

例如：在表达式"3 * 4.6"中，3 是整型，4.6 是实型，必然需要先将其中的一个计算分量转换成另一个计算分量的数据类型（此处是将3 转换成3.0），然后计算3.0 * 4.6。

混合运算中数据类型转换的法则

- 隐式类型转换
- 强制类型转换（显式转换）

1. 隐式类型转换规则



说明：1. 横向向左箭头←表示必定的转换。如，float型数据运行前必须先转换为double型。
2. 纵向的箭头↑表示，当运算对象为不同类型转换的方向。

隐式转换法则示例

```
#include <stdio.h>
void main()
{
    float a, c;
    short b, d;
    c = 4.6;
    d = 3;
    a = c * d;
    b = c * d;
    printf("a=%f\n,", a);
    printf("b=%d\n", b);
}
```

运行结果：
a=13.8
b=13

2. 强制类型转换

将一种类型的表达式值强制转换成另一种类型.

一般格式：（类型名）表达式

其中（类型名）是对其后的表达式作强制类型转换。它将表达式的值强制地转换成类型名所指明的类型。

例如：
3.5 * 4.6 得16.1
3.5 * (int)4.6 得14.0
(int)3.5 * (int)4.6 得12
(int)3.5 * 4.6 得13.8
(int)(3.5 * 4.6) 得16

例如，库函数 sqrt()是求一个double 型值的平方根。为求整型变量m 的平方根，正确的写法是(int) sqrt((double)m)

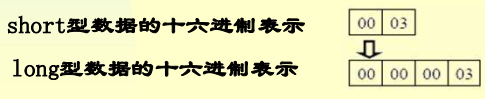
表示在求m的平方根之前，应该先将m的值转换成double型，然后去调用函数sqrt()，并将结果转换成int型。

说明：类型转换不只改变表达式值的类型，也可能会因两种表示形式上的差异，值的大小会有一些误差。

1. 如果将实型（优先级高）的数据转换成整型（优先级低）的数据，存储单元的值将实行取整，即舍去小数部分. 3.1415 ➡ 3

2. 如果将整型（优先级低）的数据转换成实型（优先级高）的数据，存储单元的小数部分值将实行补零. 3 ➡ 3.0

3. 如果将较短整型的数据转换成较长整型的数据，将在存储单元的高位部分值实行补零。



4. 如果将较长整型的数据转换成较短整型的数据，存储单元的高位部分将被截去。

short型数据	511
char型数据	-1
short型数据的十六进制表示	01 FF
char型数据的十六进制表示	FF

61

数据运算小结

♦根据运算分量的数量分类

◉ 单目运算

+(取正)、-(取负)、++(自增)、--(自减)、
sizeof()(求字节数)、

!(逻辑非)、 ()(圆括号)、 (类型)(强制转换)、
~(按位非)、

以下各个运算符将在以后的课程中介绍

[ ](下标)、&(取址)、*(取值)、
.(结构分量)、->(结构指针分量)

62

◉ 双目运算

+(加)、-(减)、*(乘)、/(除)、%(求余)、
<(小于)、<=(小于等于)、>(大于)、>=(大于等于)、
= =(相等)、!=(不相等)、
&& (逻辑与)、|| (逻辑或)、
=(赋值)、复合赋值、,(逗号)、
&(按位与)、|(按位或) 、^(按位异或)、
<<(左移)、>>(右移)

◉ 三目运算

? : (条件)

运算符的优先级与结合性请见附录A(p223)

63

优先级	类型	运算符	运算功能	运算分量个数	结合方向
1(高)	初等运算符	( ) [ ] -> .	圆括号 下标 结构指针分量 结构分量	1	从左向右
2	逻辑运算 按位运算 自增减运算 取正负运算 强制转换运算 指针运算 求字节运算	! ~ ++ -- + - (类型) * & sizeof()	逻辑非 按位非 自增 自减 取正 取负 强制转换 取值 取址 求字节数	1	从右向左
3	算术运算	* / %	乘 除 求余	2	从左向右
4	算术运算	+ -	加 减	2	从左向右
5	移位运算	<< >>	左移 右移	2	从左向右
6	关系运算	< <= > >=	小于 小于等于 大于 大于等于	2	从左向右
7	关系运算	== !=	相等 不相等	2	从左向右
8	按位运算	&	按位与	2	从左向右
9	按位运算	^	按位异或	2	从左向右
10	按位运算	|	按位或	2	从左向右
11	逻辑运算	&&	逻辑与	2	从左向右
12	逻辑运算	||	逻辑或	2	从左向右
13	条件运算	? :	条件	3	从右向左
14	赋值运算	=、复合赋值	赋值	2	从右向左
15(低)	逗号运算	,	逗号		从左向右

♦ 运算符的优先级顺序

◉ 同一优先级的运算符，运算次序取决于结合方向
例如，-和++为同一优先级，结合方向为从右到左，
因此：-j++等价于-(j++)

◉ 根据一览表，可以归纳出各类运算符的优先级。
如图所示。

初等运算符 优先级高
↓
单目运算符
↓
算术运算符 先乘除后加减
↓
关系运算符
↓
逻辑运算符 不包括!(非)
↓
条件运算符
↓
赋值运算符
↓
逗号运算符 优先级低

65