第 3 章　运算符与表达式

计算机最基本的功能就是运算，C语言中提供了多种运算符，这些运算符将常量和变量按照一定的规则组合成了表达式。运算符对我们来说并不陌生，数学中学过的加（+）、减（–）、乘（×）、除（÷）等符号都是运算符，不过它们仅仅是 C 语言中运算符的一部分。有运算符便能构成算式，数学中的算式就相当于 C 语言中的表达式，数学算式是有计算结果的，所以 C 语言中的表达式也要产生一个结果，称为表达式的值。

C 语言的表达式功能强大，不仅可以运算，还可以在一个表达式中执行若干复杂的动作，如调用函数，执行输入、输出等。所以，人们往往把 C 语言视为表达式语言。

本章的主要内容如下：

各运算符的表示形式、作用及使用要点；

各运算符构成表达式的规则；

各运算符的优先级及结合性。

3.1　表达式

通俗地讲，表达式就是一个运算的式子，类似于数学中的算式，例如，2+3，4+sin(5)等，只要是算式，就会有计算结果，C 语言的表达式也不例外，只要是表达式就会产生结果，这个结果我们称为表达式的值。C 语言的表达式分为基本表达式和复杂表达式。

一个简单的变量、常量，以及一次函数调用都是一个基本表达式。

常量构成的基本表达式，例如，100，1.2，'a'，012，0x2b，"ABC"等，对应的表达式的值就是其本身。

变量构成的表达式，例如，int a=3;语句，变量 a 是一个表达式，其值是 3。

由于一次函数调用的结果也是一个常量，因此一次函数调用也属于基本表达式，表达式的值就是函数的返回值，例如，sin(0)，其表达式的值是 0。

用圆括号括起来的表达式也是基本表达式，例如，(10)、(a)、("abc")。

将基本表达式通过运算符连接在一起，就构成了复杂表达式，例如，1+2，a+b，(a–b)/2。

复杂表达式还可组成更加复杂的表达式，例如，(a+b)*(10+b)。

基本表达式相对简单，本章主要介绍运算符和运算分量结合在一起构成的复杂表达式。

3.2　优先级与结合方向

1．优先级

在 2+3*2 算式中，*（乘号）的优先级高于+（加号），因此在计算时，需要先计算 3*2。计算机在处理这样的表达式时，也是遵照相同的方式处理的，得到的结果为 8。

2．结合方向

在 2+3–5+6–7 算式中，+和–的优先级是相同的，在数学运算中，我们先运算哪一部分都可以，但是计算机的处理过程是有先后次序的，因此我们规定了结合方向，也就是当运算符优先级相同的情况下，计算机按结合方向进行运算，算术运算符的结合方向是从左向右，运算过程如下：

2+3–5+6–7
5–5+6–7
0+6–7
6–7
–1

优先级相同时，数学运算中的运算次序可以随意，而计算机中不可以这样，例如，下面两个表达式在数学运算中的结果是相同的，而在计算机中却不相同，在计算机中的运算过程如下：

2*3/6，值为 1，原因是从左向右结合，2*3/6=6/6=1；

3/6*2，值为 0，原因是从左向右结合，3/6*2=0*2=0。

3．使用优先级和结合方向处理一个表达式

算式 1+2+4*5–2*6，在数学运算中的处理方法是先计算 4*5 和 2*6，在计算机中会严格按照“优先级、结合方向”规则处理表达式，处理次序如下：

1 + 2 + 4*5–2*6	两个+优先级相同，按结合方向，计算 1+2
3 + 4 * 5 – 2*6	+优先级低于*，*高于–，计算 4*5
3+ 20 – 2*6	+、–优先级相同，按结合方向，计算 3+20
23 – 2 * 6	–优先级低于*，计算 2*6
23 – 12	计算 23–12
11	结果

3.3 基本算术运算符

1．单目算术运算符和双目算术运算符

C 语言中，将只需要一个操作数的运算符称为单目算术运算符，如–a；将需要两个操作数的运算符称为双目算术运算符，如 a+b。

2．基本算术运算符

表 3-3-1 列出了 C 语言的基本算术运算符。

表 3-3-1 C 语言的基本算术运算符

基本算术运算符	名　称	运算规则	示　例	示例结果
–	负	变相反数	–(–5)	5
*	乘	相乘	2*3	6

续表

基本算术运算符	名　　称	运算规则	示　　例	示例结果
/	除	相除	5/2	2
%	模	两个整数相除，取余数	5%2	1
+	加	相加	2+3	5
−	减	相减	2−3	−1

在表格中，负运算是单目算术运算符，优先级最高；*、/、%次之；+、−最低。

在 C 语言中没有乘方运算符，要计算 a^3 可以写成 a*a*a 的连乘形式，或者使用后面介绍的标准库函数 pow(a,3)。

除运算符的运算对象可以是各种类型的数据，但是当进行**两个整型数据相除时，运算结果也是整型数据**，即只取商的整数部分，不会四舍五入，如 2/10 的运算结果是 0（而不是 0.5），10/4 的运算结果是 2（而不是 2.5）；当**浮点数和整数相除时，结果是浮点数**，如 10.0/4 的运算结果是 2.5。

模运算要求运算对象必须是整型数据，它的功能是求两个操作数相除的余数，余数的符号与被除数的符号相同，如 11%2、11%−2 的值均为 1，−11%2、−11%−2 的结果均为−1。

程序示例如下：

```
//程序 3-3-1 基本算术运算符
#include "stdio.h"
int main(){
   float a,b;
   printf("%d,%d,%d,%d,%d\n",5+2,5-2,5*2,5/2);//输出 7，3，10，2
   a=5/2;
   b=5.0/2;
   printf("%f,%f\n",a,b);//输出 2.000000，2.500000
   return 0;
}
```

3．算术表达式

用算术运算符和圆括号将运算对象连接起来的式子称为算术表达式，例如：

```
a+1/(b+9)- 'A'
1-a%10+5.6*3
```

在算术表达式中，运算对象可以是各种类型的数据，包括整型、实型或字符型的常量、变量及函数调用。对运算对象按照算术运算符的规则进行运算，得到的结果就是算术表达式的值。由此可见，表达式的计算过程就是求表达式值的过程。求出的值也有数据类型，它取决于参加运算的操作对象。

3.4　自增、自减运算符

C 语言中提供了两个特殊的运算符：自增运算符（++）和自减运算符（−−）。它们都是单目算术运算符，分别对应低级语言指令中的自增、自减运算。执行周期短，效率高，运算符可以出现在变量的前面或后面。当出现在运算分量的前面时，称为前缀运算符，例如，++i、−−i；当出现在运算分量的后面时，称为后缀运算符，例如，i++、i−−等。

自增、自减运算符和运算对象可组成的表达式如下：

```
++i      /*先使 i 的值增大 1，然后再使用增大了值的 i*/
--i      /*先使 i 的值减小 1，然后再使用减小了值的 i*/
i++      /*先使用 i，然后再使 i 的值增大 1*/
i--      /*先使用 i，然后再使 i 的值减小 1*/
```

首先强调，**++、--是双功能运算符**，功能一是能像其他运算符一样提供一个表达式值，功能二是改变变量本身的值。另外，3.5 节中介绍的赋值运算符也属于双功能运算符。

++、--运算符既可以用于字符型、整型变量，也可以用于浮点型、双精度型变量。

使用++、--运算符时应注意以下内容：

只能用于单个变量，用于常量或复杂表达式时都将出错，如 5++、(–a)++是错误的；

运算符为后缀运算符时，如 i++，先用后加；

运算符为前缀运算符时，如++i，先加后用；

当出现 a+++b 类似情况时，遵循“最长匹配”原则，从左向右取尽可能多的符号组成运算符，可理解为(a++)+b；

在一个表达式内不可出现多个 a++或++a，如(a++)+(a++)+(++a)，不同的编译系统处理方法可能不同，结果难以预期；

在调用函数时，由于处理参数的顺序可能不一样，因此会产生歧义，例如，设 i 的初值为 3，printf("%d,%d",i,i++);语句的输出结果为“4,3”，C 语言的处理次序是从后面开始的，而 Pascal 方式是从前面开始的，因此需要避免此种情况的发生。

程序示例如下：

```
//程序 3-4-1  ++前置与后置
#include "stdio.h"
int main(){
   int a=1,b=1;
   printf("%d,%d\n",a++,++b);//输出 1，2
   return 0;
}
```

运算符的前置和后置为什么有区别呢？C 语言是高级语言，最终必须翻译成机器语言才能执行。程序示例如下：

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

在调试过程中将机器代码反汇编成汇编语言，结果如图 3-4-1 所示。

```
3:       int a=1,b;
00401028   mov         dword ptr [ebp-4],1     //a赋值1
4:       b=a++;
0040102F   mov         eax,dword ptr [ebp-4]   //把a值传给eax寄存器
00401032   mov         dword ptr [ebp-8],eax   //把eax寄存器内容传给b
00401035   mov         ecx,dword ptr [ebp-4]   //把a值装入 ecx寄存器
00401038   add         ecx,1                   //ecx寄存器加1
0040103B   mov         dword ptr [ebp-4],ecx   //把ecx寄存器内容赋给a
5:       printf("%d\n",b);
0040103E   mov         edx,dword ptr [ebp-8]
00401041   push        edx
00401042   push        offset string "%f,%f\n" (0042601c)
00401047   call        printf (00401080)
0040104C   add         esp,8
6:     }
```

图 3-4-1　结果

分析如下。

b=a++;翻译成汇编语言类似于：

```
mov b,a;                          //把 a 中的数据送入 b
add a,1;                          //a+1
```

b=++a;翻译成汇编语言类似于：

```
add a,1;                          //a+1
mov b,a;                          //把 a 中的数据送入 b
```

这就是为什么++运算符放在变量前和变量后结果不同的原因，翻译成目标代码后“取用”和“加 1”的指令次序不同。程序示例如下：

```
//程序 3-4-3  ++示例
#include "stdio.h"
int main(){
   int a=1,b,c;
   b=a++;                         //a++表达式值为 1，b 获得值 1，a 自增 1 变为 2
   c=b++;                         //b++表达式值为 1，c 获得值 1，b 自增 1 变为 2
   printf("a=%d,b=%d,c=%d\n",a,b,c);//输出 a=2，b=2，c=1
   return 0;
}
```

3.5　赋值运算符

赋值运算符（=）不仅是运算符，还是双功能运算符。其功能一是将右侧表达式的值赋给左侧的变量，例如，a=3，a 中保存结果 3；功能二与其他运算符相同，提供一个表达式值，表达式返回的值就是赋值后变量中保存的结果。赋值表达式的一般形式为

```
变量=表达式
```

+和=运算符产生表达式值的对比如图 3-5-1 所示，a+3 产生表达式值 5，a=3 产生表达式值 3，那么 b=a+3 中的=是不是运算符呢？当然也是，只是这次产生的表达式值被丢弃，并没有保存。

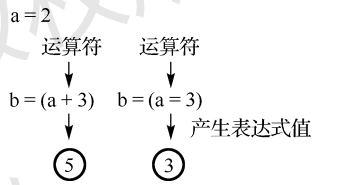

图 3-5-1　+和=运算符产生表达式值的对比

赋值运算符的左侧只能是变量，因为它表示一个存放值的空间。例如：

```
2=a;    /*错误*/
a+b=c;  /*错误*/
```

这样的赋值表达式显然是不合法的。**在使用赋值表达式时应注意以下几点：**

赋值运算符左侧的量必须是单个变量，不能是常量或表达式；

赋值运算符的结合方向是从右向左结合；

赋值运算符优先级很低，仅高于“,”；

赋值运算可连续进行。

例如：

```
a=b=c=0
```

其中，有 3 个赋值运算符。按照赋值运算符的结合方向，从右向左依次结合。这个表达式等同于 a=(b=(c=0))。其赋值过程是从右向左进行的，即先将 0 赋给 c，再将(c=0)表达式的值赋给 b，最后将(b=0)表达式的值 0 赋给 a。最终的结果是 a、b 和 c 变量的值都为 0。以下程序对变量赋值进行了演示。

```
//程序 3-5-1 =运算符
#include "stdio.h"
int main(){
    int a,b,c,d;
    a=b=2;                          //a，b 都变为 2
    b=a+3;                          //b 变为 5
    c=a+b;                          //c 变为 7
    printf("a=%d,b=%d,c=%d,d=%d\n",a,b,c,d);
                                    //输出 a=2，b=5，c=7，d=-858993460
    return 0;
}
```

其中 d 变量声明后并未赋过值，因此保存的是原来遗留的垃圾数据。

3.6 关系运算符

我们在处理数据时经常需要比较数据的大小，例如，查询某同学成绩是否及格，需要先查找一个学号（与输入数据相等的记录），再判断对应成绩是否大于等于 60。进行大小比较的运算符就是关系运算符，比较的结果是“是”或“否”。例如，2>3，结论不成立，比较的结果是“否”。也有一些编程语言会用逻辑常量 true 表示“真、是、yes”；用逻辑常量 false 表示“假、否、not”。C 语言没有逻辑常量 true 和 false，常用 0 表示 false，1 表示 true。直到出现 C++语言才有逻辑常量 true 和 false，但 C++语言中的 true 和 false 本质上仍然是 1 和 0。

C 语言中提供了 6 种关系运算符，如表 3-6-1 所示。

表 3-6-1 关系运算符

关系运算符	名　　称	运算规则	示　　例	示例结果
>	大于	左侧是否大于右侧	2>3	0
<	小于	左侧是否小于右侧	2<3	1
>=	大于等于	左侧是否大于或等于右侧	2>=3	0
<=	小于等于	左侧是否小于或等于右侧	2<=2	1

续表

关系运算符	名　　称	运算规则	示　　例	示例结果
==	相等	左侧、右侧数据是否相等	2==3	0
!=	不等	左侧、右侧数据是否不等	2!=3	1

关系运算符都是双目算术运算符，其优先级低于算术运算符，高于赋值运算符，结合方向是从左向右。在关系运算符中，<、<=、>、>=同级，高于= =和!=，= =和!=同级。

用关系运算符将表达式连接起来的式子称为关系表达式。例如，m>n、8<5、a+b<=c+d、(i=j+k)!=0 都是合法的关系表达式。

关系运算符的优先级高于赋值运算符，因此(i=j+k)!=0 中圆括号的有无是不一样的。添加圆括号表示将 j+k 的值赋给 i，然后判定 i 是否不等于 0；删除圆括号，相当于 i=((j+k)!=0)，也就是先判断 j+k 的值是否不等于 0，然后将逻辑值 0 或 1 赋给 i。

使用关系运算符需要特别注意不能将“==”写成“=”，例如：

```
int a=0,b;
b=(a==5);        //判断 a 是否是 5，b 获得值为 0，a 中的值不会改变
```

如果错写成

```
b=(a=5);         //a、b 都赋值为 5，b 获得值为 5，同时 a 的值改为 5
```

在程序中会引发严重的逻辑错误。为避免这种错误，编码时可颠倒左、右两边的量。例如，a==5，如果错写成 a=5，编译程序无法发现错误，使用颠倒左、右两边的量的方式，即 5==a，如果错写成 5=a，编译无法通过，可以发现错误。

另外一个错误也是初学者容易犯的，例如，有 int a=50，如果我们要判断 a 是否介于 5 和 7 之间，数学中的写法是 5<a<7，但是在 C 语言程序中将产生逻辑错误。在 C 语言中会先计算 5<a，因为不成立，所以结果是 0，再比较 0<7，成立，最终结果是 1。显然这和我们想要的结论是相反的，那么，要判断 a 是否介于 5 和 7 之间通常需要使用逻辑运算符，将在 3.7 节介绍。程序示例如下：

```
//程序 3-6-1 关系运算符程序示例
#include "stdio.h"
int main(){
    int a=2,b=3,c,d,e;
    c=a>=2;
    d=a==b;
    e=b!=3;
    printf("c=%d,d=%d,e=%d\n",c,d,e);        //输出 c=1，d=0，e=0
    return 0;
}
```

3.7　逻辑运算符

1．逻辑运算符的概念

当描述条件时，经常会使用“并且”“或者”等表示条件间的关系。例如，年满 18 周岁并且身体健康可以报名参军，表示年满 18 周岁和身体健康这两个条件必须都达到，才可以报名参军，若有一个条件不成立，则结果就不成立。再如，有六级英语证书或托福成绩超过 90

分的同学可以免修大学英语课程，表示六级英语证书和托福成绩超过 90 分这两个条件只要有一个成立，结果就成立。这里所说的“并且”“或者”关系可以用逻辑运算符表示。

C 语言中提供了&&（与）、||（或）、!（非）逻辑运算符，逻辑运算符的关系如表 3-7-1 所示。

表 3-7-1 逻辑运算符的关系

逻辑运算符	名　称	含　义	运算规则	示　例	示例结果
!	非	不	结论变反，0 变 1，1 变 0	！(2<3)	0
&&	与	并且	全 1 出 1，见 0 出 0	2<3&&1	1
||	或	或者	见 1 出 1，全 0 出 0	x>=0||x<0	1

逻辑运算符中，！是单目算术运算符，优先级最高，其次是&&，最低的是||。逻辑运算的结果只能是 0 或 1，代表 false 和 true。C 语言中规定，**在逻辑判断中非 0 值等同于 1**，因此，0.0001&&0.001 表达式的值是 1，12345&&0.000 表达式的值为 0。

2．逻辑运算符的短路现象

逻辑运算并不完全遵循优先级和结合方向的运算规则，其短路现象如下：

0 && 表达式，由于&&的结果一定是 0，因此右侧表达式不再执行；

1 || 表达式，由于||的结果一定是 1，因此右侧表达式不再执行。

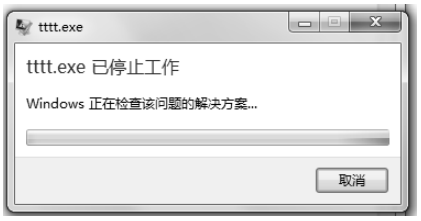

图 3-7-1 出现严重错误

为什么会出现短路现象呢？例如：

```
if(a!=0 && 20/a>2)…
```

表达式 a!=0 && 20/a>2，如果 a 的值是 0，0 作为除数，20/a 一旦执行会出现严重错误，如图 3-7-1 所示。而实际情况是，由于 C 语言的短路现象，a!=0 不成立，结果为 0，因此后面的 20/a 不再执行，就避免了系统故障。

3．逻辑表达式

用逻辑运算符将关系表达式或逻辑值连接起来的符合 C 语言语法的式子称为逻辑表达式，如(a>b)&&(c>d)、!((a+b)>c)。程序示例如下：

```
//程序 3-7-1 逻辑表达式程序示例
#include <stdio.h>
int main(){
    int a=0,b=1,c=2,d;
    c=a++&&b++;      //a++表达式值为 0，因此 b++不执行，c 获得值 0
    d=!(a>0||c++);  //上式中使 a 变为 1，a>0 表达式值为 1，c++不执行，d 被赋值 0
    printf("a=%d,b=%d,c=%d,d=%d\n",a,b,c,d);
                    //输出 a=1，b=1，c=0，d=0
    return 0;
}
```

3.8 条件运算符

条件运算符“?:”是 C 语言中唯一的三目算术运算符，它连接三个运算量，例如：

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

由条件运算符组成的表达式称为条件表达式，一般形式为

```
表达式 1?表达式 2:表达式 3
```

条件运算符的执行方法是：

先计算表达式 1 的值；

如果表达式 1 的值为非 0，则用表达式 2 的值作为整个条件表达式的值；

如果表达式 1 的值为 0，则用表达式 3 的值作为整个条件表达式的值。

例如：

```
int a=3,b;
b=a>0?15:20;
```

a>0 的值为 1，因此 b 的值为 15。

表达式 1 起条件判别作用，根据其值的真假来选择执行后面两个表达式中的哪一个。

例如，要求出 a 和 b 中最大的一个，并赋予变量 max，可使用如下语句：

```
max=(a>b)?a:b;
```

在执行时，先计算表达式 a>b 的值，如果 a>b，结果为 1，就将 a 的值作为整个条件表达式的值赋给 max，否则就将 b 的值作为整个条件表达式的值赋给 max。

条件运算符的优先级高于赋值运算符和逗号运算符，但比其他运算符的优先级都低。

条件运算符的结合方向是自右向左的，对 a>0 ? b : c>0 ? e : f 有以下两种解释：

```
a>0?b:(c>0?e:f)      //从右向左结合，将 c>0?e:f 先看成整体，正确
(a>0?b:c>0)?e:f      //从左向右结合，将 a>0?b:c>0 先看成整体，错误
```

三个运算分量不限于简单的算术表达式，可以是多个表达式的组合，甚至可以是函数调用。程序示例如下：

```
//程序 3-8-1 条件运算符程序示例
#include <stdio.h>
int main(){
    int a=2,b=3,c=4,d;
    d=a>b?++b:++c;       //++b 没有执行，并非计算完后没有取用，而是不执行这个分支
    printf("b=%d,c=%d,d=%d\n",b,c,d);   //输出 b=3，c=5，d=5
    return 0;
}
```

从上面程序示例可以看出，**条件运算符也具有短路现象**，其会根据第 1 个表达式的值来决定执行第 2 个表达式还是第 3 个表达式，没被选择的表达式不会执行。

3.9　逗号运算符

在 C 语言中，逗号的用途主要有两种：分隔和运算。

1．分隔

逗号是 C 语言中的标点符号之一，用来分隔多个数据。在变量定义时，具有相同类型的

多个变量可在一行中定义，之间用逗号分隔即可。例如：

```
int a,b,c;
char m,n;
```

另外，函数的参数也可用逗号分隔，例如：

```
printf("a=%d  b=%d  c=%d",a,b,c);
```

2．运算

逗号作为运算符时，其右侧的值是表达式的值，如 2,3，表达式的值是 3，“+”运算符与“,”运算符的对比如图 3-9-1 所示。

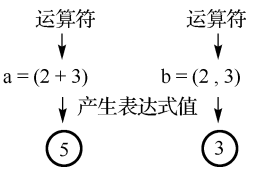

图 3-9-1 “+”运算符与“,”运算符的对比

x=(1+2,6*3)的执行过程如下：

```
x=(1+2,6*3)      //先计算 1+2，得到 3
x=(3,6*3)        //然后执行 6*3，得到 18
x=(3,18)         //逗号表达式的值是 18
x=18             //赋给 x 变量的值也是 18
18
```

需要注意的是，**逗号运算符的优先级在所有运算符中是最低的**，因此如果上式写为 x=1+2,6*3，则执行过程变为

```
x=1+2,6*3
x=3,6*3
3,18
18
```

整个表达式最后的值是 18，而 x 获得的值是 3。逗号运算符的结合方向是从左向右的，它可以将多个表达式连接起来，如 a=b,c++,d– –就是一个逗号表达式。逗号表达式的求值过程是从左向右、逐个求表达式的值，最后整个表达式的值取最右的一个表达式的值。上面运算的最终表达式的值并没有取用，我们只是用这种方式将每个子表达式执行了一遍，相当于用“,”列出了一些需要执行的表达式列表，让计算机按次序逐个完成。若将 a=b,c++,d– –表达式中“,”换成“;”，则变成 a=b;c++;d– –;三条语句，而 a=b,c++,d– –;只是一条语句。程序示例如下：

```
//程序 3-9-1 逗号运算符程序示例
#include <stdio.h>
int main(){
    int a,b;
    a=(2,3);
    b=2,3;     //=优先级高于,
    printf("a=%d,b=%d\n",a,b);   //输出 a=3，b=2
    return 0;
}
```

```
//程序 3-9-2 逗号连接表达式
#include <stdio.h>
int main(){
    int a,b,c;
    a=1,a++,b=++a,c=2+a; //子表达式从左向右顺序执行
    printf("a=%d,b=%d,c=%d\n",a,b,c); //输出 a=3，b=3，c=5
```

```
        return 0;
    }
```

3.10　位逻辑运算符

位运算是一种对运算对象按二进制位进行的操作运算。我们知道，计算机中所有的数据都是用二进制数形式存储的。在汇编语言中，通过对二进制数按位进行运算处理，可以将字或字节中的某一位置 0 或置 1，也可以对某一位进行测试，再根据测试结果进行相应的处理。而 C 语言通过位运算能够实现汇编语言的大部分功能，这种处理能力是一般高级语言所不具备的，这使得 C 语言可用在自动控制领域。位运算分为位逻辑运算和移位运算。位逻辑运算符的运算规则如表 3-10-1 所示。

表 3-10-1　位逻辑运算符的运算规则

位逻辑运算符	名　　称	运算规则	示　　例	示例结果
～	按位取反	每个二进制位 0 变 1，1 变 0	～(–2)	1
&	按位与	对应二进制位全 1 出 1，见 0 出 0	5&6	4
|	按位或	对应二进制位见 1 出 1，全 0 出 0	5|6	7
^	按位异或	对应二进制位不同出 1，相同出 0	5^6	3

～、&、^、|的优先级依次降低，但是为了程序的可读性，最好使用圆括号表明优先级。由于位运算是对内存中的二进制位进行运算，因此与存储空间的大小有关系，只要在不影响结果的情况下，我们可以采用尽可能少的位数计算。

（1）～（按位取反）运算：每个二进制位 0 变 1，1 变 0

–2 二进制原码　1000　0010B

　　二进制补码　1111　1110B

———————————————

～ 运算结果　　0000　0001B

十进制值 1

（2）&（按位与）运算：对应二进制位全 1 出 1，见 0 出 0

5　二进制补码　0000 0101B

6　二进制补码　0000 0110B

———————————————

& 运算结果　　0000 0100B

十进制值 4

（3）|（按位或）运算：对应二进制位见 1 出 1，全 0 出 0

5 二进制补码　0000 0101B

6 二进制补码　0000 0110B

———————————————

| 运算结果　　0000 0111B

十进制值 7

（4）^（按位异或）运算：对应二进制位不同出 1，相同出 0

5 二进制补码　0000 0101B

6 二进制补码　0000 0110B

---

^ 运算结果　　0000 0011B

十进制值 3

注意：只有字符或整数可参与位逻辑运算，浮点数不可参与。

3.11　移位运算符

移位运算就是将二进制位整体左移或右移，计算机中的移位运算包括算术移位和逻辑移位，逻辑移位中空出的位置都补 0，而算术移位中要保证数的正、负号不会改变，就是在右移时，符号位仍然保持原来的正、负号不变，也就是正数补 0，负数补 1。移位运算符有两个：左移运算符（<<）和右移运算符（>>）。它们都是双目算术运算符，并且要求两个运算分量都是整型量。由移位运算符和运算分量构成的表达式称为移位表达式。

1．左移运算符（<<）

左移运算符的一般形式为

```
表达式 1<<表达式 2
```

在移位表达式中，“表达式 1”是移位对象，“表达式 2”是移位的位数。它的功能是：将表达式 1 的值（以二进制数形式表示）向左移动 *n* 位，*n* 的值由表达式 2 确定，右侧空出的位补 0，而左侧溢出的位丢弃。

我们用 char a=6;执行 a<<1，查看运算过程，如图 3-11-1 所示。

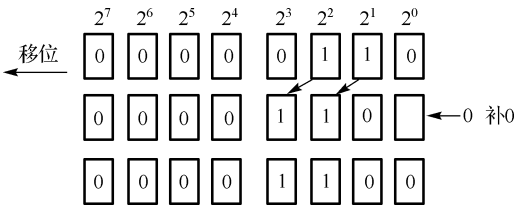

图 3-11-1　运算过程

从图 3-11-1 中可以看出，1 分别从 2^2、2^1，提升到了 2^3、2^2，相当于每位二进制位乘 2，所以每**左移一位相当于乘 2**。

但是如果出现了溢出，如数字移到了符号位上，符号位移出了表达空间，则计算机也会继续执行。例如：

```
char a=127; a=a<<1;
```

a 的二进制数表示：0111 1111

左移 1 位，0 丢失，右侧补 0：1111 1110

求出原码：1000 0010

对应十进制值：–2

程序示例如下：

```
//程序 3-11-1  <<移位运算
#include <stdio.h>
int main(){
     char a=6;  //a 原码 0000 0110，补码 0000 0110
     char b=-6; //b 原码 1000 0110，补码 1111 1010
     a=a<<2;    //变为 0001 1000
     b=b<<2;    //变为 1110 1000，原码 1001 1000
     printf("a=%d,b=%d\n",a,b); //输出 a=24，b=-24
     return 0;
}
```

2．右移运算符（>>）

右移运算符的一般形式为

```
表达式 1>>表达式 2
```

“表达式 1”是移位对象，“表达式 2”是移位的位数。它的功能是：将表达式 1 的值（以二进制数形式表示）向右移动 *n* 位，*n* 的值由表达式 2 确定，并且表达式 2 的值必须是正整数。例如，a=00001011，移位表达式 a>>3 的结果是 00000001，即将 a 的各二进制位全部向右移 3 位，右侧溢出的位丢弃。

在右移时，要注意符号位的问题。如果移位对象是无符号量，那么右移时左侧空出来的位全以 0 填充，称为逻辑移位；如果移位对象是有符号量，那么补的是符号位数字，即正数补 0，负数补 1，称为算术移位。程序示例如下：

```
//程序 3-11-2
#include <stdio.h>
int main(){
    char a=4;   //原码 0000 0100B，补码 0000 0100B
    char b=-4;  //原码 1000 0100B，补码 1111 1100B
    a=a>>1;     //右移，补符号位数字，0->0000 010，对应原码 0000 0010
    b=b>>1;     //右移，补符号位数字，1->1111 110，对应原码 1000 0010
    printf("a=%d,b=%d\n",a,b);      //输出 a=2，b=-2
    return 0;
}
```

```
//程序 3-11-3 算术移位与逻辑移位
#include <stdio.h>
int main(){
    signed short a=-2048;               //a 原码 1000 1000 0000 0000
                                        //a 补码 1111 1000 0000 0000
    unsigned short b=63488;             //b 值 1111 1000 0000 0000
    printf("a=%hX,b=%hX\n",a,b);        //输出 a=F800，b=F800
    a=a>>1;                             //算术移位
    b=b>>1;                             //逻辑移位
    printf("a=%hX,b=%hX\n",a,b);        //输出 a=FC00，b=7C00
}
```

注意：只有字符或整数可参与移位运算，浮点数不可参与。

3.12 复合运算符

复合运算符是由算术运算符、位运算符等双目算术运算符和赋值运算符结合在一起组成的，以达到简化书写程序和提高运算效率的目的，复合运算符有+=、-=、*=、/=、%=、|=、^=、&=、>>=、<<=。例如：

a+=b+c　　等价于　　a=a+(b+c)
a*=b+c　　等价于　　a=a*(b+c)
a%=b+c　　等价于　　a=a%(b+c)
a&=b+c　　等价于　　a=a&(b+c)
a<<=2　　等价于　　a=a<<2

复合运算符在书写时要注意两个运算符之间不能有空格，否则将出现错误。复合运算符是一个运算符，它的优先级和结合方向与赋值运算符相同，例如：

```
a=2;
a*=5+4; //“*=”的优先级和“=”相同，因此先执行 5+4，然后再执行 a=a*9 操作
```

阅读以下程序，分析输出结果：

```
//程序 3-12-1 复合运算符示例
#include <stdio.h>
int main()
{
    char c1,c2;
    c1='a';c2='b';
    printf("c1=%c,c2=%c\n",c1,c2);      //输出 c1=a，c2=b
    c1+=3;c2-=32;
    printf("c1=%c,c2=%c\n",c1,c2);      //输出 c1=d，c2=B
    return 0;
}
```

下面简要分析输出结果。

由于 c1、c2 定义为 char 变量，并分别赋值为'a'和'b'，所以第 1 个 printf 输出 c1=a,c2=b。然后，c1 的值加 3，c2 的值减 32，也就是 c1 储存 ASCII 码 97 加 3，变为 100，即字符 d 的 ASCII 码。c2 中的 ASCII 码 98 减 32，变为 66，即字符 B 的 ASCII 码，所以第 2 个 printf 输出 c1=d,c2=B。

3.13 类型转换与混合运算

C 语言中不同类型的数据进行运算称为混合运算。例如：

```
123+'b'*2-12.34
```

当运算对象是不同类型的数据时，系统自动将操作数按照低类型转换为高类型的原则，转换成相同类型后再计算表达式的值。如果参与运算的是变量，则改变的仅仅是参与运算的值，而不是原变量，对原变量无影响。如果这种转换方法仍不能满足编程需要，则也可以使用类型转换运算符对操作数的类型进行强制转换。

1．数据类型自动转换的基本规则与混合运算

不同类型的数据进行运算时，系统会进行类型的自动转换。转换的基本原则是：自动将精度低、表示范围小的运算对象类型向精度高、表示范围大的对象类型转换，以便得到较高精度的运算结果，转换规则如图 3-13-1 所示。

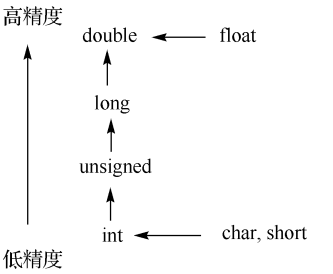

图 3-13-1　转换规则

在表达式中若有 char 或 short 型数据，则一律转换成 int 型参加计算，因为 VC++ 6.0 编译器使用的运算器字长是固定的，例如，表达式'A'–'B'的值为 int 型。

在表达式中若有 float 型数据，则一律转换成 double 型参加计算，如果 x 和 y 都是 float 型，则表达式 x+y 的值为 double 型，这是为了减少处理的复杂度和提高精度。

由于 VC++ 6.0 编译器中整型和长整型长度一致，因此将类型转换做如下总结：

（1）处理整型类（char、short、int、long）数据时，一律转换为 int（long）型处理；

（2）处理浮点类（float、double）数据时，一律转换为 double 型处理；

（3）当整型类与浮点类数据均有时，将整型类数据转换为 double 型处理。

需要注意的是，只有当**整型类与浮点类数据均有时，才将整型类数据转换为 double 型处理**，例如，5/2+0.5 的结果为 2.5，而非 3.0，因为在执行 5/2 时不存在类型转换问题。

2．赋值运算中的类型转换

当赋值运算符两边的运算对象类型不同时，需要进行类型转换。与前面转换规则不同，赋值运算中的类型转换是以左侧变量为准的，因为赋值是用左侧的变量空间保存右侧的数值。赋值运算中的类型转换方法是：将赋值运算符右侧值转换为左侧变量类型之后再赋值。具体转换如下。

（1）将浮点类数据转换为整型类数据时，将舍弃浮点类数据的小数部分，只保留整数部分。例如，a 是 int 型，a=123.65 等价于 a=123。

（2）将整型类数据赋给浮点型变量时，数值不变，只将形式改为浮点类数据，即小数点后自动补 0。例如，a 是 float 型，a=123 等价于 a=123.0。

（3）将一个双精度整型数据赋给浮点型变量时，截取其前面 7 位有效数字，存放到浮点型变量的存储单元中。截取前会进行四舍五入。

（4）将一个浮点型数据赋给双精度型变量时，数值不变，有效数字扩展到 16 位。

（5）将一个整型数值赋给字符变量时，只保留其最低 8 位，高位部分舍弃。

（6）将字符型数值赋给整型变量时，一些编译系统不管其值大小都作为正数处理，而另一些编译系统在转换时，若字符型数据值大于 127，就作为负数处理。

（7）将一个 unsigned 型数据赋给一个占据同样大小存储单元的整型变量时，原值照赋，内部的存储方式不变。

（8）将带符号的整型数据赋给长整型变量时，当两者长度不一致时，只将整型数据放入长整型变量的低字节中，高字节全部补整型数据的符号位，而在 VC++ 6.0 编译器中两者的空间是相同的，没有变化。

3．强制类型转换

一般形式为

```
(数据类型标识符) (表达式)
```

其功能是将“表达式”的值转换成“数据类型标识符”所表示的数据类型。

针对强制类型转换应注意以下几点。

（1）若表达式仅是单个常量或变量，则常量或变量不必用圆括号括起来；若是含有运算符的表达式，则必须加圆括号（因为类型转换运算符的优先级高于双目运算符）。例如：

```
(float) n        /*表示将表达式 n 的值转换成 float 型*/
(int) n+m        /*表示将 n 的值转换成 int 型后再与 m 进行相加运算*/
(int) (n+m)      /*表示先对 n 和 m 进行加运算，再将运算结果转换成 int 型*/
```

（2）数据类型标识符的圆括号是必需的，因为许多类型标识符由多个单词组成，如 unsigned short。

（3）参与运算的常量或变量的类型转换，仅产生一个临时的转换结果参加运算，其原类型和值均不改变。

（4）类型转换会占用系统时间，过多的转换将降低程序的运行效率。因此，在程序设计过程中，应尽量选择好数据类型，减少不必要的类型转换。

3.14 其他运算符

1．取地址运算符（&）

取地址运算符（&）是单目前缀运算符，用来获取变量在内存中的地址。一般形式为

```
&变量名
```

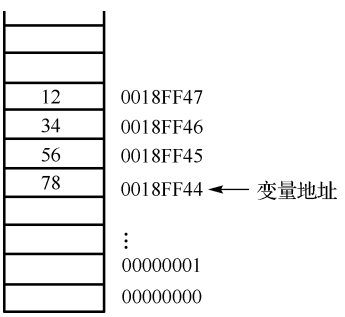

图 3-14-1 内存结构

取地址运算符的运算对象只能是变量，它的运算结果是变量的存储地址，一个变量会占用多字节，而每个字节都有地址，其中最小的那个地址代表变量地址，例如，有 long a=0x12345678; 语句，其内存结构如图 3-14-1 所示。

可以看出 a 占据了 0018FF44 到 0018FF47 4 字节的空间，这 4 字节都有地址，其中最小地址 0018FF44 就是 a 的地址。

人的阅读习惯是从左向右，从上到下的，而计算机存储数字是低位数放低地址，高位数放高地址，如 12 是高位数，放高地址，78 是低位数，放低地址。为了照顾阅读习惯，图中低地址在下面。有时低地址也会画在上面，如后续描述字符串存储部分。

2．长度运算符（sizeof()）

长度运算符（sizeof()）是单目前缀运算符，一般形式为

```
sizeof(数据类型标识符)
```

或

```
sizeof(变量名)
```

长度运算符的运算对象只能是变量名或数据类型标识符，它的运算结果为该变量或该数据类型的长度（占用字节数）。例如，short a; sizeof(short)和 sizeof(a)的值都是短整型数据的长度，即为 2。

3.15　运算符的优先级与结合性一览表

运算符的优先级与结合性一览表如表 3-15-1 所示。

表 3-15-1　运算符的优先级与结合性一览表

运算符类型	运算分量个数	优先级	运算符	含　义	结合性
单体运算符	1	1	( ) [ ] –> .	圆括号 下标运算符 指向结构体成员运算符 结构体成员运算符	从左向右
单目运算符	1	2	! ～ ++ –– – (type) * & Sizeof()	逻辑非运算符 按位取反运算符 自增运算符 自减运算符 负号运算符 类型转换运算符 指针运算符 地址与指针运算符 长度运算符	从右向左
算术运算符	2	3	* / %	乘法运算符 除法运算符 求余运算符	从左向右
		4	+ –	加法运算符 减法运算符	
移位运算符	2	5	<< >>	左移运算符 右移运算符	从左向右
关系运算符	2	6	<、 <=、 >、>=	大小比较关系运算符	从左向右
		7	= = 、 !=	相等、不等运算符	
位运算符	2	8	&	按位与运算符	从左向右
		9	^	按位异或运算符	
		10	|	按位或运算符	
逻辑运算符（！除外）	2	11	&&	逻辑与运算符	从左向右
		12	||	逻辑或运算符	
条件运算符	3	13	?:	条件运算符	从右向左
赋值运算符	2	14	=	赋值运算符	从右向左
复合运算符	2		+=、–=、*=、/=、%=、>>=、<<=、&=、^=、|=	复合运算符	
逗号运算符	2	15	,	逗号运算符	从左向右

所有运算符的优先级共分为 15 级，单体运算符的优先级最高（为 1），逗号运算符的优先级最低（为 15）。

当一个数据旁有多个运算符时，应先执行优先级高的运算，再依次执行优先级低的运算。例如：

```
x|y&b+c
```

该表达式中有 3 个运算符，按优先级的高低为序，依次是+、&和|。所以，该表达式等价于 x|(y&(b+c))。先计算最内层圆括号内的表达式，然后逐层向外扩展。

表 3-15-1 中同一行的运算符有相同的优先级，如果它们出现在同一个表达式中，则按照结合性进行计算。例如：

```
a%b*c
```

该表达式中的运算符%和*具有相同的优先级，这时看它们的结合性是从左向右，所以该表达式等价于(a%b)*c。先计算 a%b，然后将其结果和 c 相乘。

要注意单目运算符中的*不是乘号，是后面要学到的指针。同理，单目运算符中的&，是取地址而不是按位与操作。尽管两者写法相同，但它们实现的运算不同，优先级也不同。

3.16 常用数学函数

完成初等运算使用本章前面介绍的运算符就可以了，而像正弦、余弦、指数、对数等运算是无法用运算符直接完成的，需要使用函数来完成计算。这些函数定义在 math.h 头文件中。因此在使用这些函数时，程序前需要添加语句#include <math.h>，表 3-16-1 列出了常用数学函数。

表 3-16-1 常用数学函数

函数名	函数类型和形参类型	功　能	表达式示例	值
cos	double cos(double x);	计算 cos(*x*)的值	cos(90.0/180*3.14)	0.0
exp	double exp(double x);	求 e^x	exp(1)	2.71828
fabs	double fabs(double x);	求 *x* 的绝对值	fabs(−5.5)	5.5
abs	int abs(int x)	求 *x* 的绝对值	abs(−5)	5
log10	double log10(double x);	求 $\log_{10}(x)$	log10(100)	2.0
log	double log(double x);	求 ln(*x*)	log(2.718282)	1
pow	double pow(double x, double y);	计算 x^y 的值	pow(2,3)	8.0
sin	double sin(double x);	计算 sin(*x*)的值	sin(90.0/180*3.14)	1.0
sqrt	double sqrt(double x);	计算 *x* 的平方根	sqrt(4)	2.0
tan	double tan(double x);	计算 tan(*x*)的值	tan(45.0/180*3.14)	1.0

以 sin 函数为例，double sin(double x);函数名前的 double 表示函数返回值类型，即计算结果为 double 型，圆括号内的 double 表示参数类型也是 double 型，当代入数值不是 double 型时，会进行自动转换，例如：

```
a=sin(30.0/180*3.14);          /*计算 30°的正弦值，a 为 0.499*/
```

fabs 函数和 abs 函数，两者的功能都是求绝对值，fabs 函数返回值为 double 型，fabs(–5.5)返回值为 5.5；abs 函数返回值为 int 型，abs(–5.5)返回值为 5。

log 函数，是数学函数 ln，也就是求以 e 为底的对数，以 10 为底的对数要写为 log10。

更多的数学函数见本书附录部分。

在此需要说明的是，三角函数使用弧度而非度数，计算时要将度数转换为弧度代入，示例中的 30°之所以要写成 30.0，是因为在 C 语言中，30/180 的结果是 0。使用三角形斜边长度和锐角度数，求两个直角边长度的程序示例如下：

```
//程序 3-16-1 使用三角形斜边长度和锐角度数，求两个直角边长度
#include "stdio.h"
#include "math.h"
int main(){
    float a,b,c,angle;
    printf("请输入斜边长度  锐角度数\n");
    scanf("%f%f",&c,&angle);
    a=c*sin(angle/180*3.14);
    b=c*cos(angle/180*3.14);
    printf("两个直角边长度:%f,%f\n",a,b);
    return 0;
}
```

3.17　本章小结

不同形式的表达式是通过不同的运算符构成的。一个常量、一个变量都可以看成是一个简单表达式。简单表达式通过运算符连接可以构成复杂表达式。

运算符可以分为单目运算符、双目运算符和三目运算符。

在算术表达式中，不同类型的数据可以混合运算。混合运算时，要按照一定的规则先将两个不同类型的数据转换成统一的类型后再进行运算。整个转换过程是自动的，而且是逐步进行的，转换的原则是从低类型向高类型转换。

每个运算符都规定了它的优先级，相同优先级的运算符的运算次序由它的结合性确定。C 语言没有逻辑常量 true 和 false，因此用 1 表示 true，用 0 表示 false。逻辑运算中非零值等同于 1，逻辑运算还有其特别的短路现象。

高等算术运算是通过调用函数实现的，使用时要添加语句#include <math.h>。

习题 3

3-1　选择题

（1）设 int a=3,b = 4, c = 5，下面表达式值为 0 的是（　　）。

A．'a'&&'b'　　B．a<=b

C．a||b+c&&b-c　　D．!((a<b)&&!c||1)

（2）若有以下定义：char a; int b; float c; double d;，则表达式 a*b+d–c 的值的类型为（　　）。

A．float　　B．int　　C．char　　D．double

（3）设有语句 int a = 3;，执行语句 a+=a–= a*a;后，变量 a 的值是（　）。

A．3　　B．0　　C．9　　D．−12

（4）设有语句 int a = 3;，执行语句 printf("%d", –a ++);后，输出的结果是（　），变量 a 的值是（　）。

A．3　　B．4　　C．−3　　D．−2

（5）表达式"25%3*sqrt(16.0)/2"值的数据类型为（　）。

A．float　　B．int　　C．char　　D．double

（6）有声明 int a=2;float b=5,c;，下面表达式合法的是（　）。

A．b%a　　B．(–b)++　　C．a&b　　D．a&&b

（7）有声明 int a,b,c;，下面表达式不合法的是（　）。

A．a=b+c　　B．a=b=c　　C．a+b=c　　D．a+(b=c)

（8）有 int a=1,b=0;，下面表达式值不为 1 的是（　）。

A．a++　　B．++b　　C．a|b　　D．!a

（9）有 int a=1,b=2;，下面表达式值为 0 的是（　）。

A．a&&b　　B．a&b　　C．a^b　　D．a%b

（10）有 int a=1,b=2;，下面合法的表达式是（　）。

A．(a+1)(b–3)　　B．a–/b　　C．2a　　D．a–b++

3-2　若 int a=7,b=2;，则下面表达式的值是什么？

（1）a*b

（2）a/b

（3）a–b

（4）a%b

（5）a&&b

（6）a&b

（7）a||b

（8）a|b

（9）!a

（10）～a

（11）a<b

（12）a<<b

3-3　若 int x=3,y=z=4，下列各式的结果是什么？

（1）(z>=y>=x)?1:0

（2）z>=y&&y>=x

（3）x<y?x:y

（4）x<y?x++:y++

（5）z+=x>y?x++:y++

3-4　写出下面程序运行后的结果。

（1）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main(){
   int x;
   x=-3+4*5-6;
   printf("x1=%d,",x);
   x=3+4%5-6;
   printf("x2=%d,",x);
   x=-3*4%-6;
   printf("x3=%d\n",x);
   return 0;
}
```

（2）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main() {
   int i,j;
   float x,y;
   i=2;
   j=3;
   x=4.0;
   y=1.0+i/j+x+1.5;
   printf("y=%f\n",y);
   return 0;
}
```

（3）下面程序运行后的结果为_____。

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

（4）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main(){
   int x,y,z;
   x=3;y=2;z=0;
   x+=y+=z;
   printf("%d,",x<y);
   z=y=x++*3;
   printf("%d %d,",y>=z,x);
   x=y>z>=5;
   printf("%d %d %d\n",x,y,z);
   return 0;
}
```

（5）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main(){
   int x,y,z;
   x=3;y=2;z=0;
   x=x&&y||z;
   printf("(1)%d,",x);
   printf("(2)%d,",x|| !y&&z);
   x=03;y=02;z=01;
   printf("(3)%d,",x|y&z);
   printf("(4)%d,",x|y&~z);
   printf("(5)%d,",x^y&~z);
   x=1;y= -1;
   x=x<<3;
   printf("(6)%d,",x);
   y=y<<3;
   printf("(7)%d\n",y);
   return 0;
}
```

（6）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main(){
   int sum=10,cap;
   cap=sum++,cap++,++cap;
   printf("%d\n",cap);
   return 0;
}
```

（7）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main(){
   int x,y,z;
   x=y=z=1;
   x+=y+=z;
   printf("(1)%d,",x<y?y: x);
   printf("(2)%d,",x<y?x++: y++);
   printf("(3)x=%d,y=%d,",x,y);
   printf("(4)%d,",z+=x<y?x++: y++);
   printf("(5)x=%d,y=%d,z=%d,",x, y, z);
   x=5;
   y=z=6;
   printf("(6)%d,", (z>=y>=x)? 1: 0);
   printf("(7)%d\n", z>=y&&y>=x);
   return 0;
}
```

（8）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main(){
```

```
    int num,sum,t;
    num=sum=7;
    sum=num++;
    t=++sum+num++;
    printf("%d,%d",sum,num);
    return 0;
}
```

（9）下面程序运行后的结果为_____。

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

（10）下面程序运行后的结果为_____。

```
#include <stdio.h>
int main(){
    int a=-1, b=4, k;
    k=(a++<=0)&&(!(b--<=0));
    printf("%d %d %d\n", k, a, b);
   return 0;
}
```

3-5　将下面语句组进行简写。

（1）将 int a; int b;改成一条语句_____。

（2）将 x=0; y=0;改成一条语句_____。

（3）将 x=x+y;改用复合运算符_____。

（4）将 x=x+1;改用单目算术运算符_____。

（5）将 y=x; −x;改成一条语句_____。

3-6　试编写一个程序，实现输入两个整型数据，输出两个数的和。

3-7　试编写一个程序，要求从键盘输入两个整数，分别存入 a、b 变量，然后将 a、b 中的值交换后再输出。

3-8　已知三角形的三边长 *a*、*b*、*c*，求三角形面积的公式为

$$S=\frac{1}{2}(a+b+c),\ \text{area}=\sqrt{s(s-a)(s-b)(s-c)}$$

要求编写程序，实现从键盘输入 *a*、*b*、*c* 的值，计算并输出三角形的面积。

提示：程序运行时应保证输入的 *a*、*b*、*c* 值满足三角形成立的条件，这样计算得到的三角形面积才有意义。另外，将面积计算的数学公式写成合法的 C 语言表达式为

```
area=sqrt(s*(s-a)*(s-b)*(s-c))
```

注意，写成

```
area=sqrt(s(s-a)(s-b)(s-c))
```

是错误的。

将数学公式 $S=\frac{1}{2}(a+b+c)$ 写成表达式

```
s=0.5*(a+b+c)
```

或

```
s=1.0/2*(a+b+c)
```

都是正确的。而写成

```
s=1/2*(a+b+c)
```

虽然是合法的，但结果是错误的，请读者思考为什么。