

本章学习目标

- 了解表达式的使用；
- 掌握赋值运算符的使用；
- 掌握算术运算符的使用；
- 掌握关系运算符的使用；
- 掌握逻辑、位逻辑运算符的使用；
- 掌握复合运算符的使用。

每一个 C 语言程序都需要对不同数据类型的数据进行处理并运算,否则程序将没有任何意义。因此,掌握 C 语言中各种运算符以及表达式的应用是必不可少的。本章将主要介绍运算符以及相关表达式的使用方法,其中包括赋值运算符、算术运算符、关系运算符、逻辑运算符、位逻辑运算符以及复合赋值运算符。

3.1 表 达 式

C 语言中的表达式由操作符和操作数组成,最简单的表达式可以只有一个操作数。根据表达式所包含操作符的个数,可以将表达式分为简单表达式和复杂表达式,简单表达式只含有一个操作符,而复杂表达式可以包含两个或两个以上操作符。

常见的表达式如下所示。

```
8
1+2
Num1+(Num2 * Num3)
```

表达式本身不做任何操作,只用来返回结果值。当程序不对返回的结果值进行任何操作时,返回的结果值不起任何作用。如上述表达式中,第一个表达式直接返回 8,第二个表达式返回 1 与 2 的和,第三个表达式返回 Num1 与 Num2、Num3 乘积的和。

表达式既可以放在赋值语句的右侧也可以放在函数的参数中。

! 注意:

表达式返回的结果值是有类型的,表达式隐含的数据类型取决于组成表达式的变量和常量的类型。

例 3.1 表达式的使用。

```

1  #include <stdio.h>
2
3  int main(int argc, const char * argv[])
4  {
5      int a, b, c;
6
7      a = 1;
8      b = 2;
9      /* 表达式中使用变量 a 加常量 2 * /
10     c = a + 2;
11     printf("c = %d\n", c);          /* 输出变量 c 的值 * /
12     /* 表达式中使用变量 b 加常量 2 * /
13     c = b + 2;
14     printf("c = %d\n", c);          /* 输出变量 c 的值 * /
15     /* 表达式中使用变量 a 加变量 b * /
16     c = a + b;
17     printf("c = %d\n", c);          /* 输出变量 c 的值 * /
18     return 0;
19 }


```

 输出：

```

c = 3
c = 4
c = 3

```

 分析：

第 5 行：声明变量的表达式，使用逗号通过一个表达式声明 3 个变量。

第 7、8 行：使用常量为变量赋值的表达式。

第 10 行：表达式将变量 a 与常量 2 相加，然后将返回的值赋值给变量 c。

第 11 行：使用函数 printf() 输出变量 c 的值。

第 13 行：表达式将变量 b 与常量 2 相加，然后将返回的值赋值给变量 c。

第 14 行：使用函数 printf() 输出变量 c 的值。

第 16 行：表达式将变量 a 与变量 b 相加，然后将返回的值赋值给变量 c。

第 17 行：使用函数 printf() 输出变量 c 的值。

3.2 表达式语句

表达式组成的语句称为表达式语句，具体如下所示。

```

int a = 5;
a;

```

第 1 行代码为 a 赋初始值为 5，同时返回 a 的值为 5，因此“int a = 5”是一个表达式。由于代码末尾多了一个分号，其变为一个表达式语句。第 2 行代码直接返回 a 的值 5，末尾

也多了一个分号,因此第 2 行代码同样是一个表达式语句。

3.3 运算符

C 语言的运算符范围很广,将除了控制语句和输入输出以外的几乎所有的基本操作都作为运算符处理。例如,将赋值符“=”作为赋值运算符,将方括号作为下标运算符等。本节将介绍 C 语言中常用的几种运算符。

运算符用于执行程序代码运算,会针对一个以上的操作数进行运算。C 语言中,常用的运算符如表 3.1 所示。

表 3.1 C 语言常用的运算符

含 义	运 算 符
算术运算符	+、-、*、/、%
自增自减运算符	++、--
赋值运算符	=
复合赋值运算符	+ =、- =、* =、/ =、% =、>> =、<< =、& =、 =、^ =
关系运算符	>、<、= =、! =、> =、< =
逻辑运算符	!、&&、
位运算符	~、 、^、&、<<、>>
条件运算符	?:
逗号运算符	,
指针运算符	*
取地址运算符	&
点运算符	.
下标运算符	[]
函数调用运算符	{ }
括号运算符	()
箭头运算符	->

如表 3.1 所示,需要一个操作数参与的运算符称为单目运算符,如++、& 等;需要两个操作数参与的运算符称为双目运算符,如+、* 等。

3.4 赋值运算符

在 C 语言程序中,“=”并非等于的意思,其表示的是赋值符号,即赋值运算符,其作用是将一个数据赋值给一个变量。在声明变量时,可以为其赋一个初始值,该初始值可以是一个常量,也可以是一个表达式的结果,如下所示。

```
char c = 'A';
int i = 10;
int Count = 1 + 2;
```

如上述赋值操作,得到赋值的变量 `c`、`i`、`Count` 称为左值,其在赋值符号的左侧,产生值的表达式称为右值,其在赋值符号的右侧。

! 注意:

并不是所有的表达式都可以作为左值,如常数只可以作为右值。

在声明变量时,直接对其赋值称为赋初值,即变量的初始化。也可以先将变量声明,再进行变量的赋值操作,如下所示。

```
int i;           //声明变量
i = 12;         //为变量赋值
```

3.5 算术运算符与表达式

3.5.1 算术运算符

C 语言中的算术运算符主要用来实现各种数学运算,包括两个单目运算符(正与负),5 个双目运算符,即乘法、除法、取模、加法以及减法。算术运算符具体描述如表 3.2 所示。

表 3.2 算术运算符

运算符	含义	举例	结果
-	负号运算符	-2	-2
+	正号运算符	+2	+2
*	乘法运算符	2 * 3	6
/	除法运算符	5 / 2	2
%	取模运算符	5 % 2	1
+	加法运算符	2 + 3	5
-	减法运算符	3 - 2	1

如表 3.2 中,加减乘除运算符与数学中的四则运算相通,这里不再详细介绍。其中,取模运算符 `%` 用于计算两个整数相除得到的余数。例如,5 除以 3 的结果为 1 余 2,求模运算的结果为 2。这里需要注意求模运算符 `%` 两侧只能是整数,结果的正负取决于被求模数(即运算符左侧的操作数),如 $(-5) \% 3$,结果为 -2。取模运算符的使用如例 3.2 所示。

例 3.2 取模运算。

```
1 #include <stdio.h>
2
3 int main(int argc, const char * argv[])
4 {
```

```

5   printf("%d\n", 5%3);
6   printf("%d\n", (-5)%3);
7   printf("%d\n", 5%(-3));
8   return 0;
9  }
```

 输出:

```

2
-2
2
```

 分析:

第 5 行: 输出 5 除以 3 得到的余数。

第 6 行: 输出 -5 除以 3 得到的余数。

第 7 行: 输出 5 除以 -3 得到的余数。

由三次执行结果可知,取模运算结果的正负取决于被取模数(运算符左侧操作数)。

 注意:

运算符“-”作为减法运算符时,为双目运算符;作为负值运算符时,为单目运算符。

3.5.2 算术表达式

在表达式中使用算术运算符,则将表达式称为算术表达式,如下所示。

```

Num = (3+5) / Rate;
Area = Height * Width;
```

需要说明的是,两个整数相除的结果为整数,如 7/4 的结果为 1,舍去小数部分。如果除数或被除数为负数,则采取“向零取整”的方法,即为 -1,如例 3.3 所示。

例 3.3 整数相除。

```

1  #include <stdio.h>
2  int main(int argc, const char * argv[])
3  {
4      printf("%d\n", 7/4);
5      printf("%d\n", (-7)/4);
6      printf("%d\n", 7/(-4));
7      return 0;
8  }
```

 输出:

```

1
-1
-1
```

3.5.3 算术运算符的优先级与结合性

1. 算术运算符的优先级

C语言中规定了各种运算符的优先级和结合性。对于算术运算符,表达式求值会按照运算符的优先级高低次序执行,其中*、/、%的优先级别高于+、-的级别。例如,在表达式中同时出现*与+,则会先运行乘法再运行加法,如下所示。

```
R = x + y * z;
```

在表达式中,因为*比+的优先级高,所以会先执行 $y * z$ 的运算,再加 x 。

2. 算术运算符的结合性

当算术运算符的优先级别相同时,结合方向为“自左向右”,如下所示。

```
R = a - b + c;
```

如上述表达式语句中,因为加法与减法的优先级别相同,所以 b 先与减号相结合,执行 $a - b$ 的操作,然后再执行加 c 的操作,该操作过程称为“自左向右的结合性”。

3.6 自增、自减运算符

C语言提供了两个用于变量递增与递减的特殊运算符,分别为自增(自加)运算符“++”与自减运算符“--”。自增运算符使操作数递增1,自减运算符使操作数递减1。

在C语言程序设计中,经常使用运算符++(--)来递增(递减)变量的值。其用法格式如下所示。

```
int n = 0;
n++;
n--;
```

第1行代码将 n 的值初始化为0。

第2行代码使用自增运算符++将 n 的值加1,该语句执行完毕后, n 的值由0变为1。

第2行代码执行完后 n 的值为1,第3行代码再通过自减运算符--使 n 的值减少1,该语句执行完毕后, n 的值变为初始值0。

! 注意:

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 $2++$ 或 $(a+b)--$ 都是不合法的。因为2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5,那么没有用来存放自减后的结果4的变量。

自增运算符与自减运算符可以放在变量的前面或后面,放在变量的前面称为前缀,放在变量的后面称为后缀,具体如下所示。

```
--a;           //自减前缀符号
a--;           //自减后缀符号
++b;           //自加前缀符号
b++;           //自加后缀符号
```

在表达式内部,作为运算的一部分,以上两种使用方法可能有所不同。如果运算符放在变量前,则变量在参加表达式运算之前完成自增或自减运算;如果运算符放在变量后面,则变量的自增或自减运算在参加表达式运算之后完成。

1. 前置自增、自减运算


前置自增运算符的作用是先将变量的值增加 1,然后再取增加后的值;前置自减运算符的作用是先将变量的值减少 1,然后再取减少后的值。接下来通过一个示例展示前置自增和自减运算符的作用,具体如例 3.4 所示。

例 3.4 前置自增、自减运算。

```
1  #include <stdio.h>
2
3  int main(int argc, const char * argv[])
4  {
5      int i = 0;
6
7      printf("i = %d\n", ++i);
8      printf("i = %d\n", i);
9      printf("i = %d\n", --i);
10     printf("i = %d\n", i);
11
12     return 0;
13 }
```

 输出:

```
i = 1
i = 1
i = 0
i = 0
```

 分析:

第 5 行: 将 i 的值初始化为 0。

第 7 行: 输出 $++i$ 的值,自增运算符 $++$ 放在 i 的前面,因此它是前置自增运算符,作用是先将 i 的值增加 1,再取 i 的值, i 的初始值为 0,加 1 后变为 1,这样再输出 i 的值,输出结果为 1。

第 8 行: 再次输出 i 的值, i 的值为 1,表示 i 的值产生变化。

第 9 行: 输出 $--i$ 的值,自减运算符 $--$ 放在 i 的前面,因此它是前置自减运算符,作用是先将 i 的值减少 1,再取 i 的值, i 的值为 1,减 1 后变为 0,这样再输出 i 的值,输出结果为 0。

第 10 行: 输出 i 的值, i 的值为 0,表示 i 的值产生变化。

2. 后置自增、自减运算

后置自增运算符的作用是先取变量的值,然后再使其值增加 1;后置自减运算符的作用


也是先取变量的值,然后再使其值减少 1。接下来通过一个示例展示后置自增和自减运算符的作用,具体如例 3.5 所示。

例 3.5 后置自增、自减运算。

```
1  #include <stdio.h>
2
3  int main(int argc, const char * argv[])
4  {
5      int i = 0;
6
7      printf("i = %d\n", i++);
8      printf("i = %d\n", i);
9      printf("i = %d\n", i--);
10     printf("i = %d\n", i);
11
12     return 0;
13 }
```

 输出:

```
i = 0
i = 1
i = 1
i = 0
```

 分析:

第 5 行: 将 i 的值初始化为 0。

第 7 行: 输出 $i++$ 的值,自增运算符 $++$ 放在 i 的后面,因此它是后置自增运算符,作用是先取变量 i 的值执行赋值操作,再将 i 的值加 1, i 的初始值为 0,先执行赋值,因此输出 i 的值为 0,赋值后 i 的值变为 1。

第 8 行: 再次输出 i 的值, i 的值为 1,表示 i 的值产生变化且发生在赋值操作之后。

第 9 行: 输出 $i--$ 的值,自减运算符 $--$ 放在 i 的后面,因此它是后置自减运算符,作用是先取变量 i 的值执行赋值操作,再将 i 的值减 1, i 的值为 1,先执行赋值,因此输出 i 的值为 1,赋值后 i 的值变为 0。

第 10 行: 再次输出 i 的值, i 的值为 0,表示 i 的值产生变化且发生在赋值操作之后。

3.7 关系运算符与表达式

在 C 语言中,关系运算符的作用是判断两个操作数的大小关系。

3.7.1 关系运算符

关系运算符包括大于、大于等于、小于、小于等于、等于和不等,如表 3.3 所示。

表 3.3 关系运算符

符 号	功 能	符 号	功 能
>	大于	<=	小于等于
>=	大于等于	==	等于
<	小于	!=	不等于

3.7.2 关系表达式

关系运算符用于对两个表达式的值进行比较,然后返回一个真值或假值(1 或 0)。返回真值或假值取决于表达式中的值和运算符。真值表示指定的关系成立,假值则表示指定的关系不正确。具体的关系表达式如下所示。

```

2 > 1           //2 大于 1,因此关系成立,表达式的结果为真
2 >= 1          //2 大于 1,因此关系成立,表达式的结果为真
2 < 1           //2 大于 1,因此关系不成立,表达式的结果为假
2 <= 1          //2 大于 1,因此关系不成立,表达式的结果为假
2 == 1          //2 大于 1,因此关系不成立,表达式的结果为假
2 != 1          //2 不等于 1,因此关系成立,表达式的结果为真

```

关系运算符通常用来构造条件表达式,用在程序流程控制语句中。例如,if 语句用于判断条件而执行语句块,在其中使用关系表达式作为判断条件,如果关系表达式返回为真则执行下面的语句块,如果关系表达式返回为假则不执行,具体操作如下所示。

```

if (Num < 5) {
    ...           //判断条件为真,则执行该代码
}

```

! 注意:

在 C 语言程序设计时,等号运算符“==”与赋值运算符“=”切勿混淆使用。

3.7.3 关系运算符的优先级与结合性

关系运算符的结合性都是自左向右。如表达式 $i < j < k$ 在 C 语言中虽然是合法的,但该表达式并不是测试 j 是否位于 i 和 k 之间,因为 $<$ 运算符是左结合,所以该表达式等价于 $(i < j) < k$,即表达式首先检测 i 是否小于 j ,然后用比较后产生的结果(1 或 0)与 k 进行比较。

3.8 复合赋值运算符与表达式

3.8.1 复合赋值运算符

赋值运算符与其他运算符组合,可以构成复合赋值运算符,C 语言中一共有 10 种复合赋值运算符,具体如表 3.4 所示。

表 3.4 复合赋值运算符

运算符	含义	举例	结果
<code>+=</code>	加法赋值运算符	<code>a += 1</code>	<code>a + 1</code>
<code>-=</code>	减法赋值运算符	<code>a -= 1</code>	<code>a - 1</code>
<code>*=</code>	乘法赋值运算符	<code>a *= 1</code>	<code>a * 1</code>
<code>/=</code>	除法赋值运算符	<code>a /= 1</code>	<code>a / 1</code>
<code>%=</code>	取模赋值运算符	<code>a %= 1</code>	<code>a % 1</code>
<code>>>=</code>	按位右移赋值运算符	<code>a >>= 1</code>	<code>a >> 1</code>
<code><<=</code>	按位左移赋值运算符	<code>a <<= 1</code>	<code>a << 1</code>
<code>&=</code>	按位与赋值运算符	<code>a &= 1</code>	<code>a & 1</code>
<code> =</code>	按位或赋值运算符	<code>a = 1</code>	<code>a 1</code>
<code>^=</code>	按位异或赋值运算符	<code>a ^= 1</code>	<code>a ^ 1</code>

如表 3.4 中,前 5 种用于算术运算,后 5 种用于位运算。

3.8.2 复合赋值表达式

本节将只介绍前面 5 种用于算术运算的复合赋值运算符(位运算的复合赋值运算符后续再介绍)。

1. 加法赋值运算符

加法赋值运算符即将加法运算符与赋值运算符组合,运算符表达式对应的语句如下所示。

```
a += 2;
```

如上述操作语句,先将变量 `a` 加 2,再将结果赋值给 `a`。假如 `a` 的值为 1,则 `a += 2` 后,`a` 的值变为 3。

2. 减法赋值运算符

减法赋值运算符即将减法运算符与赋值运算符组合,运算符表达式对应的语句如下所示。

```
a -= 2;
```

如上述操作语句,先将变量 `a` 减 2,再将结果赋值给 `a`。假如 `a` 的值为 3,则 `a -= 2` 后,`a` 的值变为 1。

3. 乘法赋值运算符

乘法赋值运算符即将乘法运算符与赋值运算符组合,运算符表达式对应的语句如下所示。

```
a *= 2;
```

如上述操作语句,先将变量 a 乘 2,再将结果赋值给 a 。假如 a 的值为 1,则 $a * = 2$ 后, a 的值变为 2。

4. 除法赋值运算符

除法赋值运算符即将除法运算符与赋值运算符组合,运算符表达式对应的语句如下所示。

```
a /=2;
```

如上述操作语句,先将变量 a 除以 2,再将结果赋值给 a 。假如 a 的值为 4,则 $a /= 2$ 后, a 的值变为 2。

5. 取模赋值运算符

取模赋值运算符即将取模运算符与赋值运算符组合,运算符表达式对应的语句如下所示。

```
a %=2;
```

如上述操作语句,先将变量 a 除以 2,再将余数赋值给 a 。假如 a 的值为 5,则 $a %= 2$ 后, a 的值变为 1。

接下来通过具体的示例验证上述 5 种复合赋值运算符,具体如例 3.6 所示。

例 3.6 复合赋值运算符。

```
1  #include <stdio.h>
2
3  int main(int argc, const char * argv[])
4  {
5      int a =1, b =3, c =1, d =4, n =5;
6      printf("%d\n", a +=2);
7      printf("%d\n", b -=2);
8      printf("%d\n", c * =2);
9      printf("%d\n", d /=2);
10     printf("%d\n", n %=2);
11
12     return 0;
13 }
```

 输出:

```
3
1
2
2
1
```

 注意:

复合赋值运算符右侧可以是带运算符的表达式,如以下操作。

```
a *= 1 + 2;
```

该语句等同于 $a = a * (1 + 2)$ ；运算符 $*$ 右侧的值必须先求出，因此需要使用括号，而不等同于 $a = a * 1 + 2$ ；不加括号则是另外一个结果，严重错误。

C 语言采用复合赋值运算符是为了简化程序，采用复合赋值运算符的表达式计算机更容易理解，可以提高编译效率。

3.9 逻辑运算符与表达式

3.9.1 逻辑运算符

逻辑运算符用于表达式执行判断真或假并返回真或假。逻辑运算符有 3 种，具体如表 3.5 所示。

表 3.5 逻辑运算符

符 号	功 能
&&	逻辑与
	逻辑或
!	单目逻辑非

! 注意：

逻辑与运算符“&&”和逻辑或运算符“||”都是双目运算符。

3.9.2 逻辑表达式

使用逻辑运算符可以将多个关系表达式的结果合并在一起进行判断，具体如下所示。

```
Result = A && B;           //当 A 和 B 都为真时，结果为真
Result = A || B;          //A、B 其中一个为真时，结果为真
Result = !A;              //如果 A 为真，结果为假
```

一般情况下，逻辑运算符用来构造条件表达式，用在控制程序的流程语句中，如 if、for 语句等。

在 C 语言程序中，通常使用单目逻辑非运算符“!”将一个变量的数值转换为相应的逻辑真值或假值(1 或 0)，具体如下所示。

```
Result = !!Value;
```

3.9.3 逻辑运算符的优先级与结合性

逻辑运算符的优先级从高到低依次为单目逻辑非运算符“!”、逻辑与运算符“&&”和逻辑或运算符“||”，这些逻辑运算符的结合性都是自左向右。

逻辑运算符的使用具体如例 3.7 所示。

例 3.7 逻辑运算符。

```

1  #include <stdio.h>
2
3  int main(int argc, const char * argv[])
4  {
5      int Value1, Value2;
6
7      Value1 = 5;
8      Value2 = 0;
9
10     printf("Value1 && Value2 的结果是%d\n", Value1 && Value2);
11     printf("Value1 || Value2 的结果是%d\n", Value1 || Value2);
12
13     printf("!!Value1 的结果是%d\n", !!Value1);
14     return 0;
15 }


```

 输出：

```

Value1 && Value2 的结果是 0
Value1 || Value2 的结果是 1
!!Value1 的结果是 1

```

 分析：

第 7 行：赋值变量 Value1 的值为非 0 值。

第 8 行：赋值变量 Value2 的值为 0。

Value1 的值为非 0(表示成立), Value2 的值为 0(表示不成立), 因此在逻辑与表达式中, 判断为不成立(结果为 0), 在逻辑或表达式中, 判断为成立(结果为 1), Value1 经过第一次取非运算后, 其值变为 0, 再次取非后, 其值变为 1。

3.10 位逻辑运算符与表达式

3.10.1 位逻辑运算符

位逻辑运算符应用在一些特定的场合中, 可以实现位的设置、清零、取反和取补操作。位逻辑运算符包括位逻辑与、位逻辑或、位逻辑非、取补, 具体如表 3.6 所示。

表 3.6 位逻辑运算符

符 号	功 能
&	位逻辑与
	位逻辑或
^	位逻辑非
~	取补

如表 3.6 所示,前三个位逻辑运算符都是双目运算符,最后一个运算符为单目运算符。位逻辑运算符通常用于对整型变量进行位的设置、清零与取反,以及对特定位进行检测。

3.10.2 位逻辑表达式

在 Linux 内核驱动程序中,位逻辑运算符可以通过位操作设置寄存器中的值,操作输入输出设备。位逻辑运算符的使用如下所示。

```
Result=Value1 & Value2;
```

如上述表达式语句,假设 Value1、Value2 为整型变量,Value1 的值为 5,Value2 的值为 6,将 Value1、Value2 转换为二进制数分别为 101、110。位逻辑运算即对数值的每一个对应位执行位操作,因此执行的结果为 100,转换为十进制数 4。

位逻辑运算符的使用,具体如例 3.8 所示。

例 3.8 位逻辑运算符。

```
1  #include <stdio.h>
2
3  int main(int argc, const char * argv[])
4  {
5      int Value1, Value2, Result;
6
7      Value1 =5;
8      Value2 =6;
9
10     Result=Value1 & Value2;
11
12     printf("Result 的值为%d\n", Result);
13     return 0;
14 }
```



输出:

```
Result 的值为 4
```

3.11 运算符的优先级

在一个表达式中,可能包含有多个不同的运算符以及不同数据类型的数据对象。由于表达式有多种运算,不同的结合顺序可能得到不同的结果甚至运算错误。因此,在表达式中含多种运算时,必须按一定顺序进行结合,才能保证运算的合理性和结果的正确性。

表达式的结合次序取决于表达式中各种运算符的优先级,优先级高的运算符先结合,优先级低的运算符后结合。每种同类型的运算符都有内部的运算符优先级,不同类型的运算符之间也有相应的优先级顺序。一个表达式中既可以包括相同类型的运算符,也可以包括不同类型的运算符。

在 C 语言中,运算符的优先级与结合性如表 3.7 所示。

表 3.7 运算符的优先级与结合性

优先级	运算符	名称或含义	使用形式	结合方向	说 明
1	[]	数组下标	数组名[常量表达式]	自左向右	
	()	圆括号	(表达式)、函数名(形参表)		
	.	成员选择(对象)	对象.成员名		
	->	成员选择(指针)	对象指针->成员名		
2	-	负号运算符	-表达式	从右向左	单目运算符
	(类型)	强制类型转换	(数据类型)表达式		
	++	自增运算符	++变量名、变量名++		单目运算符
	--	自减运算符	--变量名、变量名--		单目运算符
	*	取值运算符	* 指针变量		单目运算符
	&	取地址运算符	& 变量名		单目运算符
	!	逻辑非运算符	! 表达式		单目运算符
	~	按位取反运算符	~表达式		单目运算符
sizeof	长度运算符	sizeof(表达式)			
3	/	除	表达式/表达式	从左向右	双目运算符
	*	乘	表达式*表达式		双目运算符
	%	取余	整型表达式%整型表达式		双目运算符
4	+	加	表达式+表达式	从左向右	双目运算符
	-	减	表达式-表达式		双目运算符
5	<<	左移	变量<<表达式	从左向右	双目运算符
	>>	右移	变量>>表达式		双目运算符
6	>	大于	表达式>表达式	从左向右	双目运算符
	>=	大于等于	表达式>=表达式		双目运算符
	<	小于	表达式<表达式		双目运算符
	<=	小于等于	表达式<=表达式		双目运算符
7	==	等于	表达式==表达式	从左向右	双目运算符
	!=	不等于	表达式!=表达式		双目运算符
8	&	按位与	表达式&表达式	从左向右	双目运算符
9	^	按位异或	表达式^表达式	从左向右	双目运算符
10		按位或	表达式 表达式	从左向右	双目运算符
11	&&	逻辑与	表达式&&表达式	从左向右	双目运算符
12		逻辑或	表达式 表达式	从左向右	双目运算符

续表

优先级	运算符	名称或含义	使用形式	结合方向	说 明
13	?:	条件运算符	表达式 1? 表达式 2:表达式 3	从右向左	三目运算符
14	=	赋值运算符	变量=表达式	从右向左	
	/=	除后赋值	变量/=表达式		
	=	乘后赋值	变量=表达式		
	%=	取模后赋值	变量%=表达式		
	+=	加后赋值	变量+=表达式		
	-=	减后赋值	变量-=表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&.=	按位与后赋值	变量&.=表达式		
	^=	按位异或后赋值	变量^=表达式		
15	=	按位或后赋值	变量 =表达式		
	,	逗号运算符	表达式,表达式,...	从左向右	

3.12 本章小结

本章主要介绍了 C 语言程序中常用的各种运算符以及表达式的使用。首先介绍了表达式与运算符的概念,帮助读者了解后续章节所需的准备知识,然后分别详细介绍了赋值运算符、算术运算符、关系运算符、复合赋值运算符、逻辑运算符、位逻辑运算符的具体使用。不同等级的运算符具有不同的优先级以及结合性,因此在使用运算符时,需要时刻考虑,避免因优先级问题产生不必要的错误。

3.13 习 题

1. 填空题

- (1) 表达式 $17\%3$ 的结果是_____。
- (2) 若 `int a=10`;则执行 `a+= a-=a*a`;后,a 的值是_____。
- (3) 在 C 语言中,要求操作数都是整数的运算符是_____。
- (4) 若 `int sum=7,num=7`;则执行语句 `sum=num++`;`sum++`;`++num`;后 sum 的值为_____。
- (5) 若 `int a=6`;则表达式 $a\%2+(a+1)\%2$ 的值为_____。

2. 选择题

- (1) 若变量 a,i 已正确定义,且 i 已正确赋值,则以下合法的语句是()。

A. `a==1` B. `++i`; C. `a++=5`; D. `a=int(i)`;

- (2) 在 C 语言程序中,运算对象必须是整型的运算符是()。
- A. %= B. / C. <= D. =
- (3) 若 `int a=7; float x=2.5, y=4.7;` 则表达式 `x+a%3*(int)(x+y)%2/4` 的值是()。
- A. 2.500000 B. 2.750000 C. 3.500000 D. 0.000000
- (4) 以下选项中,与 `k=n++`; 完全等价的表达式是()。
- A. `k=n; n=n+1;` B. `n=n+1; k=n;`
C. `k=++n;` D. `k+=n+1;`
- (5) 设 `x` 和 `y` 均为 `int` 型变量,则语句 `x+=y; y=x-y; x-=y` 的功能是()。
- A. 把 `x` 和 `y` 按从大到小排列 B. 把 `x` 和 `y` 按从小到大排列
C. 无确定结果 D. 交换 `x` 和 `y` 中的值
- (6) 假设整型变量 `a` 与 `b` 被分别赋值为十进制数 7 与 8,则执行位与操作后的结果为()。
- A. 0 B. 15 C. 7 D. 8
- (7) 以下运算符优先级最低的是()。
- A. () B. > C. / D. &&

3. 思考题

- (1) 请简述前置自加运算符与后置自加运算符的区别。
- (2) 请简述除法运算符与求模运算符的区别。

4. 编程题

- (1) 定义一个变量 `a`,且 `a` 的初始值为 5,依次输出前置自加和后置自减的值。
- (2) 定义两个 `int` 型变量 `n` 和 `m`,且设定初始值为 4 和 8,依次输出 `n+=m`、`n-=m`、`n*=m`、`m/=n`、`m%=n` 复合赋值运算的结果。