第 3 章 选择结构程序设计

在顺序结构中,各语句是按排列的先后次序顺序执行的,是无条件的,不必事先进行任何判断。但在实际生活中,常常有这样的情况:要根据某个条件是否成立决定是否执行指定的任务。例如:

- 如果你在家,我去拜访你;　　(需要判断你是否在家)
- 如果考试不及格,要补考;　　(需要判断是否及格)
- 周末我们去郊游;　　(需要判断是否是周末)
- 如果 a>b,输出 a。　　(需要判断 a 是否大于 b)

判断的结果应该是一个逻辑值:“是”或“否”,在计算机语言中用“真”和“假”表示。例如,当 a>b 时,满足“a>b”条件,就称条件“a>b”为真,如果 a≤b,不满足“a>b”条件,就称条件“a>b”为假。

由于程序需要处理的问题往往比较复杂,因此,在大多数程序中都会包含条件判断。选择结构就是根据指定的条件是否满足,决定执行不同的操作(从给定的两组操作中选择其一)。

3.1 简单的选择结构程序

先通过以下几个程序,初步了解怎样在 C 语言程序中用选择结构处理问题。

【例 3.1】 输入两个实数,按代数值由小到大的顺序输出这两个数。

解题思路:有两个变量 a 和 b,若 a≤b,则两个变量的值不必改变,若 a>b,则把 a 和 b 的值互换,然后顺序输出 a 和 b,即可实现题目要求。因此此题的算法是:做一次比较,然后决定是否进行值的交换。关于两个变量互换值的方法,已在例 2.8 中介绍了。

类似这样简单的问题可以不必先写出算法或画流程图,而直接编写程序。或者说,算法在编程者的脑子里,相当于在算术运算中对简单的问题可以“心算”而不必在纸上写出来一样。

编写程序:

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

```
{
  float a,b,temp;
  printf("please enter a and b: ");
  scanf("%f,%f",&a,&b);
  if(a>b)
    {temp=a;a=b;b=temp;}
  printf("%7.2f,%7.2f\n",a,b);
  return 0;
}
```

运行结果：

```
please enter a and b: 3.6,-3.2↙
  -3.20,  3.60
```

【例 3.2】 输入 a,b,c 三个数,要求按由小到大的顺序输出。

解此题的算法比上一题稍复杂一些。现在先用伪代码写出算法：

```
begin
if a>b 将 a 和 b 对换     (a 是 a,b 中的小者)
if a>c 将 a 和 c 对换     (a 是 a,c 中的小者,因此 a 是三者中最小者)
if b>c 将 b 和 c 对换     (b 是 b,c 中的小者,也是三者中次小者)
输出 a,b,c 的值
end
```

编写程序：按以上算法编写程序。

```
#include<stdio.h>
int main()
{
  float a,b,c,temp;
  printf("please enter a,b,c: ");
  scanf("%f,%f,%f",&a,&b,&c);
  if(a>b)
    {temp=a;a=b;b=temp;}            //实现 a 和 b 的互换
  if(a>c)
    {temp=a;a=c;c=temp;}            //实现 a 和 c 的互换
  if(b>c)
    {temp=b;b=c;c=temp;}            //实现 b 和 c 的互换
  printf("%7.2f,%7.2f,%7.2f\n",a,b,c);
  return 0;
}
```

运行结果：

```
please enter a,b,c: 33.52,-27.65,100.45↙
-27.65, 33.52, 100.45
```

3.2　选择结构中的关系运算

3.1 节的程序中,在 if 语句括号中给出一个需要判别的条件,例如 a>b,a>c,b>c。这些“条件”在程序中是用一个表达式来表示的。类似这种表示判别条件的表达式还有:

```
a+b>c
b*b-4*a*c>0
'a'<'v'
```

这种式子显然不是数值表达式,它包括了“<”和“>”这样的比较符号,这些式子的值并不是一个普通的数值,而是一个逻辑值(“真”或“假”)。例如,问对方:“你是中国人吗?”回答只有两个:“是”或“不是”,而不能回答:“3”或“4”。

用来进行比较的符号称为**关系运算符**(或比较运算符,它用来比较运算符两侧的数据),上面这些表达式称为**关系表达式**。

3.2.1　关系运算符及其优先次序

C 语言提供 6 种关系运算符:

① <　　(小于)　　}
② <=　　(小于或等于)　　}优先级相同(高)
③ >　　(大于)　　}
④ >=　　(大于或等于)　　}
⑤ ==　　(等于)　　}优先级相同(低)
⑥ !=　　(不等于)　　}

关于优先次序的说明:

(1) 前 4 种关系运算符(<,<=,>,>=)的优先级别相同,后两种也相同。前 4 种高于后两种。例如,“>”优先于“==”,而“>”与“<”优先级相同。

(2) 关系运算符的优先级低于算术运算符。

(3) 关系运算符的优先级高于赋值运算符。

以上关系见图 3.1。

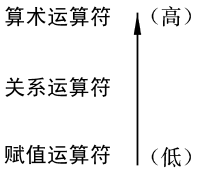


图　3.1

例如:

c>a+b	等效于	c>(a+b)
a>b==c	等效于	(a>b)==c
a==b<c	等效于	a==(b<c)
a=b>c	等效于	a=(b>c)

3.2.2　关系表达式

用关系运算符将两个表达式(可以是算术表达式或关系表达式、逻辑表达式、赋值表达式、字符表达式)连接起来的式子,称**关系表达式**。例如,下面都是合法的关系表达式:

```
a+b>b+c
(a=3)>(b=5)
'a'<'z'
(a>b)>(b<c)
```

前面已说明，条件判断的结果是一个逻辑值（“真”或“假”）。同理，关系表达式的值也是一个逻辑值。例如，关系表达式“5==3”的值为“假”，“5>0”的值为“真”。

在 C99 之前，C 语言没有逻辑型数据（C++ 有逻辑变量和逻辑型常量，以 True 表示“真”，以 False 表示“假”）。在 C 的关系运算中，以 1 代表“真”，以 0 代表“假”。例如，当 a=3，b=2，c=1 时，则：

- 关系表达式“a>b”的值为“真”，表达式的值为 1。
- 关系表达式“(a>b)==c”的值为“真”（因为 a>b 的值为 1，等于 c 的值），表达式的值为 1。
- 关系表达式“b+c<a”的值为“假”，表达式的值为 0。

说明：从本质上来说，关系运算的结果（即关系表达式的值）不是数值，而是逻辑值，但是由于 C 语言追求精练灵活，没有提供逻辑型数据（其他高级语言如 Pascal，FORTRAN，C++ 都允许定义和使用逻辑型数据，C99 也增加了逻辑型数据，用关键字 bool 定义逻辑型变量）。为了便于处理关系运算和逻辑运算的结果，C 语言以 1 代表“真”，以 0 代表“假”，并在编译系统中按此实现（这种规定只是 C 语言的特殊处理方法，不要误认为是所有计算机语言的普遍规则）。用 C 语言的人要注意这样的规定。

由于用了 1 和 0 代表真和假，而 1 和 0 又是数值，所以在 C 程序中还允许把关系运算的结果（即 1 和 0）看作和其他数值型数据一样，可以参加数值运算，或把它赋值给数值型变量。例如，若 a，b，c 的值为 3，2，1。请分析下面的赋值表达式：

```
d=a>b      d 的值为 1
f=a>b>c    f 的值为 0（因为">"运算符是自左至右的结合方向，先执行 a>b，得值为 1，再执
           行关系运算 1>c，得值 0，赋给 f）
```

这是 C 的灵活性的一种表现，允许把关系表达式作为一般数值来处理，对有经验的人，可以利用它实现一些技巧，使程序精练专业，但是对初学者来说，可能会不好理解，容易弄错。在学习阶段，还是应当强调程序的清晰易读，不要写出别人不懂的程序。

3.3 选择结构中的逻辑运算

有时需要判断的条件不是一个简单的条件，而是一个复合的条件，如：

- 是中国公民，且在 18 岁以上才有选举权。这就要求同时满足两个条件：中国公民和大于 18 岁。
- 5 门课都及格，才能升级。这就要求同时满足 5 个条件。
- 70 岁以上的老人和 10 岁以下儿童，入公园免票。这就要对入园者检查两个条件，即 age>70 或 age<10，必须满足其中之一。

以上问题仅用一个关系表达式是无法表示的，需要用一个逻辑运算符把两个关系表

达式组合在一起才能处理。在 BASIC 和 Pascal 语言中用 AND,OR 和 NOT 作为逻辑运算符,分别代表逻辑运算符“与”“或”“非”。

例如:

```
(a>b) AND (x>y)
```

其中的 AND 是**逻辑运算符**,代表“与”,即运算符两侧的关系表达式(或其他逻辑量)的值都为“真”(二者的条件都满足)。上面表达式的意思是:“a>b”与“x>y”两个条件都**同时满足**。如果已知 a>b 和 x>y,则上面的表达式的值为“真”。

3.3.1　逻辑运算符及其优先次序

在 C 语言中不直接用 AND,OR 和 NOT 作为逻辑运算符,而用其他符号代替,见表 3.1。

表 3.1　C 语言逻辑运算符及其含义

运算符	含义	举例	说　明
&&	逻辑与	a && b	如果 a 和 b 都为真,则结果为真,否则为假
‖	逻辑或	a ‖ b	如果 a 和 b 有一个或一个以上为真,则结果为真,二者都为假时,结果为假
!	逻辑非	!a	如果 a 为假,则!a 为真,如果 a 为真,则!a 为假

“&&”和“‖”是双目(元)运算符,它要求有两个运算对象(操作数),如(a>b)&&(x>y)和(a>b)‖(x>y)。“!”是一目(元)运算符,只要求有一个运算对象,如!(a>b)。

表 3.2 为逻辑运算的“真值表”。用它表示当 a 和 b 的值为不同组合时,各种逻辑运算得到的值。

表 3.2　逻辑运算的真值表

a	b	!a	!b	a && b	a ‖ b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

怎样看这个表呢?以表中第 2 行为例,当 a 为真,b 为假时,!a 为假,!b 为真,a && b 为假,a ‖ b 为真。这是很简单的,也是最基本的。

如果在一个逻辑表达式中包含多个逻辑运算符。如:

!a && b ‖ x>y && c。怎样确定它的运算顺序呢?C 语言规定按以下的优先顺序:

(1)!(非)→&&(与)→‖(或),即“!”为三者中最高的。

(2)逻辑运算符中的“&&”和“‖”低于关系运算符,“!”高于算术运算符,见图 3.2。

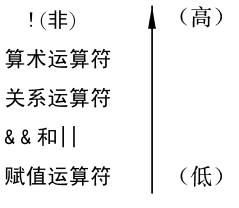


图　3.2

例如：

(a>b) && (x>y)可写成 a>b && x>y

(a==b) || (x==y)可写成 a==b || x==y

(!a) || (a>b)可写成!a || a>b

3.3.2 逻辑表达式

用逻辑运算符将关系表达式或逻辑量连接起来的式子就是逻辑表达式。

逻辑表达式的值是一个逻辑量“真”或“假”。前已说明：C 语言编译系统在表示逻辑运算结果时，以数值 1 代表“真”，以 0 代表“假”。但是在判断一个逻辑量是否为“真”时，测定它的值是 0 还是非 0，如果是 0 就代表它为“假”，如果是非 0 则认为它是“真”。因为逻辑量只有两种可能值，所以把被测定的对象划分为两种情况（0 和非 0），以便于处理。

例如：

（1）若 a=4，则!a 的值为 0。因为 a 的值为非 0，被认作“真”，对它进行“非”运算，得“假”，“假”以 0 代表。

（2）若 a=4，b=5，则 a&&b 的值为 1。因为 a 和 b 均为非 0，被认为是“真”，因此 a&&b 的值也为“真”，值为 1。

（3）a，b 的值分别为 4 和 5，则 a || b 的值为 1。因为 a 和 b 均为非 0，即“真”。

（4）a，b 的值分别为 4，5，则!a || b 的值为 1。因为!a 为“假”，而 b 为“真”。

（5）4&&0 || 2 的值为 1。因为 4&&0 为“假”而 2 为非 0，故进行“或”运算结果为“真”。

通过这几个例子可以看出，由系统给出的逻辑运算结果不是 0 就是 1，不可能是其他数值。而在逻辑表达式中作为参加逻辑运算的运算对象（操作数）可以是 0（“假”）或任何非 0 的数值（按“真”对待）。如果在一个表达式中不同位置上出现数值，应区分哪些作为数值运算或关系运算的对象，哪些作为逻辑运算的对象。例如：

```
5>3 && 8<4-!0
```

表达式自左至右扫描求解。首先处理“5>3”（因为关系运算符优先于逻辑运算符“&&”）。在关系运算符两侧的 5 和 3 作为数值参加关系运算，“5>3”的值为 1（代表真）。再进行“1 && 8<4-!0”的运算，8 的左侧为“&&”，右侧为“<”运算符，根据优先规则，应先进行“<”的运算，即先进行 8<4-!0 的运算。而 4 的左侧为“<”，右侧为“-”运算符，而“-”优先于“<”，因此应先进行“4-!0”的运算，由于“!”的级别最高，因此先进行“!0”的运算，得到结果 1。然后进行“4-1”的运算，得到结果 3，再进行“8<3”的运算，得 0，最后进行“1 && 0”的运算，得 0。

实际上，逻辑运算符两侧的运算对象不但可以是 0 和 1，或者是 0 和非 0 的整数，也可以是字符型、实型或指针型等数据。系统最终以 0 和非 0 来判定它们属于“真”或“假”。例如：

```
'c'&& 'd'
```

的值为 1(因为'c' 和'd' 的 ASCII 码都不为 0,按“真”处理),所以 1 && 1 的值为 1。

可以将表 3.2 改写成表 3.3 的形式。

表 3.3　逻辑运算的真值表

a	b	!a	!b	a && b	a ‖ b
非 0	非 0	0	0	1	1
非 0	0	0	1	0	1
假	非 0	1	0	0	1
假	0	1	1	0	0

说明:在计算机对逻辑表达式的求解中,并不是所有的逻辑运算符都被执行,只是在必须执行下一个逻辑运算符才能求出表达式的解时,才执行该运算符。例如:

(1) a && b && c。只有 a 为真(非 0)时,才需要判别 b 的值,只有 a 和 b 都为真的情况下才需要判别 c 的值。只要 a 为假,就不必判别 b 和 c(此时整个表达式已确定为假)。如果 a 为真,b 为假,不判别 c,见图 3.3。

(2) a ‖ b ‖ c。只要 a 为真(非 0),就不必判断 b 和 c。只有 a 为假,才判别 b。a 和 b 都为假才判别 c,见图 3.4。

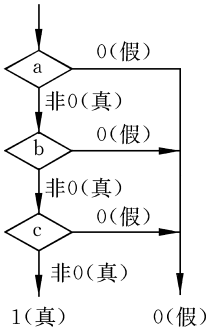


图　3.3

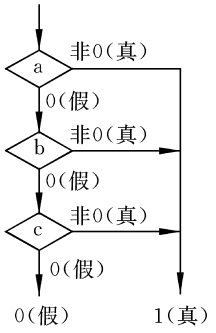


图　3.4

也就是说,对 && 运算符来说,只有 a≠0,才继续进行右面的运算。对 ‖ 运算符来说,只有 a 等于 0,才继续进行其右面的运算。因此,如果有下面的逻辑表达式:

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

当 a=1,b=2,c=3,d=4,m 和 n 的原值为 1 时,由于“a>b”的值为 0,因此 m=0,而“n=c>d”不被执行,因此 n 的值不是 0 而仍保持原值 1。这点请读者注意。

熟练掌握 C 语言的关系运算符和逻辑运算符后,可以巧妙地用一个逻辑表达式来表示一个复杂的条件。

例如,要判别用 year 表示的某一年是否是闰年。闰年的条件是符合下面二者之一:

①能被 4 整除,但不能被 100 整除,如 2016。②能被 4 整除,又能被 400 整除,如 2000(注意,能被 100 整除,不能被 400 整除的年份不是闰年,如 2100)。可以用一个逻辑表达式来表示:

```
(year %4==0 && year %100 !=0) ||year %400==0
```

当 year 为某一整数值时，如果上述表达式值为真（1），则 year 为闰年；否则 year 为非闰年。

可以加一个“!”用来判别非闰年：

```
!((year %4==0 && year %100 !=0)||year %400==0)
```

若此表达式值为真（1），year 为非闰年。也可以用下面逻辑表达式判别非闰年：

```
(year %4 !=0)||(year %100==0 && year %400!=0)
```

若表达式值为真，year 为非闰年。请注意表达式中右边的一对括号内的不同运算符（%，!=，&&，==）的运算优先次序。

3.4 用 if 语句实现选择结构

有了以上的基础，就可以顺利地利用选择结构进行编程了。在 C 语言中，可以用不同的方法实现选择结构（包括 if 语句、条件表达式、switch 语句等），其中 if 语句是最基本的，用得最多。本节先介绍 if 语句。在 if 语句中包含一个逻辑表达式，用它判定所给定的条件是否满足，并根据判定的结果（真或假）决定选择执行哪一种操作（在 if 语句中给出两种可能的选择）。

3.4.1 if 语句的三种形式

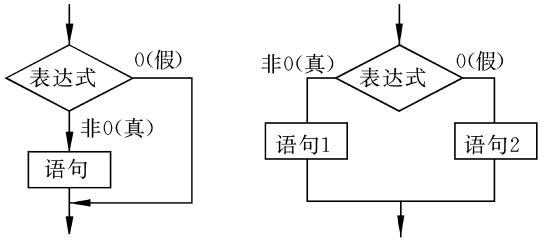
C 语言提供了三种形式的 if 语句供用户选用。

1. if（表达式）语句

例如：

```
if(x>y) printf("%d\n",x);
```

这种 if 语句的执行过程见图 3.5（a）。



（a）　　（b）

图　3.5

2. if（表达式）语句 1 else 语句 2

例如：

```
if (x>y)
    printf("%d\n",x);
else
    printf("%d\n",y);
```

这种 if 语句的执行过程见图 3.5(b)。

3.

if（**表达式 1**）**语句 1**
else if（**表达式 2**）**语句 2**
else if（**表达式 3**）**语句 3**
　　⋮
else if（**表达式 m**）**语句 m**
else 语句 n

流程图见图 3.6。

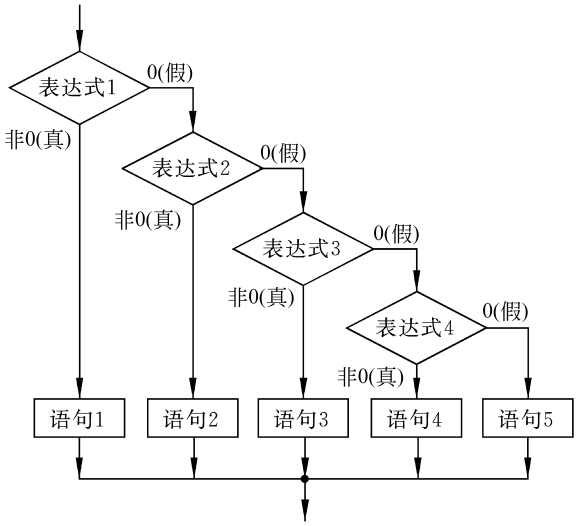


图　3.6

例如：

```
if (number>500) cost=0.15;
else if (number>300) cost=0.10;
else if (number>100) cost=0.075;
else if (number>50) cost=0.05;
else cost=0;
```

说明：

(1) 3 种形式的 if 语句中在 if 后面都有“表达式”,一般为逻辑表达式或关系表达式。例如：

```
if(number>300 && number<=500) cost=0.10;
```

在执行 if 语句时先对括号中的表达式求解，若表达式的值为 0，按“假”处理，若表达式的值为非 0，按“真”处理，执行指定的语句。假如有以下 if 语句：

```
if(3) printf("OK");
```

是合法的，执行结果输出 OK，因为表达式的值为 3，按“真”处理。由此可见，表达式的类型不限于逻辑表达式，可以是任意的数值类型（包括整型、实型、字符型、指针型数据等）。下面的 if 语句也是合法的：

```
if('a') printf("%d",'a');
```

执行时输出'a' 的 ASCII 码 97。

（2）if 语句中有内嵌语句，每个内嵌语句都要以分号结束。例如：

```
if (x>0)
    print ("%f\n",x);
else
    printf("%f\n",-x);
```

行末各有一个分号（;）

分号是 C 语句中不可缺少的部分，即使是 if 语句中的内嵌语句也不能例外。如果无此分号，则出现语法错误。读者可以上机试验一下。

（3）不要误认为上面是两个语句（一个 if 语句和一个 else 语句）。它们都是属于同一个 if 语句。else 子句不能作为独立语句单独使用，它只能是 if 语句的一部分，与 if 配对使用。

（4）在 if 和 else 后面可以只含一个内嵌的操作语句（如上例），也可以有多个操作语句，但应当用花括号（{}）将几个语句括起来成为一个复合语句。例如：

```
if (a+b>c && b+c>a && c+a>b)
  {
    s=0.5*(a+b+c);
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("area=%6.2f",area);
  }
else
  printf("it is not a trilateral");
```

注意在 else 上面一行的右花括号（}）外面不需要再加分号。因为{}内是一个完整的复合语句，不需要另附加分号。

3.4.2 if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句称为 if 语句的嵌套。一般形式如下：

```
if( )
    if( ) 语句 1
    else 语句 2
else
```

内嵌 if

if( ) 语句 3
else 语句 4　　} 内嵌 if

应当注意 if 与 else 的配对关系。else 总是与它上面的最近的未配对的 if 配对。假如写成：

```
if()
  if()语句 1
else
  if() 语句 2
else 语句 3
```

内嵌 if

编程序者把第一个 else 写在与第一个 if(外层 if)同一列上,希望第一个 else 与第一个 if 对应,但实际上第一个 else 是与第二个 if 配对的,因为它们相距最近。写成这样的锯齿形式并不能改变 if 语句的执行规则。这个 if 语句实际的配对关系表示如下：

```
if()
  if()语句 1
  else
    if() 语句 2
    else 语句 3
```

内嵌 if

因此最好使外层 if 和内嵌 if 都包含 else 部分(如 3.4.2 节最早列出的形式),即

if( )
　if( )语句 1
　else
　　if( ) 语句 2
　　else　语句 3
else 语句 4

这样 if 的数目和 else 的数目相同,从内层到外层一一对应,不容易出错。

如果 if 与 else 的数目不一样,为实现程序设计者的意图,可以加花括号来确定配对关系。例如：

```
if ()
    { if () 语句 1}          //内嵌 if
else 语句 2
```

这时“{ }”限定了内嵌 if 语句的范围,“{ }”内是一个完整的 if 语句。因此 else 与第一个 if 配对。

通过下面的例子可以具体地了解如何正确地使用 if 的嵌套。

【例 3.3】　有一函数：

$$y=\begin{cases}-1 & (x<0)\\ 0 & (x=0)\\ 1 & (x>0)\end{cases}$$

编程序,要求输入一个 x 值后,输出 y 值。

解题思路：先用伪代码写出算法。

```
输入 x
若 x<0,则 y=-1
若 x=0,则 y=0
若 x>0,则 y=1
输出 y
```

或

```
输入 x
若 x<0,则 y=-1
```

否则

```
若 x=0,则 y=0
若 x>0,则 y=1
输出 y
```

也可以用流程图表示，见图 3.7。

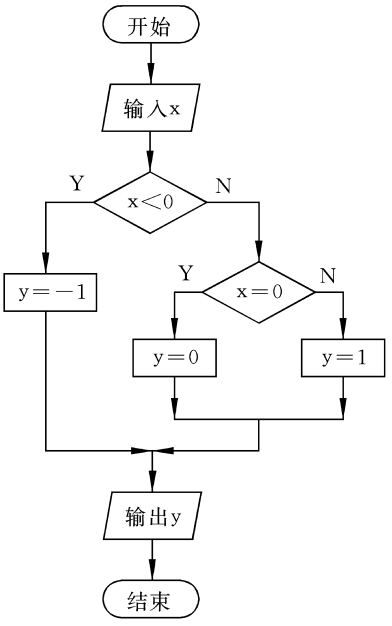


图 3.7

编写程序：按照上面的算法，有人用 C 语言写出以下几个不同的程序，请读者分析哪个是正确的。

程序 1：

```
#include<stdio.h>
int main()
{ int x,y;
  printf("enter x: ");
  scanf("%d",&x);
  if(x<0)
      y=-1;
  else
      if(x==0) y=0;
      else y=1;
  printf("x=%d,y=%d\n",x,y);
  return 0;
}
```

程序 2：将程序 1 的 if 语句(第 6~10 行)改为

```
if(x>=0)
  if(x>0) y=1;
  else y=0;
else y=-1;
```

程序 3：将上述 if 语句改为

```
y=-1;
if(x!=0)
```

```
  if(x>0) y=1;
else y=0;
```

程序 4：将上述 if 语句改为

```
y=0;
if(x>=0)
  if(x>0) y=1;
else y=-1;
```

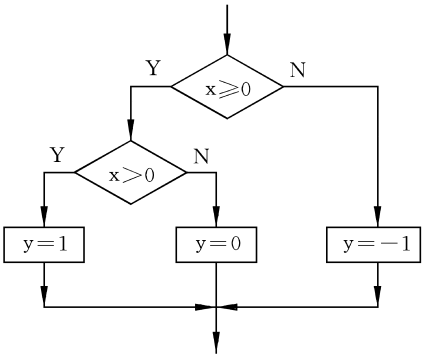


图　3.8

读者可以分别画出程序 1～程序 4 的流程图，便可以判断出：只有程序 1 和程序 2 是正确的。图 3.7 是程序 1 的流程图，显然它是正确的。图 3.8 是程序 2 的流程图，它也能实现题目的要求。

程序 3 的流程图见图 3.9，程序 4 的流程图见图 3.10，它们是不能实现题目要求的。请注意程序中的 else 与 if 的配对关系，例如程序 3 中的 else 子句是和它上一行的内嵌的 if 语句配对，而不与第 2 行的 if 语句配对。

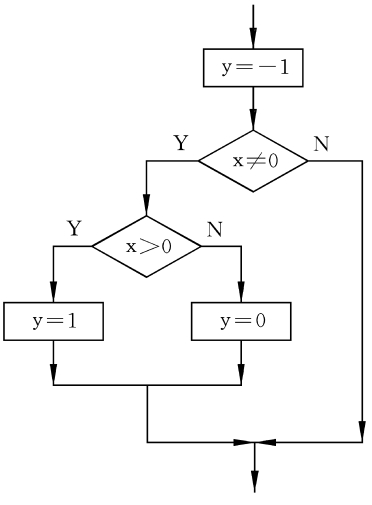


图　3.9

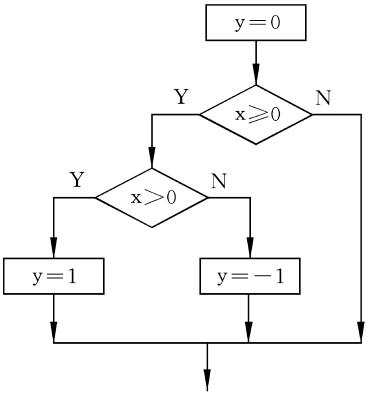


图　3.10

为了使逻辑关系清晰，避免出错，一般把内嵌的 if 语句放在外层的 else 子句中（如程序 1 那样），这样由于有外层的 else 相隔，内嵌的 else 不会被误认为和外层的 if 配对，而只能与内嵌的 if 配对，这样就不会搞混，如像程序 3 和程序 4 那样写就很容易出错。

* 3.5　用条件表达式实现选择结构

有时，在 if 语句中，在被判别的条件为“真”或“假”时，都用一个赋值语句向同一个变量赋值。例如：

```
if(a>b)
    max=a;
else
    max=b;
```

当 a>b 时将 a 的值赋给 max，当 a≤b 时将 b 的值赋给 max。可以看到无论 a>b 是否满足，都是向同一个变量赋值。此时可以用**条件表达式**来处理，使程序更简练。

上面的 if 语句可以用以下的语句代替：

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

其中，赋值号“=”右侧的“(a>b)？a：b”是一个“条件表达式”。它是这样执行的：如果(a>b)条件为真，则条件表达式的值为 a；否则取值 b。然后把此值赋给 max 变量。

条件表达式的一般形式为

表达式 1？表达式 2：表达式 3

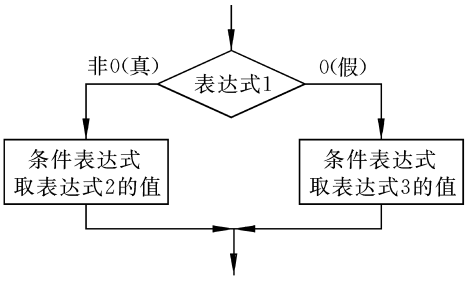


图 3.11

在条件表达式“(a>b)?a：b”中，a>b 是“表达式 1”，变量 a 是“表达式 2”，变量 b 是“表达式 3”。条件表达式中的“?”和“：”一起构成**条件运算符**。条件运算符“?:”要求有 3 个操作对象，称三目(元)运算符。它是 C 语言中唯一的三目运算符。它的执行过程见图 3.11。

可以看出，条件表达式也是一个选择结构。它和 if 语句不同之处在于：它不能执行任意的内嵌语句(如输入输出)，而只是使条件表达式取不同的值。一般的用法是将条件表达式的值赋给一个变量(如上面的 max)。

说明：

(1) 条件运算符的执行顺序：先求解表达式 1，若为非 0(真)则求解表达式 2，此时表达式 2 的值就作为整个条件表达式的值。若表达式 1 的值为 0(假)，则求解表达式 3，表达式 3 的值就是整个条件表达式的值。下面的赋值表达式

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

执行结果就是将条件表达式的值赋给 max，也就是将 a 和 b 二者中的大者赋给 max。

(2) 条件运算符优先于赋值运算符，因此上面赋值表达式的求解过程是先求解条件表达式，再将它的值赋给 max。

条件运算符的优先级别比关系运算符和算术运算符都低。因此，

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

其中的括号可以不要，可写成

```
a>b?a:b
```

前面加括号是为了清晰，便于理解。如果有

```
a>b?a:b+1
```

相当于

```
a>b?a:(b+1)
```

而不相当于

```
(a>b?a:b)+1
```

(3) 条件运算符的结合方向为“自右至左”。如果有以下条件表达式：

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

相当于

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

先求解右边的条件表达式。如果 a=1,b=2,c=3,d=4,则条件表达式的值等于 4。

(4) 条件表达式还可以写成以下形式：

```
a>b?(a=100):(b=100)                    (表达式 2 和表达式 3 是赋值表达式)
```

或

```
a>b?printf("%d",a): printf("%d", b)    (表达式 2 和表达式 3 是函数表达式)
```

即“表达式 2”和“表达式 3”不仅可以是数值表达式,还可以是赋值表达式或函数表达式。最下面的条件表达式相当于以下 if…else 语句：

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

(5) 条件表达式中,表达式 1 的类型可以与表达式 2 和表达式 3 的类型不同。例如：

```
x?'a': 'b'
```

如果整型变量 x 的值为非 0,条件表达式的值为'a' ,如为 0,条件表达式的值为'b' 。

表达式 2 和表达式 3 的类型也可以不同,此时条件表达式的值的类型为二者中较高的类型。例如：

```
x>y?1:1.56
```

如果 x≤y,则条件表达式的值为 1.56,若 x>y,值应为 1,由于 1.56 是实型,比整型高,因此,将 1 转换成实型值 1.0。表达式的值是实数 1.0。

以上规则不必死记,有了此概念,必要时查一下即可。

【例 3.4】 输入一个字符,判别它是否是大写字母,如果是,将它转换成小写字母;如果不是,不转换。然后输出最后得到的字符。

解题思路：首先需要判别字母的大小写,因此要用选择结构,判别的结果只有两种可能,都把它放在一个字符变量中输出。这种情况用条件表达式处理最方便。

编写程序：

```
#include<stdio.h>
int main()
  { char ch;
    scanf("%c",&ch);
    ch=(ch>='A'&& ch<='Z')?(ch+32): ch;
    printf("%c\n",ch);
    return 0;
  }
```

运行结果：

```
A↙
a
```

程序分析：条件表达式“(ch>='A' && ch<='Z')?(ch+32)：ch”的作用是，如果字符变量 ch 的值为大写字母，则条件表达式的值为(ch+32)，即相应的小写字母，32 是小写字母和大写字母 ASCII 码的差值。如果 ch 的值不是大写字母，则条件表达式的值为 ch，即不进行转换。最后输出 ch 的值(必然是小写字母)。

善于利用条件表达式，可以使程序写得精练、专业。

3.6 利用 switch 语句实现多分支选择结构

if 语句只有两个分支可供选择，而实际问题中常常需要用到多分支的选择。例如，学生成绩分类(85 分及以上为'A' 等，70～84 分为'B' 等，60～69 分为'C' 等)；人口统计分类(按年龄分为老、中、青、少、儿童)；工资统计分类；银行存款分类等，当然这些都可以用嵌套的 if 语句来处理，但如果分支较多，则嵌套的 if 语句层数多，程序冗长而且可读性降低。C 语言提供 switch 语句用来处理多分支选择。它的一般形式如下：

```
switch(表达式)
{
  case 常量表达式 1：语句 1
  case 常量表达式 2：语句 2
        ⋮
  case 常量表达式 n：语句 n
  default          ：语句 n+1
}
```

【例 3.5】 要求按照考试成绩的等级输出百分制分数段，A 等为 85 分及以上，B 等为 70～84 分，C 等为 60～69 分，D 等为 60 分以下。成绩的等级由键盘输入。

解题思路：这是一个多分支选择问题，根据百分制分数将学生成绩分为 4 个等级，如果用 if 语句来处理至少要用 3 层嵌套的 if，进行 3 次检查判断。可以用 switch 语句，进行一次检查即可得到结果。

编写程序：

```
#include<stdio.h>
int main()
  {
    char grade;
    scanf("%c",&grade);
    printf("Your score: ");
    switch(grade)
    {
      case 'A': printf("85~100\n");break;
      case 'B': printf("70~84\n");break;
      case 'C': printf("60~69\n");break;
      case 'D': printf("<60\n");break;
      default: printf("data error!\n");
    }
    return 0;
  }
```

运行结果：

```
A↙                          (从键盘输入大写字母 A,按 Enter 键)
Your score: 85~100          (输出对应的分数段)
```

程序分析：定义 grade 为字符变量,从键盘输入一个大写字母,赋给变量 grade,switch 得到 grade 的值并把它和各 case 中给定的值('A' ,'B' ,'C' ,'D' 之一)相比较,如果和其中之一相同(称为匹配),则执行该 case 后面的语句(即 printf 语句),输出相应的信息。如果输入的字符与'A' ,'B' ,'C' ,'D' 都不相同,就执行 default 后面的语句,输出“输入数据有错”的信息。注意在每个 case 后面的语句中,最后都有一个 break 语句,它的作用是使流程转到 switch 语句的末尾(即右花括号处)。流程图见图 3.12。

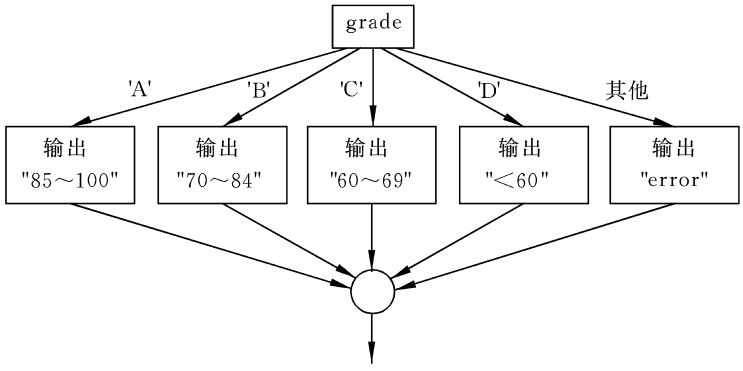


图　3.12

说明：

(1) switch 后面括号内的“表达式”,表达式的值应为整型类型(包括字符型)。

（2）switch 下面的花括号内是一个复合语句。这个复合语句包括若干语句，它是 switch 语句的语句体。语句体内包含多个以关键字 case 开头的子句和最多一个以 default 开头的子句。case 后面跟一个常量（或常量表达式），如：case 'A'，它们和 default 都是起标号（label 或称标签、标记）的作用，用来标志一个位置。执行 switch 语句时，先计算 switch 后面的“表达式”的值，然后将它与各 case 标号比较，如果与某一个 case 标号中的常量相同，流程就转到此 case 标号后面的语句。如果没有与 switch 表达式相匹配的 case 常量，流程转去执行 default 标号后面的语句。

（3）可以没有 default 标号，此时如果没有与 switch 表达式相匹配的 case 常量，则不执行任何语句，流程转到 switch 语句的下一个语句。

（4）每一个 case 常量表达式的值必须互不相同；否则就会出现互相矛盾的现象（对表达式的同一个值，有两种或多种执行方案）。

（5）各个 case 和 default 的出现次序不影响执行结果。例如，可以先出现“default：…”，再出现“case 'D'：…”，然后是“case 'A'：…”。

（6）case 标号只起标记的作用。在执行 switch 语句时，根据 switch 表达式的值找到匹配的入口标号，并不在此进行条件检查，在执行完一个 case 标号后面的语句后，就从此标号开始执行下去，不再进行判断。例如在例 3.6 中，如果在各 case 子句中没有 break 语句，将连续输出：

```
Your score: 85~100
70~84
60~69
<60
enter data error!
```

注意：一般情况下，在执行一个 case 子句后，应该用 break 语句使流程跳出 switch 结构，即终止 switch 语句的执行。最后一个子句（今为 default 子句）可以不加 break 语句，因为流程已到了 switch 结构的结束处。

（7）在加了 break 语句后，在 case 子句中虽然包含了一个以上执行语句，但可以不必用花括号括起来，会自动顺序执行本 case 子句中的所有的执行语句，当然加上花括号也可以。

（8）多个 case 标号可以共用一组执行语句。例如：

```
case 'A':
case 'B':
case 'C': printf(">=60\n");break;
```

当 grade 的值为'A'，'B'，'C' 时都执行同一组语句，输出“>=60”，然后换行。

3.7 选择结构程序综合举例

以上介绍了选择结构的算法以及 C 语言实现选择结构的语句，在此基础上可以进一步学习编写包含选择结构的 C 程序。

【例 3.6】 写程序,判断某一年是否是闰年。

解题思路: 前面已介绍过判别闰年的方法。现在用图 3.13 来表示判别闰年的算法(用 N-S 图表示算法)。用 N-S 图表示多级选择结构,简单清晰,层次分明。以变量 leap 代表是否是闰年的信息。根据闰年规则逐项进行判断,最后若判定是闰年,就令 leap=1;若非闰年,令 leap=0。最终检查 leap 是否为 1(真),若是,则输出“闰年”信息。

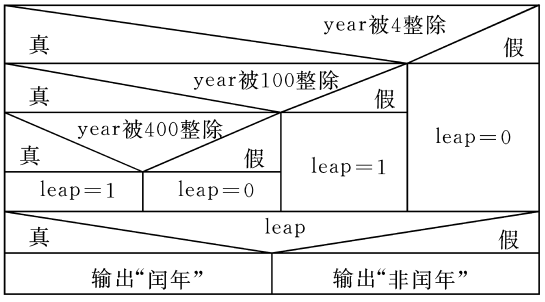


图　3.13

编写程序:

```
#include<stdio.h>
int main()
  {
    int year,leap;
    printf("please enter a year: ");
    scanf("%d",&year);
    if (year%4==0)
      {
        if (year%100==0)
          {
            if (year%400==0)
              leap=1;
            else
              leap=0;
          }
        else
          leap=1;
      }
    else
      leap=0;
    if (leap)
      printf("%d is",year);
    else
      printf("%d is not",year);
```

```
    printf("a leap year.\n");
    return 0;
  }
```

运行结果：

```
① please enter a year: 2016↙
  2016 is a leap year.
② please enter a year: 2100↙
  2100 is not a leap year.
```

程序分析：请仔细分析 if 与 else 的配对关系。为了使程序结构清晰，便于他人阅读，也便于日后自己维护，在写程序时应尽量写成锯齿形式，内嵌语句向右缩进 2 列，同一层次的成分（如同一层的 if 和 else）出现在同一列上。

也可以将程序中第 7~20 行改写成以下的 if…else if…else 形式语句（本章 3.4.1 节中介绍的第 3 种 if 语句）：

```
if(year%4!=0)
  leap=0;
else if(year%100!=0)
  leap=1;
else if(year%400!=0)
  leap=0;
else
  leap=1;
```

也可以用一个逻辑表达式包含所有的闰年条件（在本章 3.3.2 节的最后已有介绍），将上述 if 语句用下面的 if 语句代替：

```
if((year %4==0 && year %100 !=0)||(year %400==0))
  leap=1;
else
  leap=0;
```

【例 3.7】 求 $ax^2+bx+c=0$ 方程的解。要求能处理任何的 a，b，c 值的组合。

解题思路：在第 2 章例 2.3 中曾处理此问题，但前提是 a，b，c 的值满足判别式 b^2-4ac 大于或等于 0 的情况，即方程应有有理解。但是根据代数知识，应该有以下几种可能。

① $a=0$，不是二次方程，而是一次方程。

② $b^2-4ac=0$，有两个相等的实根。

③ $b^2-4ac>0$，有两个不等的实根。

④ $b^2-4ac<0$，有两个共轭复根。

画出 N-S 流程图表示算法（见图 3.14），可以看到，用 N-S 流程图表示算法，很容易理解，一目了然。