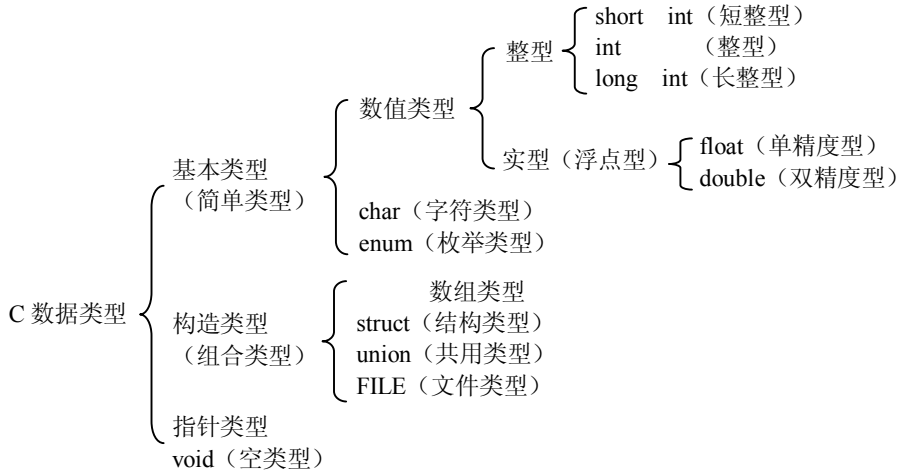
第 2 章　C 语言的数据类型与基本操作

数据是指能够输入计算机并被计算机处理的数字、字母和符号的集合，人们看到的景物和听到的声音，都可以用数据来描述。数据与操作是程序的两个要素，数据处理是计算机的用途之一。C 语言的数据结构是以数据类型形式出现的。数据类型不同，它所表达数据的范围、精度和所占据存储空间的大小均不相同。



无论什么类型数据都有常量和变量两种表现形式。任何数据都必须且只能以指定的类型和形式在程序中出现。本章只介绍基本类型中的整型、实型、字符型数据。关于枚举、构造类型（指针、数组、结构体、共用体）等类型的数据将在后面不同的章节中予以介绍。

2.1　常量与变量

2.1.1　常量

在程序运行过程中，其值不改变的量称为常量。在 C 语言中，常量有不同的类型，分别是整型常量（int）、实型常量（float 和 double）、字符常量（char）、字符串常量。整型常量也可分为短整型（short int）、长整型（long int）和无符号型（unsigned int）等。

1．整型常量

整型常量通常是指数学上的整数，分为十进制、八进制和十六进制常量。

（1）十进制整型常量形式，如 15，19，123，-345 等。

（2）八进制整型常量形式，以数字 0 开头，如：05，012，0123 都是八进制，它们分别代表十进制数的 5，10，83。

（3）十六进制整型常量形式，以数字 0 和字母 x（或大写字母 X）开头。如：0x10，0Xff，0X8 均为十六进制整数，它们分别代表十进制数的 16，255，8。

注意：①八进制数只能用合法的八进制的数字表示，各位数字分别是 0～7。十六进制数只能用合法的十六进制数字表示，十六进制各位上的数字分别是 0～15，其中 10、11、12、13、14、15 既可以用大写字母 A，B，C，D，E，F 表示，也可以用小写字母 a，b，c，d，e，f 表示。②常量的长度及表示数据的长度范围，通常与机器类型有关。③八进制和十六进制只是数的书写方式，对数的存储方式不会产生影响，因为整数都是以二进制形式存储的，跟表示方式无关，④C 语言中，一般只使用无符号的八进制和十六进制数，而不使用有符号的八进制和十六进制数，因此没有专门的无符号八进制和十六进制标识符。

2. 实型常量

实型常量也称数值常量，它有正值和负值之分，只能用十进制形式表示。实型常量可以用小数形式或指数形式表示，如：345，.345，345.0，3.14159，1.5e5，2.9e-7 等。后两个数都采用了指数形式，分别表示 1.5×10^{5} 和 2.9×10^{-7}。实型常量不分单精度型和双精度型，但可以赋给一个 float 型或 double 型变量。

注意：指数形式的浮点常量 E 或 e 前面必须有数字，E 或 e 后面必须为整数。因此，E8，6e7.8 都是不合法的浮点常量。

3. 字符常量

它是用一对单撇号括起来的一个字符，如'a'，'A'，'？'，'#'。注意，单撇号只是字符与其他部分的分隔符，或者说是字符常量的定界符，不是字符常量的一部分，当输出一个字符常量时不输出此撇号；也不能用双撇号代替单撇号，如"a"不是字符常量。

注意：撇号中的字符不能是撇号或反斜杠。如'''或'\'都不是合法的字符常量。

4. 转义字符

在程序设计中，有些字符和命令无法通过键盘直接输入，因而借用一个符号或者数字来表示，C 语言用反斜杠“\”来表示转义字符的起始符，转义字符系列有两类：一类是字符转义系列，另一类是数字转义系列。C 语言用反斜杠“\”来表示转义字符，有三种表示方法：

（1）用反斜杠开头后面跟一个字母代表一个控制字符。

（2）\\代表“\”，用\'代表引号字符。

（3）转义字符除用来形成一个外设控制命令外，还用来输出不能直接从键盘上输入或不能用字符常量书写出的 ASCII 字符。

字符转义系列如表 2-1 所示。

表 2-1　字符转义系列

字符形式	功能
\n	换行
\t	横向跳格（即跳到下一个输出区）
\v	竖向跳格
\b	退格
\r	回车

续表

字符形式	功能
\f	走纸换页
\\	反斜杠字符“\”
\'	单引号字符

数字转义系列如表 2-2 所示。

表 2-2　数字转义系列

字符形式	功能
\ddd	1～3 位八进制所代表的字符
\xhh	1～2 位十六进制所代表的字符

作为字符常量使用，转义字符必须用一对单引号括起来。例如'\n'、'\101'和 '\ x41'。

【例 2.1】

```
//打印人民币符号￥
main()
{
      printf ("Y\b=\n");
      return 0;
}
```

该程序运行时先打印一个字符 Y。这时打印头已走到下一个位置，用控制代码\b 使打印头回退一格，即回到原先已打印好的 Y 位置再打印字符“=”，两个字符重叠形成人民币符号￥。当然，这一输出只能在打印机上实现，而不能在显示器上实现。因为显示器无此重叠显示功能（在显示后一字符时原来此位置上的字符消失）。

转义字符除用来形成一个外设控制命令外，还用来输出不能直接从键盘上输入或不能用字符常量书写出的 ASCII 字符。这时要在反斜杠“\”后跟一个代码值，这个代码值最多用 3 位八进制数（不加前缀）或两位十六进制数（以 x 作前缀）表示。

【例 2.2】

```
main ()
{
     char ch;
     ch='\362';   //将八进制数 362 的 ASCII 字符赋给 ch
     printf ("%c\n",ch);
     printf ("china\n\101\t\\\n");
     printf ("%c,%c,%c,%c,%c\n", '\x40', '\101', '\\', '\'', '\"');
     return 0;
}
```

在 IBM PC 机上运行可在显示屏上输出：

```
≥
china
```

A \
@ , A , \ , ' , "

程序有三个输出语句，先执行“printf ("%c\n",ch);”语句，将'\362'以字符的形式输出符号“≥”；第二个“printf ("china\n\101\t\\\n");”语句有 5 个转义字符，分别是'\n'、'\101'、'\t'、'\\'和'\n'。输出“china”后遇到转义字符'\n'换行，换行后遇到转义字符'\101'，输出'A'后遇到转义字符'\t'，水平移到下一个制表位置，遇到转义字符'\\'，输出符号“\”，再遇到'\n'进行换行。

第三个输出语句，'\x40'是用十六进制数 40 表示的转义字符，其十进制数为 64，字符为@；'\101'是用八进制数 101 表示的转义字符，其十进制数为 65，字符为 A，见附录 A。'\\'表示字符\；'\"'表示字符'；'\'"表示字符"；%c 表示按字符输出。

当然，也可以用转义字符输出其他字符，如：

\363 或 \ xf3　　表示字符≤

\012 或 \ x1A　　表示字符→

\100 或 \ x40　　表示字符 @

5. 字符串常量

字符串常量简称字符串，是用双撇号括起来的零个或多个字符系列。如："hello"，"Programming in C"，"A"，"a"，" "、"C 语言"等。

字符串以双撇号为定界符，但双撇号并不属于字符串。要在字符串插入撇号，应借助转义字符。例如要处理字符串"I say: 'Goodbye! ' "时，可以把它写为"I say:\ 'Goodbye!\ ' "。

字符串中的字符数称为该字符串的长度。字符串常量在机器内存储时，系统自动在字符串的末尾加一个“字符串结束标志”，它是转义字符“\0”。如字符串"hello"在内存中存储为：

h	e	l	l	o	\0

也就是说字符串在存储时要多占用一个字节来存储“\0”。实际上每个字符都是用其 ASCII 代码来存储的。“\0”的代码为 0，它的含义为“空操作”，即不产生任何动作，只起“标记”作用。上面的字符串实际上存储形式为：

104	101	108	108	111	0

要特别注意字符常量与字符串常量的区别。如'A'是一个字符常量，而"A"则是一个字符串常量。它们的存储形式分别为：

'A' ⟶ | 65 |　　"A" ⟶ | 65 | 0 |

" "表示一个空字符串，在内存中占用一个字节，即：

" " ⟶ | 0 |

6. 符号常量

除了上面介绍的直接常量外，C 语言也可以用标识符代替常量。

【例 2.3】 符号常量的使用。

```
#define PRICE   30
main()
{
    int num,total;
```

```
    num=10;
    total=num*PRICE;
    printf ("total=%d\n", total);
    return 0;
}
```

程序中用# define 命令行定义 PRICE 代表常量 30，此后凡在本文件中出现的 PRICE 都代表 30，可以和常量一样进行运算，程序运行结果为：total=300。

有关#define 命令行的详细用法参见第 6 章。

这种用一个标识符代表一个常量的，称为符号常量，即标识符形式的常量。请注意符号常量不同于变量，它的值在其作用域（在本例中为主函数）内不能改变，也不能再被赋值。如再用赋值语句给 PRICE 赋值是错误的。如：

```
PRICE=40;
```

习惯上，符号常量名用大写，变量名用小写，以示区别。使用符号常量的好处是：

（1）含义清楚。如上面的程序中，看程序时从 PRICE 就可知道它代表价格。因此定义符号常量名时应考虑“见名知意”。在一个规范的程序中不提倡使用很多的常数，如：sum=15*30*23.5*43。在检查程序时往往搞不清各个常数究竟代表什么。应尽量使用“见名知意”的变量名和符号常量。

（2）在需要改变一个常量时能做到“一改全改”。例如在程序中多处用到某物品的价格，如果价格用常数表示，则在价格调整时，就需要在程序中做多处修改，若用符号常量 PRICE 代表价格，只需改动一处即可。如：

```
# define PRICE 35
```

在程序中所有以 PRICE 代表的价格就会一律自动改为 35。

7. 布尔类型

（1）C89 中的布尔值。多年以来，C 语言一直缺乏布尔类型，C89 标准中也没有定义布尔类型，而在程序中需要存储成“真”或者成“假”的值，一般解决的方法就是先声明一个 int 型变量，然后将其赋值 1 或 0。

```
int flag;
flag=0;
……
flag=1;
```

这种方法只是借助“1”或“0”来表示“真”或“假”。

（2）C99 中的布尔值。C99 标准中提供了_Bool 型，布尔变量可以声明为：

```
_Bool   flag;
```

_Bool 是无符号的整型变量，C99 中除了_Bool 类型的定义外，还提供了一个新的包含命令<stdbool.h>，在<stdbool.h>中提供了 true 和 false 的值分别代表 1 和 0，使得_Bool 实现起来非常方便。

2.1.2　变量

变量指在程序运行中其值可以发生变化的量。变量在内存中占据一定的存储单元，该存储单元中存放变量的值。变量通常用来保存程序运行中的输入数据，计算获得的中间结果和最终结果。变量的命名规则和用户标识符相同，给变量取名时，为了便于理解程序，一般都采用

“见名知意”的原则。

1. 变量的声明

C 语言规定，在程序中用到的每个变量都要声明它们属于哪一种类型。

变量声明的格式为：

```
数据类型符 变量名 1,变量 2,……,变量名 n;
```

例如：

```
int x;
int y;
```

或等效为：

```
int x,y;
```

注意：①定义变量的语句必须以“;”号结束，在一个定义语句中也可以同时定义多个变量，变量之间用“,”隔开。②对变量的定义可以在函数体之外，也可以在函数体或复合语句中。

例如：

```
int a,b;
char ch;
```

2. 变量的初始化

C 语言允许在声明变量的同时对其初始化，例如：

```
int sum=0;              /*说明变量 sum 的类型为整型，初始值为 0*/
float pi=3.1416;
char c='w';
```

也可以对声明的变量的一部分初始化，如：

```
int i, sum=0,j;
```

【例 2.4】 整型变量的定义和引用。

```
#include"stdio.h"
main()
  {
     int x ,y,z,s;                  //基本整型变量定义，一次可定义一个或多个变量
     x=3;y=4;z=5;                   //整型变量的引用
     s=x+y+z;                       //整型变量的引用
     printf("\ns=%d",s);
     return 0;
  }
```

程序运行结果如下：

```
S=12
```

注意：

（1）不同类型的数据在内存中占据不同长度的存储区，而且采用不同的表示方式（指数据在机器内部的表示方式）。例如，在 Visual C++ 6.0 的环境下，用 4 个字节存放一个整数，以定点形式存放；而用 4 个字节存放一个实数，以指数形式存放。在程序中引用一个变量，实际上是对指定的存储空间的引用，必须先开辟（分配）存储空间才能引用它。

（2）一种数据类型对应着一个值的范围。例如，一个有符号的短整型数据的存储范围为 -32768 ~ 32767 之间，一个无符号短整型数据的存储范围为 0 ~ 65535 之间。具体情况如表 2-2 所示。

（3）一种数据类型对应着一组允许的操作。例如，对整数数据可以进行“求余”运算（5%2 的值为 1，即 5 除以 2 的余数为 1），而实型数不能进行求余运算。数值型数据可以进行四则运算，而结构体型数据就不能进行四则运算。

总之，在引用变量之前必须先用声明语句指定变量的类型，这样在编译时就会根据指定的类型分配其一定的存储空间，并决定数据的存储方式和允许操作的方式。

注意：赋值号“=”左边只能是变量，不能是常量或表达式，因为只有变量才有存储空间，才能接收赋值。

【例 2.5】 定义变量，并给变量赋值。

```
#include "stdio.h"
void main()
{
    unsigned int y;
    char ch;
    y = 0x1afe1f3;
    ch = 'A';
    printf("%x , %x , %c\n", y, ch, ch);
    return 0;
}
```

程序运行结果如下：

```
1afe1f3, 41, A
```

说明：变量赋值后的结果如图 2-1 所示，为便于分析用了十六进制形式。内存中存储的是二进制形式，如 f3 在内存中是 11110011。%x 表示对应的表达式按十六进制输出。

地址	存贮单元	变量名
…		
1000	f3	y
1001	e1	
1002	af	
1003	1	
1004	41	ch
1005		
1006		
…		

图 2-1　给变量赋值

一个变量定义后，涉及四个相关的内容：变量名、变量的存储空间及内容、变量的地址、变量的作用域，在例 2.5 中，变量 y 的变量名是 y，变量 y 的存储空间为 4 个字节，变量 y 的地址是 1000，是 y 的存储空间的首字节的地址。进一步地理解详见第 7 章和第 9 章。

2.2 标识符和关键字

任何一种程序设计语言都有自己的一套语法规则以及由这些语法规则基本符号按照语法规则构成的各种语法成分。例如，前面讲到的常量、变量、表达式、语句和函数等，基本的语法单位是指具有一定语法意义的最小语法成分，C 语言的基本语法单位从编译程序的角度讲即为词法分析单位，习惯上把它称为“单词”。组成单词的基本符号称为 C 语言的字符集，C 语言的字符集由机器系统所使用的字符集决定，大多数 C 语言的实现及标准 C 使用的字符集是 ASCII 字符集（见附录 A）。

语言的语法单位分为六类：标识符、关键字、常量、字符串、运算符及分隔符。

1. 标识符

标识符是给程序中的实体（变量、常量、函数、数组及结构体）以及文件所起的名字。简单地说，标识符是一个名字，可以由程序设计者指定，也可以由系统指定。语言中的标识符命名规则为：

（1）以字母（A～Z、a～z）和下划线“_”开头，由字母、数字字符（0～9）和下划线组成的字符系列。下面的标识符是合法的：

a　b1　ab　x13　name_students　filel_buf　SIZE_PI

下面不是合法的 C 标识符：

456th	（数字开头）
cade-of-moves	（含有非法的“-”）
piece　flag	（含有非法字符空格）
$123.4	（$开头，含有非法字符）
w.w	（含有非法字符“ .”）
a&b	（含有非法字符“&”）

（2）系统内部使用了一些用下划线开头的标识符（如_fd，_cleft，_mode），为了防止与用户已定义的标识符冲突，建议用户在定义标识符时尽量不要用下划线开头。

（3）C 语言对标识符的长度无规定，有的 C 版本规定的长度可能很短，具体使用时应以所使用 C 版本的规定为准。

（4）C 语言区分大写字母和小写字母，将大写字母和小写字母认为是两个不同的字符。如 TOTAL 和 total 被认为是两个标识符。

在定义标识符时，建议遵循以下原则：

1）尽量做到“见名知意”，以增加程序的可读性。如 sum，area，score，day，name 等。

2）变量名和函数名用小写，符号常量用大写。

3）在容易出现混淆的地方尽量避免使用容易认错的字符。如：

0（数字）————O（大写字母）————o（小写字母）

1（数字）————I（i 的大写字母）————l（L 的小写字母）

2（数字）————Z（大写字母）————z（小写字母）

例如：no 和 nO，11 和 ll 等都是非常容易给阅读程序者造成困惑的。

2. 关键字

关键字是由编译程序预定义的、具有固定的含义的、在组成结构上均由小写字母构成的标识符。因此，用户不能用它们来作为自己定义的常量、变量、类型或函数的名字。所以，关键字又称为保留字，即被保留作为专门用途的特殊标识符。C 语言中用户自己定义的标识符的含义不再包括关键字。

ANSI C 标准中定义了 32 个关键字，C99 新增了 5 个关键字，请参阅附录 B。

下面列出了 C 语言的一些主要关键字：

数据类型：char, int, float, double, void

输入输出：scanf, printf, getchar, putchar, getch, getche

语句：if, else, switch, case, default, break, while, for, do, continue, goto, return

运算符：sizeof

因此，前面程序中的 int，float，return 是关键字而不是标识符。各关键字的作用将在后续的各章节中逐一介绍。

2.3　整型数在计算机中的存储方式

计算机中，内存储器的最小存储单位称为“位”（bit），每一个位中或存放 0，或存放 1，因此称为二进制位。大多数计算机把 8 个二进制位组成一个“字节”（byte），并给 4 个字节分配一个地址。若干字节组成一个“字”（word），用一个字来存放一条机器指令或一个数据。一个字含多少个字节随机器而不同。如果一台计算机系统字长为 4 个字节（一个字节为 8 个二进制位），就称这台计算机的字长为 32 位。

一个字节有 8 个二进制位，本书中把最右边的一位称为最低位，把最左边的一位称为最高位。在 Visual C++ 6.0 环境下的 C 编译系统中，一个 int 数据通常用 4 个字节（32 位二进制）存放。其中最高位（最左边的一位）用以存放整数的符号。若是正整数，最高位置 0；若是负整数，最高位置 1。因此，从最高位就立刻能判别出存放的整数是正整数还是负整数。

C 语言中，数据在内存中的存储涉及原码、反码和补码的问题，下面通过例子来说明：

【例 2.6】利用 8 位二进制定点数表示法计算

$$(-50)_{10}+(+33)_{10}$$

首先将这两个十进制数转换成 8 位二进制定点数，然后再对这两个二进制数相加，即

```
   10110010    (-50)10 的 8 位二进制定点数
+) 00100001    (+33)10 的 8 位二进制定点数
------------
   11010011
```

最后得到：

$$(-50)_{10}+(+33)_{10}=(10110010)_2+(00100001)_2=(11010011)_2=(-83)_{10}$$

这个结果显然是错误的。正确的结果应为$(-17)_{10}$，二进制表示为$(10010001)_2$。

如果将这个问题转化为减法运算，即：

$$(-50)_{10}+(+33)_{10}=(+33)_{10}-(+50)_{10}$$

再作减法有：

```
   00100001    (+33)10的八位二进制定点数
-) 00110010    (+50)10的八位二进制定点数
-------------
   11101111
```

最后得到

$(-50)_{10}+(+33)_{10}= (+33)_{10}-(+50)_{10} =(00100001)_2-(00110010)_2=(11101111)_2=(-111)_{10}$

显然，这个结果也不正确。

由这个例子可以看出，直接对二进制定点数进行加减运算可能会得到错误的结果。因此，为了便于运算，在计算机中，二进制定点数一般不直接用以上的方法表示。

下面介绍二进制定点数在计算机中的三种表示法：原码、反码和补码。

（1）原码。把整数的绝对值用二进制表示，最高位用于表示符号，0 表示正数，1 表示负数。

$(+50)_{10}$的 8 位二进制的原码为$(00110010)_2$

$(-50)_{10}$的 8 位二进制的原码为$(10110010)_2$

$(+33)_{10}$的 8 位二进制的原码为$(00100001)_2$

在二进制原码中，使用的二进制位越多，所能表示的数的范围就越大。例如：

$(+156)_{10}$的 16 位二进制的原码为$(0000000010011100)_2$

$(-156)_{10}$的 16 位二进制的原码为$(1000000010011100)_2$

从前面的例子可以看出，采用原码表示后，符号不同的两个数不能直接相加，或者说两个同号数不能直接相减。

（2）反码。反码表示法规定：正数的反码和原码相同；负数的反码是对该数的原码除符号位外各位取反（即将“0”变为“1”，“1”变为“0”）。例如：

$(+50)_{10}$的 8 位二进制的原码为$(00110010)_2$

$(+50)_{10}$的 8 位二进制的反码为$(00110010)_2$

$(-50)_{10}$的 8 位二进制的原码为$(10110010)_2$

$(-50)_{10}$的 8 位二进制的反码为$(11001101)_2$

容易验证，一个数的反码的反码就是原码本身。

（3）补码。补码表示法规定：正数的补码和原码相同；负数的补码是在该数的反码的最后（即最右边）一位上加 1。例如：

$(+50)_{10}$的 8 位二进制的原码为$(00110010)_2$

$(+50)_{10}$的 8 位二进制的补码为$(00110010)_2$

$(-50)_{10}$的 8 位二进制的原码为$(10110010)_2$

$(-50)_{10}$的 8 位二进制的反码为$(11001101)_2$

$(-50)_{10}$的 8 位二进制的补码为$(11001110)_2$

容易验证，一个数的补码的补码还是原码本身。

引入了补码以后，计算机中的加减运算都可以用加法来实现。并且，两数的补码之“和”等于两数“和”的补码。在采用补码运算时，符号位也当作一位二进制数一起参与运算。

【例 2.7】 采用二进制补码计算$(33)_{10}-(50)_{10}$和$(33)_{10}+(50)_{10}$。

由于

$$(33)_{10}-(50)_{10}=(-50)_{10}+(33)_{10}$$

因此

```
    11001110      (-50)10的 8 位二进制的补码
+)  00100001      (+33)10的 8 位二进制的补码
------------
    11101111
```

最后得到：

$(33)_{10}-(50)_{10}=(-50)_{10}+(33)_{10}=(11001110)_{补}+(00100001)_{补}$

$=(11101111)_{补}$（对补码除符号位外求反末位加 1 后得到）

$=(-17)_{10}$

下面计算$(33)_{10}+(50)_{10}$

```
    00100001      (+33)10的 8 位二进制的补码
+)  00110010      (+50)10的 8 位二进制的补码
------------
    01010011
```

最后得到：

$(33)_{10}+(50)_{10}=(00100001)_{补}+(00110010)_{补}$

$=(01010011)_{补}$（正数的补码与原码相同）

$=(83)_{10}$

这个结果显然也是正确的

由此可以看出，在计算机中，所有的加减运算都可以统一化成补码的加法运算，并且其符号位一起参与运算，结果为补码，这种运算既可靠又方便。对于反码，它只起到由原码转换为补码的中介作用。

2.4　有符号的数据类型和无符号的数据类型

在内存中存储整数时，一般用最高位（即最左边一位）表示符号，以 0 表示正数，以 1 表示负数。数值是以补码形式存放的。有关补码的概念前面已经介绍过，此处不再赘述。

前面谈到的 short，int，long 型数据，都是指带符号的，也就是含有 signed（带符号）。因此，int 与 signed 等价；short 与 signed short 等价；long 与 signed long 等价。

C 语言还允许使用无符号（unsigned）的整型数据，它将二进制形式的最左位不作为符号，而与右边各位一起用来表示数值。也就是说，如果定义一个数据类型为 unsigned int，则它只能存放正数而不能存放负数。

字符型的数据也有 signed 和 unsigned 两种类型。字符型的数据占一个字节（8 位）。标准的 ASCII 字符的允许范围为 0～127，用 7 位表示就可以了，最左边一位补 0。

数值型和字符型数据总共有 11 种，如表 2-2 所示。

表 2-2　VC ++ 6.0 中基本数据类型的数据所占空间与取值范围

类型	类型标识符	长度（Byte）	取值范围及精度
字符型	char（字符型）	1	-128～127
	unsigned char（无符号字符型）	1	0～255
	[signed]char（有符号字符型）	1	-128～127
整型	int（整型）	4	-2147483648～2147483647
	unsigned int（无符号整型）	4	0～4294967295
	[signed] int（有符号整型）	4	同 int
	short [int]（短整型）	2	-32768～32767
	unsigned short [int]（无符号短整型）	2	0～65535
	[signed] short [int]（有符号短整型）	2	同 short [int]
	long　[int]（长整型）	4	-2147483648～2147483647
	[signed] long [int]（有符号长整型）	4	-2147483648～2147483647
	unsigned long [int]（无符号长整型）	4	0 ~ 4294967295
实型	float（单精度浮点型）	4	-3.4×10^{-38}～3.4×10^{38}，6 位有效数字
	double（双精度浮点型）	8	-1.7×10^{-308}～1.7×10^{308}，16 位有效数字

说明：

（1）表中“类型标识符”列中的[ ]的内容是可选的。也就是说，有这部分内容和没有这部分内容的作用是一样的。例如 signed char 和 char 作用相同；signed short int，signed short，short，short int 四者作用相同，其余类推。

（2）实型数据没有 unsigned 和 signed 之分，均带符号。

（3）本表中的数据范围是依据 Visual C++ 6.0 而言的，也就是说数据存储范围的大小与机器系统有关。具体怎样测试数据在机器中的存储长度将在 2.5.3 节中介绍。

（4）C99 中，提供了两个额外的标准整数类型：long long int 和 unsigned long long int，增加这两个整数类型的原因：一是为了满足日益增长的对超大型整数的需求；二是为了适应支持 64 位运算的新处理器的能力。这两个 long long 类型要求至少 64 位宽，所以 long long int 类型的范围通常为-2^{63}（-9 223 372 036 854 775 808）～$2^{63}-1$（9 223 372 036 854 775 807），unsigned long long int 类型值的范围为 0～$2^{64}-1$（18 446 744 073 709 551 615）。

在 C99 中以 LL 或 ll（两个大小写字符要一致）结尾的整数表示 long long int 型。如果在 LL 或 ll 前面或者后面增加 U（或 u），则该整数表示为 unsigned long long int 型。

2.5　运算符和表达式

运算（即操作）是对数据的加工。C 语言对数据的基本操作和处理几乎全是由运算符来完成的。这些符号称为运算符或操作符。被运算的对象——数据称为运算量或操作数。运算符执

行对操作数的各种操作，按操作数的数目分类有单目（一元）运算符、双目（二元）运算符和三目（三元）运算符；按运算符的功能分类有算术运算符、关系运算符、逻辑运算符、自增与自减运算符、赋值运算符和条件运算符。此外，表示数组下标的“[ ]”、表示函数调用的“（）”、表示顺序求值的“,”，以及类型强制转换符“（类型）”也都作为运算符看待。具体归纳如下：

（1）算术运算符　　（+、-、*、/、++、--）
（2）关系运算符　　（>、<、==、>=、<=、!=）
（3）逻辑运算符　　（!、&&、||）
（4）位运算符　　（<<、>>、|、∧、&）
（5）赋值运算符　　（=及其扩展赋值运算符）
（6）条件运算　　（?:）
（7）逗号运算符　　（,）
（8）指针运算符　　（*和&）
（9）求字节运算符　　（sizeof）
（10）强制类型转换运算符　（类型）
（11）分量运算符　　（.、->）
（12）下标运算符　　（[　]）
（13）其他　　（如函数调用运算符）

表达式描述了对哪些数据，以什么顺序以及施以什么样的操作。它由运算符与运算量组成，运算量可以是常量，也可以是变量，还可以是函数，例如：a+3、t+sin(a)、x=a+b 等都是表达式。

表达式的运算规则是由运算符的功能和运算符的优先级与结合性所决定的。为使表达式按一定的顺序求值，编译程序时所有运算符分成若干组，按运算符执行的先后顺序为每组规定一个等级，称为运算符的优先级，优先级高的运算符先执行运算。处于同一优先级的运算符顺序称为运算符的结合性，运算符的结合性有从左到右和从右到左两种顺序，简称左结合和右结合。C 语言的所有运算符及优先级和结合性见附录 C。

2.5.1　赋值运算符和赋值表达式

1. 赋值运算符与赋值表达式

“=”就是赋值运算符。

赋值表达式：由赋值运算符组成的表达式称为赋值表达式。它的一般形式：

<变量>=<表达式>

如：x=5;

赋值表达式的求解过程：将赋值运算符右侧的表达式的值赋给左侧的变量。赋值的含义是将赋值运算符右边的表达式的值存放到左边变量名标识的存储单元中。x=5 就是将整型数据 5 赋给变量 x 的存储单元中，语句“x=10+y;”执行赋值运算（操作），将 10+y 的值赋给变量 x。

说明：

（1）赋值运算符左边必须是变量，右边可以是常量、变量、函数调用或由常量、变量、函数调用组成的表达式。例如：“x=10;y=x+10;y=func();”都是合法的赋值表达式，12=a、2*a=3*5+55 都是不合法的赋值表达式。

（2）赋值符号“=”不同于数学中的等号，它没有相等的含义，C 语言中 x=x+1 是合法的，而数学上不合法。

（3）赋值运算时，当赋值运算符两边的数据类型不同时，将由系统自动进行类型转换。转换原则是：先将赋值号右边表达式的类型转换为左边变量的类型，然后赋值。

（4）赋值运算符“=”可以理解为“←”，是一个把右边表达式的值存储到左边变量的存储空间的单向操作。

赋值运算符的优先级仅仅高于逗号运算符，具有自右向左的结合性。

【例 2.8】已知“int a=2,b=5,x,y;”，求执行表达式“x=y=a+b;”后 x，y 的值。

由于算术运算符的优先级高于赋值运算符，先计算表达式 a+b 的值，结果为 7。按照赋值运算符的结合方向为自右向左的结合，求解“x=y=a+b;”的值，等价于求解表达式“x=(y=a+b)”的值。操作数 y 是先与右边的运算符结合，即先将 7 赋给变量 y，表达式 y=7 的值是 7。再做左边赋值运算，即将表达式 y=7 的值赋给变量 x。变量 x 的值是 7。最后得到变量 x 和 y 的值都是 7。

【例 2.9】设有定义“int a,b,c;”，求执行表达式 a=(b=65)/(c=6)后 a 的值。

按照括号优先，先求解表达式 b=65，结果为 65；再求解表达式 c=6，结果为 6；再进行除法运算 65/6，结果为 10；最后将 10 赋值给变量 a，得到表达式 a=(b=65)/(c=6)的值是 10。

2. 复合赋值运算符

在赋值符“=”之前加上某些运算符（如+、-、*、/、%等），可以构成复合赋值运算符，复合赋值运算符可以构成赋值表达式。C 语言中许多双目运算符可以与赋值运算符一起构成复合运算符，它的一般形式为：

<变量>算术符合赋值运算符<表达式>

例如：

a+=b　　等效为：a=a+b

a-=b　　等效为：a=a-b

a*=b　　等效为：a=a*b

a/=b　　等效为：a=a/b

a%=b　　等效为：a=a%b

注意：y*=a+b；应等价于 y=y*(a+b)，而不是 y= y *a+b。

【例 2.10】已知“int a=6,b=8;”，求执行表达式 a*=b+=12 后 a、b 的值。

求解表达式 a*=b+=12 的值，等价于求解表达式 a=a(*b+=12)的值；先求解表达式 b+=12 的值，等价于求解表达式 b=b+12 的值，先将变量 b 中存放的值 8 取出来和 12 做加法运算，将运算的结果 20 存入变量 b 中，得到变量 b 的值为 20，得到表达式 b=b+20 的值是 20。再求解 a=a*20，即将变量 a 中存放的值 6 取出来和 20 做乘法运算得到结果为 120，再将 120 存入变量 a 中，得到表达式 a*=b+=12 的值为 120。

【例 2.11】 有定义：“int a=12;”，执行表达式 a+=a-=a*=a 后，a 的值为（　　）。

A．12　　B．144　　C．0　　D．132

解析：表达式 a+=a-=a*=a 的运算方向是自右向左的，也就是说先计算 a*=a，则 a 的值为 144，再计算 a-=a，a 的值为 a-a=0，最后计算 a+=a，故表达式 a 的值都为 a+a=0，所以答案选 C。

2.5.2　算术运算符和算术表达式

算术运算符包括：+（加），-（减），*（乘），/（除），%（取余），++（增 1 运算符或称自增运算符），--（减 1 运算符或称自减运算符）。

双目运算符+、-、*、/的操作数可为任何整数或浮点数。对于“+”和“-”，还可用于指针加或减一个整数。双目运算符的两个操作数类型可以不同，运算前遵循类型的一般算术转换规则自动转换成相同的类型，运算结果的类型与转换后操作数的类型相同。类型的一般算术转换规则的基本原则是值域较窄的类型向较宽的类型转换。例如：

13+5：结果为整数 18。两个操作数类型相同，都为 int，不执行类型转换。

13.0+5：结果为双精度浮点数 18.0。操作数 13.0 类型为 double，执行加运算之前，5 被转换成 double。

'd' –97：结果为整数 3。'd'的类型为 char，97 为 int，执行减运算之前，'d'被转换成 int，'d'的 ASCII 值为整数 100。

对于除运算符“/”，如果两操作数都是整数，则执行整数除，运算结果也是整数，值为商的整数部分，小数部分被截去；否则，执行实数除，运算结果是浮点数。例如：

15/5　结果为整数 3

3/6　结果为整数 0

15.0/3 或 15/3.0 或 15.0/3.0 结果相同，均为双精度浮点数 5.0。

对于求余运算符%，规定两操作数必须都为整数，运算结果也为整数，值为左操作数除以右操作数所得的余数。例如：

17%5　　结果为整数 2

5%10　　结果为整数 5

10%5　　结果为整数 0

对于运算符/和运算符%用于负操作数的时候，C89 标准规定，如果两个操作数中有一个是负数，除法/运算的结果既可以向上取整，也可以向下取整。例如，-9/7 的结果既可以是-1 也可以是-2。对于取余运算符%，-9%7 的值可能是-2 或者 5。但在 C99 标准中，除法的结果总是向零截取的（因此-9/7 的结果为-1），i%j 的值的符号与 i 相同（因此-9%7 的值为-2）。

下面是一个由算术运算符连接操作数构成的表达式：

25+(i%j*8/i-k))

其中 i、j 必须为整数，k 可为整数或浮点数。

增 1 运算符++和减 1 运算符--是两个独特的单目运算符，其独特之处在于它们既可以用作变量的前缀，又可以作为变量的后缀。这两种用法的效果都是使变量增 1 或减 1，但用作前缀时，++和--表示先增加和先减少再取其值，而用作后缀时，++和--表示先取其值再增加和再减少，前缀式先自增（或自减）后运算，后缀式先运算后自增（或自减）。有的教材对前缀式和后缀式也说成：前缀式先自增（或自减）后引用，后缀式先引用后自增（或自减）。

++i; --i;　　在引用 i 之前，先使 i 的值增 1、减 1。

i++; i--;　　在引用 i 之后，使 i 的值增 1、减 1。

粗看起来，++i 和 i++的作用都相当于 i=i+1，但细细研究一下，++i 是先执行 i=i+1，再引用 i 的值，而 i++是先引用 i 的值后，再执行 i=i+1。

若 i 的初始值为 8，则

j=++i;　　先使 i 的值加 1，变为 9，然后再送给 j。因此，j 的值为 9。

若 i 的初始值为 8，则

j=i++;　　先将 i 的值送给 j，这样，j 的值为 8，然后 i 值再加 1，变为 9。

因此，自增或自减运算本身也是一种赋值运算。

【例 2.12】假设变量 i,j,k 的值分别为 3，5 和 3，求执行表达式“m=(++k)*j;”和“n=(i++)*j;”后 m 和 n 的值。

在计算“m=(++k)*j;”时，++k 是前缀式，先自增后运算，++k 的值为 4，然后与 j 相乘，即 m=(++k)*j=4*5=20；最后将 20 赋给变量 m。在计算“n=(i++)*j;”时，由于 i++为后缀式，先引用后自增。因此 i 是用 3 的值参与乘法运算，即 n=(i++)*j=3*5=15，在参与运算后，i 才增 1 变为 4；最后赋值给变量 n 的值是 15。

注意：++和--运算的结合方向是自右向左的。

【例 2.13】“++”和“--”的应用举例。

```
#include<stdio.h>
void main()
{    int i=8;
     printf("%d ",++i);
     printf("%d ",--i);
     printf("%d ",i++);
     printf("%d ",i--);
     return 0;
}
```

程序运行结果如下：

```
9 8 8 9
```

说明：语句“printf("%d ",++i);”中的++i 是前缀式，先自增为 9 然后参与输出运算；对于语句“printf("%d ",--i);”中的--i 是前缀式，i 中的值在执行第一个输出语句的时候已经修改为 9，先自减 1 然后参与输出运算，所以输出的值为 8；对于语句“printf("%d ",i++);”中的 i++属于后缀式，先引用参与输出运算然后变量 i 的值自动增 1；由于在执行第二个输出语句的时候 i 的值已经修改为 8，所以输出的结果为 8，此时 i 的值已经修改为 9；最后一个输出语句“printf("%d ",i--);”同样也为后缀式，先引用参与输出运算，然后自减，输出结果为 9，同时使 i 的值修改为 8。

必须说明的是：++、--运算符的结合方向是自右至左，只能用于整型变量，而不能用于常量或表达式。例如：5++或(x+y)++都是错误的。

2.5.3　长度测试运算符 sizeof

长度测试运算符 sizeof 可用来测试某个类型的变量所占用计算机内存空间的字节长度。

格式为：sizeof(类型名)

例如，下面的程序段：

```
int a;
a=sizeof(float);
printf("%d",a);
```

其输出结果为 4。

读者可以在自己的机器上运行下面的程序，观察输出结果和自己的分析是否相符。

```
main()
{
    printf("char:%d    bytes\n",sizeof(char));
    printf("short:%d    bytes\n",sizeof(short));
    printf("int:%d    bytes\n",sizeof(int));
    printf("long:%d    bytes\n",sizeof(long));
    printf("float:%d    bytes\n",sizeof(float));
    printf("double:%d    bytes\n",sizeof(double));
    return 0;
}
```

2.5.4　关系运算符和关系表达式

1. 关系运算符

关系运算实际上就是比较运算，即比较两个运算对象值的大小，表 2-3 列出了 C 语言提供的 6 个关系运算符。

表 2-3　关系运算符

符号	优先级
<（小于），>（大于），<=（小于或等于），>=（大于或等于）	高
==（等于），!=（不等于）	低

注意：（1）关系运算符为双目运算符，结合方向为自左至右。

（2）关系运算符的结果为真（1）或假（0），C 语言中没有逻辑值。

（3）算术运算符高于关系运算符。

2. 关系表达式

由关系运算符、运算对象以及小括号组成的表达式称为关系表达式。当表达式成立时，返回值为整型值 1；当表达式不成立时，返回值为整型值 0。

例如：c>a+b　等同于 c>(a+b)

a=b>c　等同于 a=(b>c)

a>b!=c　等同于(a>b)!=c

【例 2.14】 请注意下列给出的表达式及其返回值。

① 6<8 的返回值是非零（1）。

② 3>5 的返回值是 0。

③ 若 a=3，b=5，则 a>=3+10 的返回值是 0。

④ 若 x=90，则 x<=100 的返回值是 1。

⑤ 若 x=10，y=15，则 x+y==25 的返回值是 1。

⑥ 若 z=36，则 z!=36 的返回值是 0。

注意：③中表达式相当于 a>=(3+10)，先算 3+10 再与 a 比较。因为算术运算符比关系运算符的优先级高，小括号可以不写，但写了更清楚。

【例 2.15】 下列程序运行后的结果为（　　）。

```
main()
{
    int x;
    x=10>5>3;
    printf("%d\n",x);
    return 0;
}
```

解析：关系运算符优先级高于赋值运算符，同样级别的关系运算符，运算方向从左到右。先判定 10>5 为真，结果为 1，然后判定 1>3 为假，输出 0。

【例 2.16】 已知 a=3，b=2，c=1，则表达式 a>b==c 的值为（　　）。

解析：此关系表达式中“>”的优先级高于“==”，相当于(a>b)==c，由 a>b 为真，其值为 1，实际上判断 1==1，结果为“真”，用 1 表示。

对于关系表达式要注意如下事项：

（1）一个关系式中含有多个关系表达式时，要注意与数学式的区别。例如，数学式 0<x<6 表示 x 的范围为 0～6 之间。而关系表达式 0<x<6，根据左结合性，先计算出 0<x 的结果，然后再计算(0<x)<6 的结果；如 10>5>3，先计算 10>5 的结果为 1，再计算(10>5)>3 的结果，即 1>3 的结果为 0。

（2）应避免对实数做相等或不等的判断，因为实数在内存中存放时有一定的误差。如果一定要进行比较，则可用它们差的绝对值去与一个很小的数（例如 10^{-5}）相比，即 fabs(x-y)<10^{-5}。如果值为真，则表示它们是相等的。

2.5.5 逻辑运算符与逻辑表达式

1. 逻辑运算符

C 语言提供了三种逻辑运算符。逻辑运算符具有自左至右的结合性，表 2-4 列出了三种逻辑运算符的含义及运算级别。

表 2-4 逻辑运算符的含义及运算级别

运算符	意义	级别
&&	与	中
||	或	低
!	非	高

其中，&&和||运算符是双目运算符，！运算符是单目运算符，应该出现在运算对象的左边，如!a。

以上运算符的优先级次序是：!（逻辑非）级别最高，&&（逻辑与）次之，||（逻辑或）最低。逻辑运算符与赋值运算符、算术运算符、关系运算符之间从高到低的运算优先级次序是：!（逻辑非）、算术运算、关系运算、&&（逻辑与）、||（逻辑或）、赋值运算。

2. 逻辑表达式

用逻辑运算符将关系表达式连接起来构成逻辑表达式。逻辑运算的对象可以是 C 语言中

任意合法的表达式。逻辑表达式的运算结果或者为 1（“真”），或者为 0（“假”）。

例如：a>b&&x>y　　等同于　　(a>b)&& (x>y)

　　a==b||x==y　　等同于　　(a==b)||(x==y)

　　!a||a>b　　等同于　　(!a) ||(a>b)

表 2-5 列出了三种逻辑运算的真值，其中 1 表示“真”，0 表示“假”。

表 2-5　真值表

数据 a	数据 b	!a	!b	a&&b	a||b	!(a&&b)
1	1	0	0	1	1	0
1	0	0	1	0	1	1
0	1	1	0	0	1	1
0	0	1	1	0	0	1

在诸如-5&&4 的逻辑表达式中，-5 和 4 并不是逻辑值 1 或 0，该如何处理呢？在进行逻辑运算时，C 语言规定，非 0 得“真”，0 得“假”。-5 和 4 都是非 0 数，都得“真”，故逻辑表达式-5 && 4 运算结果为“真”。因而逻辑表达式-5&&4 的值为 1。

【例 2.17】 逻辑表达式!(5>3)&&(2<4)的值为（　　）。

解析：该表达式相当于(!(5>3))&&(2<4)得 0&&1，结果为 0。

值得注意的是，在数学上关系式 5>x>2 表示 x 的值应在 2～5 的范围内。但在 C 语言中不能用于 5>x>2 这样一个关系表达式来表述以上的逻辑关系。因为无论 x 是什么值，按照 C 语言的运算规则，表达式 5>x>2 的值总是 0。只有采用 C 语言提供的逻辑表达式 x>2&&x<5 才能正确表述以上关系。

【例 2.18】 已知 year 为整型变量，不能使表达式(year%4==0&&year%100!=0)||(year%400==0)的值为“真”的数据是（　　）。

A．1990　　B．1992　　C．1996　　D．2000

解析：表达式为“真”时有两种情况：第一，year 能被 4 整除，但不能被 100 整除；第二，year 能被 400 整除。故在所选项中不符合上面两种情况的只有 A，所以答案为 A。

C 语言逻辑表达式的特性：在计算逻辑表达式时，只有在必需执行下一个表达式才能求解时，才求该表达式，即并不是所有的表达式都被求解。

（1）逻辑与(&&)运算表达式中，只要前面一个表达式被判定为“假”，系统不再判定其后的表达式，整个表达式的值为 0。

例如：“a++&&b++;”，若 a 的初始值为 0，表达式首先求 a++的值，由于表达式 a++的值为 0，系统完全可以确定逻辑表达的运算结果总是为 0。因此将跳过 b++不再对它进行求值。在这种情况下，a 的值将自增，由 0 变成 1，而 b 的值将不变。若 a 的值不为 0，则系统不能仅根据表达式 a++的值来确定逻辑表达的运算结果。因此必然再对运算符&&右边的表达式 b++进行求值，这时将进行 b++的运算，使 b 的值改变。

（2）逻辑或(||)运算表达式中，只要前面一个表达式被判定为“真”，系统不再判定或求解其后的表达式，整个表达式的值为 1。

例如：“a++||b++;”，若 a 的初始值为 1，表达式首先求 a++的值，由于表达式 a++的值为 1，

无论表达式 b++为何值，系统完全可以确定逻辑表达式的运算结果总是为 1。因此也将跳过 b++，不再对它进行求值。在这种情况下 a 的值将自增 1，b 的值将不变。若 a 的值为 0，则系统不能仅根据表达式 a++=0 的值来确定逻辑表达式的运算结果。因此必然再对运算符“||”右边的表达式 b++进行求值，将进行 b++的运算，使 b 的值改变。

【例 2.19】 有定义：“int i=0,j=10,s=0;”，执行下列语句，s 和 i、j 的值是多少？

```
s=++i||++j;
```

解析：因为逻辑运算符优先级比算术运算符的级别低，逻辑式||右边的表达式在最后运算，如||运算符的左边为真，其运算结果与右边无关，因而此时计算机不对右边的表达式进行运算。答案为：s=1，i=1，j=10。

【例 2.20】 有定义“int x=0,y=0,z=0;”，执行下列语句后，各变量的值是多少？

```
++x&&++y||++z
```

解析：因为||的优先级较低，逻辑式||右边的表达式在最后运算，它相当于：++x,++y,x&&y，如 x&&y 的值为假，再执行++z，x&&y 的结果与 z 进行逻辑||运算；如果 x&&y 的值为真，将不再与 z 做逻辑||运算，因而本题的答案为：x=1，y=1，z=0。

注意：关系运算符、算术运算符和赋值运算符之间，优先级的次序是：算术运算符优先级别最高，关系运算符次之，赋值运算符最低。

下面介绍一个有趣的短路测试程序。

【例 2.21】测试短路现象。

```
#include "stdio.h"
int show(int n,int a)
  {
      printf("(%d,%d),n,a);
      return a;
  }
void main()
  {
    int 3;
    a=show(1,1)||show(2,2)&&show(3,3); printf("a=%d\n",a);
    a=show(1,0)||show(2,0)&&show(3,1); printf("a=%d\n",a);
    a=show(1,0)||show(2,1)&&show(3,0); printf("a=%d\n",a);
    return 0;
  }
```

程序中 int show(int n,int a)是一个自定义的函数，函数的功能是输出 n 和 a 的值，并返回 a 的值，有关函数定义和调用的知识将在第 6 章“函数”中予以介绍，该程序的运行结果如下：

(1,1)a=1

(1,0)(2,0)a=0

(1,0)(2,1)(3,2)a=1

主函数中“a=show(1,1)||show(2,2)&&show(3,3);”含有逻辑表达式，通过 1 和 1 调用，除了输出(1,1)外，还返回 1，由于是||运算，有 1 则 1，后面的 show(2,2)和 show(3,3)并没有被执行，出现短路现象，逻辑运算符||后面的表达式将被忽略。

主函数“a=show(1,0)||show(2,0)&&show(3,1);”则不同，show(1,0)返回 0，不能忽略逻辑运算符||右边的表达式 show(2,0)，由于 show(2,0)的返回值也是 0，所以 show(1,0)||show(2,0)的

值等于 0，对于运算符&&，也出现了短路现象，所以 show(3,1)没有被执行。

2.5.6　条件运算符与条件运算表达式

条件运算符“?:”是 C 语言提供的唯一一个三目运算符，其应用灵活，功能很强。条件运算表达式的一般格式为：

表达式 1?表达式 2:表达式 3

条件运算表达式的功能如图 2-2 所示，执行顺序为：先求解表达式 1，若为非 0，再求解表达式 2，该表达式的值就为整个表达式的值；若表达式 1 的值为 0，则求解表达式 3，该表达式的值就为整个表达式的值。

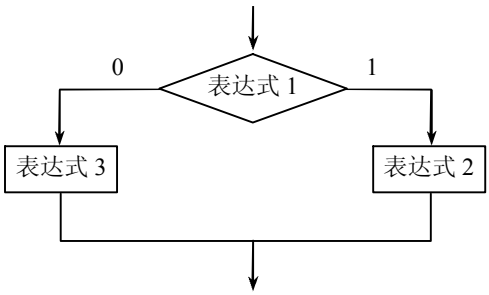


图 2-2　条件运算流程

【例 2.22】 从键盘读入一个整数赋给 x，如果 x 大于等于 0，把 x 的平方赋给 y，否则把 x 的 2 倍赋给 y。

解析：按题意程序流程图如图 2-3 所示，写成 C 语言的语句为：y=x>=0? x*x:2*x;

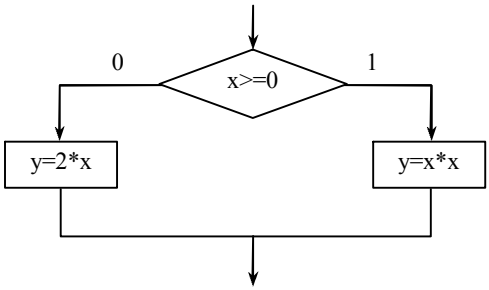


图 2-3　程序流程图

程序代码为：

```
#include"stdio.h"
void main()
{
    int x,y;
    scanf("%d",&x);
    y=x>=0?x*x:2*x;
    printf("y=%d\n",y);
    return 0;
}
```

【例 2.23】 已定义“int x=4,b=5,y;”，执行语句：y=++x>b?x:b-->++x?++b:x 后，x、b、y 的值分别为_____。

解析：条件运算符的优先级别仅高于赋值运算符，比关系运算符和算术运算符都低。是否需要求解表达式 b-->++x?++b:x，取决于++x>b?的真假，如为真使得该表达式的值 y 为 x，如为假，即再求解表达式 b-->++x?++b:x，本题++x>b 为假，由于在判断++x>b 时 x 值为 5，判断 b-->++x 时 x 值为 6，所以整个表达式的值为 6。答案为 x=6，b=4，y=6。

【例 2.24】 分析下列程序的输出结果。

```
#include "stdio.h"
void main()
{
    int i,j,k,a=3,b=2;
    i=(--a==b++)?--a:++b;
    j=a++;
    k=++b;
    printf("i=%d,j=%d,k=%d\n",i,j,k);
    return 0;
}
```

解析：因为--a==b++相当于--a,a==b,b++;，因而表达式--a==b++为真，所以 i 的值为--a，即 a=1，i=1，b=3，而且执行表达式 j=a++后 j=1，a=2，k=4。最后 i=1，j=1，k=4。

【例 2.25】 请编制程序，任意输入两个整数，输出其中较小者。

```
main()
{
    int n1,n2,min;
    clrscr();
    printf("input two number(n1,n2): ");
    scanf("%d,%d",&n1,&n2);
    min=(n1>n2)?n2:n1;
    printf("min=%d",min);
    return 0;
}
```

运行时，按屏幕提示操作：

```
input two number(n1,n2):3,5↙
```

程序运行结果如下：

```
min=3
```

【例 2.26】 编制程序，输入大写字母，则输出小写字母；输入小写字母，则输出大写字母。

```
main()
{
    char ch;
    printf("input any letter: ");
    scanf("%c",&ch);
    ch=((ch>='A')&&(ch<='z'))?ch+32:ch-32;
    printf("output:%c\n",ch);
    return 0;
}
```

运行时，按屏幕提示操作：

```
input any letter:G↙
```

程序运行结果如下：

```
output :g
```

再运行：

```
input any letter:e↙
```

则程序运行结果如下：

```
output :E
```

注意：（1）条件运算符优先级高于赋值运算符，而关系、算术运算符高于条件运算符，因而：

max=a>b?a:b 等效于 max=((a>b)?a:b)。

x=a>b?a:b+1 等效于 x=((a>b)?a:(b+1))。

（2）条件运算符结合方向为自右至左。有“a>b?a:c>d?c:d;”，它等效于“a>b?a:(c>d?c:d) ;”，式中并不表示先运算括号内的表达式，它只不过表示结合性。

（3）条件表达式中类型可以不一样，如 x? 'a': 'b'，若 x=0 则条件表达式值为'b'。

2.5.7　逗号运算符与逗号表达式

逗号运算符，又称“顺序求值运算符”。逗号表达式的一般形式为：

表达式 1,表达式 2　或　表达式 1,表达式 2,表达式 3,…,表达式 n;

运算过程是：自左至右依次计算表达式 1，扔掉表达式 1 的值，再计算表达式 2，再扔掉表达式 2 的值，…，最后计算表达式 n 的值，而整个逗号表达式的值取最后的表达式 n 的值。

【例 2.27】 分析下列逗号表达式的计算过程及结果。

① 3+5,6*3　　表达式的值是 18。

② a=(3+5,6*3)　　a 的值是 18，此为赋值表达式。

③ a=3+5,6*3　　a 的值是 8，表达式的值是 18（先做赋值运算）。

④ a=3+5,6*a　　a 的值是 8，表达式的值是 48。

⑤ (a=3+5,6*a),a+100　　a 的值是 8，表达式的值是 108。

使用逗号表达式应该注意如下事项：

（1）逗号表达式可以和另外一个表达式组成一个新的逗号表达式。例如，逗号表达式“(a=6,3*a),a+10”中，表达式 1 是“(a=6,3*a)”，表达式 2 是 a+10，先将 6 赋给 a，在计算 3*a 得到 18，a 的值不变，最后计算 a+10，得 16，整个表达式的值为 16。

（2）并不是在所有出现逗号的地方都组成逗号表达式。如在变量说明中和函数参数表中的逗号只是用作各变量之间的间隔符。

（3）逗号表达式只是把各个表达式“串联起来”，例如 5.3 节将要介绍的 for(i=10,j=1; i>=j;i--,j++)语句，它的真正目的并不是使用逗号表达式的值，而是希望分别得到变量 i 和 j 的值。

【例 2.28】 下面是给字符变量赋值的 6 种方式，仔细体会字符数据的表示方法和逗号表达式的用法。

```
#include "stdio.h"
main()
{
```

```
    char c1, c2, c3, c4, c5, c6;
    c1='A', c2='\x41', c3='\101', c4=65, c5=0x41, c6=0101;
    printf("%c, %c, %c, %c, %c, %c \n", c1, c2, c3, c4, c5, c6);
    printf("%d, %d, %d, %d, %d, %d \n", c1, c2, c3, c4, c5, c6);
    return0;
}
```

程序运行结果如下：

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

说明：程序声明了 6 个字符型变量，给字符型变量 c1，c2，c3，c4，c5 和 c6 赋值后，它们的值都是字符 A 的 ASCII 码 65，按字符输出也得到字符 A，按十进制输出时得到整数 65。用逗号分隔的多个赋值表达式“c1= 'A', c2='\x41', c3='\101', c4=65, c5=0x41, c6=0101;”由于只有一个分号，因而格式上是一个语句。本例还说明一个问题，一个字符型数据和整型数据是通用的。

2.5.8 位运算

位运算的作用是对运算对象按照二进制位进行操作的运算，它能够对字节或字中的实际位进行检测、设置或位移，它运算的对象只能是字符型或整型变量以及它们的变体，对其他类型的数据不适用。

1. 位运算符

表 2-6 列出了 C 语言的位操作运算符。

表 2-6 位运算符及其功能

位运算符	作用	举例
~	按位取反	~a，对变量 a 中全部位取反
<<	左移	a <<2，将 a 中各位全部左移 2 位
>>	右移	a>>2，将 a 中各位全部右移 2 位
&	按位与	a&b，将 a 和 b 中的各位按位进行“与”运算
|	按位或	a|b，将 a 和 b 中的各位按位进行“或”运算
∧	按位异或	a∧b，将 a 和 b 中的各位按位进行“异或”运算

位运算符还可以与赋值运算符相结合，成为位运算赋值操作。位运算赋值操作如表 2-7 所示。

表 2-7 位运算操作赋值规则

运算符	作用	举例	等价表达式
<<=	左移赋值	a<<=n	A=a<<n
>>=	右移赋值	b>>=n	B=b>>n
&=	位与赋值	a&=b	a=a&b
∧=	位异或赋值	a∧=b	a=a∧b
|=	位或赋值	a|=b	a=a|b

2. 位运算的功能

双目位逻辑运算符的运算规则如表 2-8 所示。

表 2-8　双目位逻辑运算符的运算规则

对象 a	对象 b	位与 a&b	位或 a|b	异或 a∧b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

（1）按位取反（~）。~是单目运算符，用来对一个二进制数按位取反。也就是说，将原来的 0 变为 1，1 变为 0。例如~025 是对八进制数 25（即二进制数据 00010101）按位求反。

00010101

~　⇓

11101010

【例 2.29】 分析下列程序的输出结果。

```
void main()
{
    unsigned    char a,b;
    a=0x9a;
    b=~a;
    printf ("a:%x     b:%x\n",a,b);
    return 0;
}
```

程序运行结果如下：

```
a:9a        b:65
```

（2）按位与（&）。参加运算的两个数据，按照二进制位进行“&”运算，如果两个相应的二进制位都为 1，则该位的结果为 1，否则为 0。即 0&0=0，0&1=0，1&0=0，1&1=1。

例如：3&5 并不等于 8，应该是先将其分别转化为二进制再进行与运算。

```
    00000011    （3）
&   00000101    （5）
    --------
    00000001
```

按位与有以下功能：

1）清 0。如果想将一个单元清 0，即让全部二进制位为 0，只要找一个二进制数，其中的各位符合以下条件：原来的数中为 1 的位，新数中的相应位为 0。然后将二者进行&运算，可以达到清 0 的目的。也就是原数取反，再和原数进行按位与，就可以达到清 0 的目的。

【例 2.30】 分析下列程序的运行结果。

```
void main()
{
    int x;
    x=x&~x;
```

```
    printf("%d\n",x);
    return 0;
}
```

解析：假如 x 的值为 00101011，~x 后为 11010100，则

```
    00101011
&   11010100
    00000000
```

2）取一个数中的某些指定位。如果有一个整数 a（2 个字节），想要其中的低字节，只需要将 a 与 377 按位与即可。

如果想要保留高字节，思考应该和谁按位与？

（3）按位或（|）。两个相“或”的二进制位中只要有一个为 1，该位或的结果就为 1，即：0|0=0，0|1=1，1|0=1，1|1=1。

“按位或”常用来将某一个数据的某些特征位置 1。如：a 是一个整数（16 位），表达式 a|0377 使得低 8 位全置 1，高 8 位保留原样。

【例 2.31】 分析下列程序的输出结果。

```
#include"stdio.h"
void main()
{
    char  x;
    x='B';
    x=x|0x80;
    printf("%d\n",x);
    return 0;
}
```

解析：字符'B'的二进制码为 01000010，经过运算 x=x|0x80 后，x 的值为 11000010，由于它是补码存储，以十进制数输出结果为-62。

（4）按位异或（∧）。异或运算符∧也称 XOR 运算符。它的作用是判断两个相应位的值是否“相异”（不同），若相异则结果为 1（真），否则为 0。即：0∧0=0，0∧1=1，1∧0=1，1∧1=0。

按位异或有以下应用：

1）使特定位翻转。假设有 01000010，想使低 4 位翻转，只要与 00001111 异或，异或后的结果为 01001101。

2）与 0 相异或保留原值。例如，01111010∧00000000 的结果为 01111010。又如，对负数的补码 A 求反码的方法是此数 A 与 7FH 异或，即 7∧7FH。

3）不用中间变量就可以交换两个数的值。

【例 2.32】 分析下列程序的运行结果。

```
#include"stdio.h"
void main()
{
    int a=3,b=4;
    a=a∧b;
    b=b∧a;
```

```
    a=a ∧ b;
    printf ("a=%d      b=%d\n",a,b);
    return 0;
}
```

程序运行结果如下：

```
a=4      b=3
```

（5）位移（左移<<，右移>>）。

1）左移（<<）。移位后，右端位出现的空位补 0，移出左端之外舍弃，例如：

a=a<<2

若 a=15 即二进制数 00001111，左移 2 位得 00111100。左移 1 位相当于原来的数据乘以 2，左移 2 位相当于将原来的数据乘以 4，左移比做乘法快得多。

2）右移（>>）。a=a>>2 表示将 a 的各二进制位右移 2 位，移到右端的低位被舍弃，对于无符号数，右移后高位补 0。

如 a=017 时，a 的二进制码为 00001111，a=a>>2 为 00000011¦11

此二位将被舍弃

右移一位相当于将原来的数据除以 2，右移 N 位相当于将原来的数据除以 2^N。

在右移时，需要注意符号位问题。对于无符号数，右移时左边高位移入 0；对于有符号的数据，如果原来的符号位为 0，和上面介绍的一样，如果符号位原来为 1，则左边移入 0 还是 1，取决于所用的计算机系统。

2.6　不同类型数据间的转换

C 语言允许数据的值从一种类型转换为另一种类型。下列情况之一会引起类型转换：

（1）当双目运算符的两个操作数类型不相同时，引起一般算术转换（或称运算符转换）。

（2）当一个值赋予一个不同类型的变量时，引起赋值转换。

（3）当一个值被强制为另一个类型时，引起强制类型转换。

（4）当某个值作为参数传给一个函数时，引起函数调用转换。

其中，赋值转换、一般算术转换和函数调用转换是由系统自动隐含进行的，强制类型转换是由程序员使用强制运算符指定进行的显式类型转换。

1．一般算术转换

一般算术转换（简称算术转换）的基本规则为：双目运算符的两个操作数中，值域较窄的那个类型向值域较宽的那个类型转换。值域是类型所能表示的值的最大范围。被转换的两个操作数可为任意类型。转换规则如下：

（1）将表达式中的 char 或 short 全部自动转换为相应的 int 型，将 float 转换为 double 型。即：

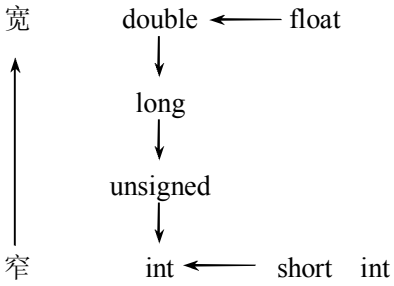
char
short　　⟹ int

unsigned char
unsigned short　　⟹ unsigned int

float ⟶ double
long
unsigned long
结果表达式中只剩下五种类型。

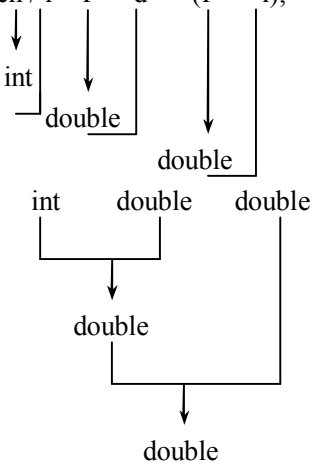
（2）当运算符的两端运算类型不一致的时候，存储值域窄的向值域较宽的转换。

类型转换的规则如下所示：



一般算术转换是在运算过程中自动进行的。如：

```
char ch;
int i,result;
float f;
double d;
result=ch / i + f*  d -  (f +  i);
```



2. 强制类型转换规则

强制类型表达式的形式为：

(类型名)操作数

“类型名”是强制类型运算符（或类型强制符），它将操作数转换为由“类型名”指定的类型，表达式的结果和类型与转换后的操作数的值和类型相同。

强制类型转换在效果上和赋值转换完全相同，它们的转换方向都不受算术转换规则的约束，强制转换与赋值转换的区别是：

（1）强制转换是显式方式，赋值转换是隐式方式，前者是人为的，后者是自动的。

（2）强制转换的结果类型由强制运算符指定的类型名决定，赋值转换的结果类型由赋值

运算符的左操作数的类型决定。

强制转换应用于除赋值转换以外的任何与算术转换方向不同的类型转换。例如：

```
(char)(3-3.14159*x)              /*得到字符型数据*/
k=(int)((int)x+(float)i+j)       /*得到整型数据*/
(float)(a=99)                    /*得到实型单精度数据*/
```

强制转换是一种单目运算。各种数据类型的标识符都可以用作强制转换运算符，但必须用圆括号把类型标识符括起来，不要写成 int(3+5)的形式。如果写成：(int)3.6+7.2，则只对 3.6 转换，相当于((int)3.6)+7.2，如果想对 3.6+7.2 这个表达式进行强制转换，应加括号，即(int)(3.6+7.2)。

注意：类型强制符属于最高优先级。

【例 2.33】

```
#include"stdio.h"
void   main()
{
    float x;
    int i;
    x=3.6;
    i=(int)x;
    printf("x=%f,i=%d\n",x,i);
    return 0;
}
```

程序运行结果如下：

```
x=3.600000,i=3
```

注意：语句 i=(int)x 把 x 强制转换后赋给 i，而 x 本身的类型并不改变。

【例 2.34】 设 x=2.5，a=7，y=4.7，算术表达式 x+a%3*(int)(x+y)%2/4 的结果为（　　）。

A．2.5　　B．7　　C．4.7　　D．2.75

解析：取模运算只能在整数之间进行，a 为整型变量，因为%、*、/这三个运算符的优先级别相同，而类型强制转换符优于*、/运算符，a 的值为 7，7%3=1，又因为(int)(2.5+4.7)=7，1*7=7，7%2=1，1/4=0，所以表达式 x+a%3*(int)(x+y)%2/4=x+0=2.5。答案为 A。

习　　题

一、选择题

1．在 C 语言中，正确的 int 类型的常数是（　　）。

A．-2U　　B．059　　C．3a　　D．0xaf

2．下列数据中，为字符的是（　　）。

A．'AB'　　B．"A"　　C．HOW　　D．'A'+2

3．若有以下定义和语句：

```
int u=010,v=0x10,w=10;
printf("%d,%d,%d\n",u,v,w);
```

则输出结果是（　　）。

A．8,16,10　　B．10,10,10　　C．8,8,10　　D．8,10,10

4．以下符合 C 语言语法的浮点型常量是（　　）。

A．1.2E0.5　　B．3.14.159E　　C．.5E-3　　D．E15

5．下列用户定义的标识符不符合规定的是（　　）。

A．f2_g3　　B．For　　C．5n　　D．_n6

6．下列标识符中是 C 语言的保留字是（　　）。

A．Int　　B．FLOAT　　C．double　　D．Char

7．下列变量定义正确的是（　　）。

A．int x_1;y;　　B．int x=y=5;　　C．int for=4;　　D．printf=2,x_y=2;

8．下列变量定义中合法的是（　　）。

A．short _a=1-ae-1　　B．double b=1+5e2.5

C．long ao=0xfdal　　D．float 2_and=1-e-3

9．设 x、y、z 和 k 都是 int 型变量，则执行表达式 x=(y=52,z=26,k=32)后，x 的值为（　　）。

A．4　　B．26　　C．32　　D．52

10．下列算术运算符中，只能用于整型数据的是（　　）。

A．-　　B．+　　C．/　　D．%

11．若 a 为 int 类型，且其值为 3，则执行完表达式 a += a -= a*a 后，a 的值是（　　）。

A．-3　　B．9　　C．–12　　D．6

12．设有如下变量定义：

```
int i=8,k,a,b;
unsigned long w=5;
double x=1,y=5.2;
```

则符合 C 语言语法的表达式是（　　）。

A．a+=a-=(b=4)*(a=3)　　B．x%(-3)

C．a=a*3=2　　D．y=int(i)

13．假设有以下变量定义：int k=7,x=12;，则能使值为 3 的表达式是（　　）。

A．x%=(k%=5)　　B．x%=(x-k%5)

C．x%=k+k%5　　D．(k%=5)–(x%=k)

14．以下程序的输出结果是（　　）。

```
main()
{
    int a=10,b=10;
    printf("%d    %d\n",--a,++b);
    return 0;
}
```

A．10　11　　B．11　13　　C．9　11　　D．11 12

15．若已定义 x 和 y 为 double 类型，则表达式 x=1,y=x+3/2 的值为（　　）。

A．1.0　　B．1.5　　C．2.0　　D．2.5

16．若有如下定义和语句：

```
char c1='a',c2='f';
printf("%d,%c\n",c2-c1,c2-'a'+'B');
```

则输出结果是（　　）。

A．2,M

B．5,G

C．2,E

D．输出项与对应的格式控制符不一致，输出结果不确定

17．设 x、y、t 均为 int 型变量，则执行语句：x=y=3; t = ++x || ++y;后，y 的值为（　　）。

A．不定值　　B．4　　C．3　　D．1

18．设 a、b、c、d、m、n 均为 int 型变量，且 a=5、b=6、c=7、d=8、m=2、n=2，则逻辑表达式(m = a > b) && (n = c > d) 运算后，n 的值是（　　）。

A．0　　B．1　　C．2　　D．3

19．语句 printf("%d",(a=3)&&(b=-3));的输出结果为（　　）。

A．无输出　　B．不确定　　C．1　　D．-1

20．当 c 值不为 0 时，在下列选项中能正确将 c 的值赋给变量 a、b 的是（　　）。

A．c=b=a　　B．(a=c)||(b=c)　　C．(a=c)&&(b=c)　　D．a=c=b

21．设 int x=2,y=1;，表达式(!x||y--)的值是（　　）。

A．-2　　B．1　　C．2　　D．-1

22．下面程序的输出结果是（　　）。

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

A．0　0　3　　B．0　1　2

C．1　0　3　　D．1　1　2

23．设二进制数 x 的值是 00110101，若想通过 x&y 运算使 x 中的低 4 位不变，高 4 位清零，则以下能实现此功能的是（　　）。

A．x=x|0x0f　　B．x=x&0x0f　　C．x=x|0xf0　　D．x=x&0xf0

24．设有以下语句：

```
char  a=3,b=6,c;
c=a ∧ b<<2;
```

则 C 的二进制值是（　　）。

A．00011011　　B．00010100　　C．00011100　　D．00011000

25．若有定义“int a=2,b=3,c=4;”则值为 0 的表达式是（　　）。

A．a||(b+b)&&(c-a)　　B．(a<b)&&!c||1

C．a&&b　　D．(!a==1)&&(!b==0)

26．设有定义“int k=1,m=2;float f=7;”，以下选项中错误的表达式是（　　）。

A．k=k>=k;　　B．k+f　　C．k%int(f)　　D．k>=f>=m

27．设有定义“int x=10,y=3,z;”，则语句“printf("%d",z=(x%y,x/y));”的输出结果为（　　）。

A．3　　B．0　　C．4　　D．1

28．设有说明语句“char c='\101';”，则变量 c（　　）。

A．包含 1 个字符　　B．包含 2 个字符

C．包含 3 个字符　　D．说明不合法

二、填空题

1．一个 C 程序由若干函数构成，其中必须有一个_____。

2．一个语句中至少应该包含一个_____。

3．若采用十进制数的表示形式，则 077 可表示为_____，0111 可表示为_____，0x29 可表示为_____，0xab 可表示为_____。

4．表达式 5%(-3)的值是_____，表达式-5%(-3)的值是_____。

5．执行下列语句后，z 的值是_____。

```
int x=4,y=25,z=2;
z=(--y/++x)*z--;
```

6．设 x、y、z 均为 int 型变量，且 x=3，y=-4，z=5，请写出下面每个表达式对应的结果。

（1）(x&&y)==(x||z)　　_____

（2）!(x>y)+(y!=z)||(x+y)&&(y-z)　　_____

（3）x++-y+(++z)　　_____

7．设 a=3，b=2，c=1，则 a>b 的值为_____，a>b>c 的值为_____。

8．设有变量定义“int i=5,j=4;”则条件表达式(--i==j++)?--i:++j 的值为_____。

9．执行下列语句后，a 的值为_____，b 的值为_____，c 的值为_____。

```
int x=10,y=9;
int a,b,c;
a=(--x==y++)?--x:++y;
b=x++;
c=y;
```

10．已知：int i,j,k;

i=j=k=3;

求下列表达式的结果。

（1）~i|i&j

（2）i^=j^=i

（3）j*k>>2&i

11．有以下程序段，执行后的结果为_____。

```
int a=2,b=2;
a+=a*=a;
b+=b*=b+b;
printf("%d,%d",a,b);
```

12．有以下程序段，执行后的结果为_____。

```
int a=2,b=3,c=4;
printf("%d,%d",a>0,a<0);
printf("%d,%d",c>b>a,c<b<a);
```

13．有以下程序段，执行后的结果为_____。

```
int i=1,j=2,k=3,m;
m=i++==1&&(++j==3||k++==3);
printf("%d,%d,%d,%d",i,j,k,m);
```

14．有以下程序段，执行后 k 的值为_____。

```
int k=0,a=1,b=2,c=3;
k=a<b?b:a;
k=k>c?c:k;
```

15．定义“int x=10,y=5,z;”，则语句“printf("%d", z=(x+=y,x/y));”的输出结果为_____。

16．有以下程序段，执行后的结果为_____。

```
int a=10,b=9;
printf("%d,%d,%d,%d\n",++a,a,--b,b);
```

三、分析下列程序的输出结果

1．写出下列程序执行后变量 a，b，c 的值。

```
char a=2,b='a';
int c;
c=a+b;
a+=c;
```

2．写出下列程序的输出结果。

```
main()
{   int x;
    x=-4+5*6-7;printf("%d\n",x);
    x=3+4%5-6;printf("%d\n",x);
    x=-3*4%-6/5;printf("%d\n",x);
    x=(10+8)%5/2;printf("%d\n",x);
    return 0;
}
```

3．写出下列程序的输出结果。

```
main()
{   int x=50,y=5,z=5;
    x=y==z;printf("%d\n",x);
    x=x==(y==z);printf("%d\n",x);
    return 0;
}
```

4．写出下列程序的输出结果。

```
main()
{   int x,y,z;
    x=y=2;
    z=3;
    y=x++-1;printf("%d\t%d\t",x,y);
    y=++x-1;printf("%d\t%d\t",x,y);
    y=--z+1;printf("%d\t%d\t",z,y);
    z=4;printf("%d\t%d\t%d ",z++,++z,--z);
    return 0;
}
```

5．写出下列程序的输出结果。

```
main()
{   int x,y,z;
    x=y=z=0;
    ++x||++y&&++z;
    printf("x=%d\ty=%d\tz=%d\n",x,y,z);
    x=y=z=0;
    ++x&&++y||++z;
    printf("x=%d\ty=%d\tz=%d\n",x,y,z);
    x=y=z=0;
    ++x&&++y&&++z;
    printf("x=%d\ty=%d\tz=%d\n",x,y,z);
    x=y=z=-1;
    ++x&&++y&&++z;
    printf("x=%d\ty=%d\tz=%d\n",x,y,z);
    x=y=z=-1;
    ++x&&++y||++z;
    printf("x=%d\ty=%d\tz=%d\n",x,y,z);
    x=y=z=-1;
    ++x||++y&&++z;
    printf("x=%d\ty=%d\tz=%d\n",x,y,z);
    return 0;
}
```

6．写出下列程序的输出结果。

```
main()
{   char c;
    short i;
    c='A';
    i=65;
    printf("c:dec=%d oct=%o hex=%x ASCII=%c\n",c,c,c,c);
    printf("i:dec=%d oct=%o hex=%x unsigned=%u\n",i,i,i,i);
    c='x';
    i=-4;
    printf("c:dec=%d oct=%o hex=%x ASCII=%c\n",c,c,c,c);
    printf("i:dec=%d oct=%o hex=%x unsigned=%u\n",i,i,i,i);
    return 0;
}
```

四、把下列命题写成 C 语言的表达式

1．x∈[-5, 5]。

2．y∈(0, 25.5)。

3．a 除 b 的余数是 3 或 1。

4．三角形的两边之和大于第三边。

5． a 和 b 中有一个小于 c。

6． a 是奇数。

7． a 不能被 b 整除。

五、解答题

已知 A，B，C，D 四人中有一个人是小偷，并且，这四个人中每个人要么说真话，要么说假话。在审讯过程中，这四个人分别回答如下：

A 说：B 没有偷，是 D 偷的。

B 说：我没有偷，是 C 偷的。

C 说：A 没有偷，是 B 偷的。

D 说：我没有偷。

现要求根据这四个人的回答，写出能确定谁是小偷的条件。