

第 5 章



选择结构程序设计

本章内容提要

- (1) 关系表达式和逻辑表达式。
- (2) if 语句。
- (3) 条件表达式。
- (4) switch 语句。

第 4 章讲述的是顺序结构程序设计,顺序结构的程序总是按照由上而下的顺序来执行。但是,并非所有的问题都可以用顺序结构的程序来解决。有时一个问题的求解需要根据情况的不同选择不同的处理方式,这就要用到选择结构。

本章先介绍选择结构的必要知识:关系运算和逻辑运算,然后讲解 if 语句、条件表达式及 switch 语句。



关系运算符和关系表达式

5.1 关系运算符和关系表达式

选择结构需要根据条件决定程序的走向,而条件通常是关系表达式或逻辑表达式。本节先介绍关系运算符和关系表达式。

5.1.1 关系运算符

C 语言中的关系运算符有 6 个: $>$ 、 $>=$ 、 $<$ 、 $<=$ 、 $==$ 、 $!=$ 。

 **说明:** $>=$ 代表 $\geq$, $<=$ 代表 $\leq$, $==$ 代表“等于”, $!=$ 代表“不等于”。

它们都是双目运算符,结合性都是自左至右。

它们的级别都比算术运算符低,都比赋值运算符和逗号运算符高。6 个运算符中, $>$ 、 $>=$ 、 $<$ 、 $<=$ 的级别相同, $==$ 、 $!=$ 的级别相同,前 4 个级别高于后两个。

 **说明:** 要详细了解各种运算符的优先级别,可参阅附录 E。

5.1.2 关系表达式

1. 关系表达式

如果一个表达式最后进行的是关系运算,则该表达式就是关系表达式。



下面几个表达式中,前3个是关系表达式,第4个是赋值表达式,最后一个逗号表达式。

```
c>a+b           //运算符由高到低的顺序是+、>
a>b==c          //运算符由高到低的顺序是>、==
a==b<c          //运算符由高到低的顺序是<、==
a=b>c           //运算符由高到低的顺序是>、=
a>b==c,a=b!=c   //运算符由高到低的顺序是>、==和!=、=、,
```

2. 关系表达式的值

关系表达式的值是一个逻辑值,数学中用“真”“假”来表示,C语言中用1和0来表示。当表达式成立时,其值为1;当表达式不成立时,其值为0。

设变量a、b、c的值分别是1、2、3,则下面几个表达式的值分别如注释中所示。

```
c>=a+b          //关系表达式,值为1
a>b==c          //关系表达式,值为0
a==b<c          //关系表达式,值为1
a=b>c           //赋值表达式,值为0,因a的值为0,a为何为0?因b>c不成立
a>b==c,a=b!=c   //逗号表达式,值为1,执行后a的值为1
f=c>b>a         //赋值表达式,值为0,因c>b>a的结果为0
```

需要指出的是,表达式的求解,一次只能进行一个运算符的运算,例如:

```
c>b>a
```

相当于是:

```
(c>b)>a
```

求解时要先算 $c>b$,得到一个结果(结果是1),然后再拿这个结果与a比较,即 $1>a$,显然不成立,故 $c>b>a$ 的结果是0(假)。



注意: 计算 $c>b>a$ 时,最后与a比较的是 $c>b$ 的结果,而不是b。



考考你: 表达式 $3==3==3$ 的值是多少?

5.2 逻辑运算符和逻辑表达式

一个复杂的条件往往是由几个简单的条件组成的,这些简单条件之间通常用逻辑运算符来连接,从而构成逻辑表达式。

5.2.1 逻辑运算符

C语言中的逻辑运算符有3个: &&(逻辑与)、||(逻辑或)、!(逻辑非)。

逻辑与(&&)是“并且”的意思,表示只有两边的条件都成立,“与”的结果才是1。例如:



逻辑运算符和逻辑表达式



```
a>=b&&a>=c           //命题: a 是最大的
```

逻辑或(`||`)是“或者”的意思,表示只要有一个条件成立,结果就是 1。例如:

```
a>b||a>c               //命题: a 不是最小的
```

逻辑非(!)是“否定”“取反”的意思,如果表达式原来是“真”(1),取反之后就是“假”(0);如果表达式原来是“假”,则取反后就是“真”。例如:

```
!(5>3)                 //值为 0
```

```
!(2==1)                //值为 1
```

上面两个逻辑表达式的值分别是 0、1。

`&&` 和 `||` 都是双目运算符, `!` 是单目运算符。它们的优先级别如图 5-1 所示。

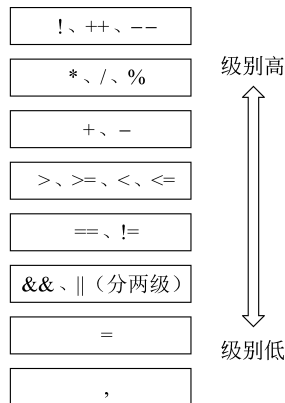


图 5-1 运算符的优先级

5.2.2 逻辑表达式

1. 逻辑表达式及其逻辑值

如果一个表达式最后进行的是逻辑运算,那么该表达式就是逻辑表达式。逻辑表达式的值也是逻辑值:真或假,如前所述,C 语言中用 1 和 0 表示真和假。

2. 逻辑运算符的运算对象

逻辑表达式中,参与逻辑运算的运算对象通常都是条件表达式,例如,判断一个年份是否是闰年的表达式:

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

但实际上,任何表达式都可以参与逻辑运算,例如:

```
'A'&&b=3
```

```
!3&&5-2||'A'&&(a=1,b=2,a+b)&&(a=b)
```

说明: 不管什么类型的表达式,当它参与逻辑运算时,用的都是它的逻辑值。

那么,怎么确定这些表达式的逻辑值?例如, `'A'` 的逻辑值是真还是假? `b=3` 的逻辑值又是什么?

C 语言中,任何表达式(包括变量、常量)都有逻辑值,只要表达式的值非 0,则认为它的逻辑值为“真”(即 1);只有表达式的值为 0 时,其逻辑值才为“假”(即 0)。例如:

- (1) `3+2` 的值是 5,若参与逻辑运算,则取它的逻辑值 1。
- (2) `-5` 参与逻辑运算时,逻辑值也是 1。
- (3) 若“`float x=1.2;`”,则 `x` 的逻辑值为 1。
- (4) `!'A'` 的逻辑值为 0。
- (5) `x=-2` 的逻辑值为 1。



(6) 设 a, b 都是 `int` 型变量, 则 $(a=1, b=2, a-b+1)$ 的逻辑值是 0。

(7) $5 > 3 \&\& 8 < 4 - !0$ 的逻辑值为 0。

 **考考你**: 表达式 $!3 \&\& 5 - 2 || 'A' \&\& (a=1, b=2, a+b) \&\& (a=b)$ 的值是多少? 表达式 $(a=b=c=3) \&\& a==b==c$ 的值又是多少?

3. 逻辑表达式求解时的短路效应

逻辑与(`&&`)和逻辑或(`||`)的两侧都各有一个式子, 在求解包含 `&&` 或 `||` 的表达式时, 都是先计算运算符左边的式子, 一旦能确定整个表达式的值, 右边就不再求解了。例如:

```
int a=1, b=10, c=2;
printf("%d, ", (a=b) || (c=b));
printf("%d, %d, %d\n", a, b, c);
```

运行结果:

1, 10, 10, 2

可以看到, c 的值还是原来的 2, 说明 $c=b$ 并没有被执行。

原因: 表达式 $(a=b) || (c=b)$ 的计算顺序是先求解 `||` 左侧的 $a=b$, 其值是 10 (因 $a=b$ 是赋值), 其逻辑值是 1, 由于 `||` 左侧已经为真, 整个逻辑表达式的值已经可以确定, 右侧 $c=b$ 不需要再求解了 (这称为短路效应), 故 c 的值是原来的 2 不变。

同理, 对于 `&&` 运算符, 一旦左侧为 0, 也不需要求解右侧的值 (短路效应)。

 **编程经验**: 在编写包含运算符 `&&` 的表达式时, 把最有可能为假的简单条件写在表达式的最左边, 在编写包含运算符 `||` 的表达式时, 把最有可能为真的简单条件写在表达式的最左边, 这样做有助于减少程序的运行时间, 提高程序的效率。

5.3 if 语句

选择结构可由两个语句来实现: `if` 语句和 `switch` 语句, 本节介绍 `if` 语句。

5.3.1 if 语句的格式

`if` 语句的格式如下:

```
if(表达式) //本行没有分号
    语句 1
[else
    语句 2]
```

 **说明**: `[]` 表示其中的内容可以省略, 即整个 `else` 分支可以省略。

 **注意**: 整个 `if` 语句格式所表示的部分, 是一条语句。



if 语句的两种格式



if 语句的执行过程：首先求解表达式的逻辑值，若为真，则执行语句 1，否则执行语句 2(若无 else 分支，则什么也不做)。

由 if 语句构成的选择结构使程序的走向形成了两个分支，程序执行时只能根据条件(表达式)是否成立选择其中一个分支，其流程如图 5-2 所示。

例 5.1 从键盘输入两个整数，若 $a > b$ 则计算 $a - b$ 的值，否则计算 $a + b$ 的值。

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

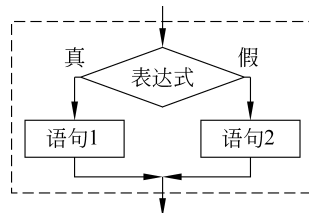


图 5-2 if 语句流程图

5.3.2 if 语句的使用说明

1. if 后面括号中的表达式可以是任何类型的表达式

if 后面括号中的表达式，可以是逻辑表达式、关系表达式、算术表达式等，还可以是常量或变量。不管什么类型的表达式，都取它的逻辑值，例如：

```
if(a > b)
if(a == b)
if(a != 0 && b != 1)
if(1)           // 相当于 if(1 != 0), 这个条件总是成立的
if(x)           // 相当于 if(x != 0)
if(a = b)       // 相当于 if((a = b) != 0)
if(a/10)        // 相当于 if((a/10) != 0)
```

例 5.2 从键盘输入一个整数，若不为 0，则输出 1；若为 0，则输出 0。
程序代码如下：

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);
    if(a)           // 相当于 if(a != 0)
        printf("1\n");
    else
```



```
    printf("0\n");  
    return 0;  
}
```

2. if 或 else 后面只能跟一条语句

语法上,if 或 else,都只能“管”一条语句,这条语句称为 if 的子句或 else 的子句。如果程序中需要 if 或 else“管”多条语句,则必须将它们放在花括号内组成一条复合语句。

例 5.3 从键盘输入两个整数,若都是偶数,则各自减半后输出它们,否则各自乘以 2 后输出它们。

程序代码如下:

```
#include <stdio.h>  
int main()  
{  
    int a,b;  
    scanf("%d%d",&a,&b);  
    if(a%2==0&&b%2==0)  
    {  
        a/=2;  
        b/=2;  
    }  
    else  
    {  
        a*=2;  
        b*=2;  
    }  
    printf("%d,%d\n",a,b);  
    return 0;  
}
```

 **提示:** C 语言中,任何能放置一条语句的地方,都可以放置一条复合语句。

 **说明:** 书写代码时,if 和 else 的子句,相对于 if 和 else,要缩进 4 个空格或一个 Tab 键跳过的距离,以示它们被 if 或 else 管辖。同一级别的语句,缩进的距离应该相同。代码的书写采用缩进格式是为了使程序清晰美观,缩进格式对编译没有影响,编译时这些空白都将被忽略。

 **试一试:** 把例 5.3 代码中 else 子句中的花括号去掉,运行程序输入 2 和 4,看结果是多少,为什么是这个结果? 单步调试找出原因。若把 if 子句中的花括号去掉,又会怎样?

3. if 和 else 的子句可以是空语句

当条件成立(或不成立)不需要做任何处理时,就可以写一个空语句。

例 5.4 从键盘输入一个整数,求其绝对值。

代码可以写成下面两种形式。



```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);
    if(a>=0)
        ;    //不做任何处理
    else
        a=-a;
    printf("%d\n", a);
    return 0;
}
```

或

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);
    if(a<0)
        a=-a;
    else
        ;    //不做任何处理
    printf("%d\n", a);
    return 0;
}
```

想一想：上面两段代码中，各有一条空语句(分号)，表示什么都不做，这两个分号可否删掉不要？

想一想：有些人编程，喜欢把上面代码中的空语句写成“a=a;”，即：

```
if(a<0)
    a=-a;
else
    a=a;    //该语句的执行过程是什么？让计算机做这个操作有没有意义？
```

或

```
if(a>=0)
    a=a;    //该语句的执行过程是什么？让计算机做这个操作有没有意义？
else
    a=-a;
```

4. if 语句可以没有 else 分支

当条件不成立不需要做任何事情时，可以省略 else 分支，例如，例 5.4 中右侧的代码就可以简化为

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



编程经验：任何一个条件其实都有两种写法，例如，若条件能写成 $a > b$ ，那么就一定可以写成 $a \leq b$ ；如果可以写成 $a = b$ ，肯定也可以写成 $a \neq b$ ，当条件反过来写时，if 和 else 的子句需要互换，如例 5.4 中的代码所示。两种写法中，一般会有一种写法能使后面的代码更简单、更容易实现。编程过程中当需要写条件时，通常要先在脑子里把两种写法比较一下，选择使代码简单的那一种。



注意：像上面这种代码，省略 else 分支不影响程序的运行结果，就一定要省略。那些无意义的代码即便自己愿意写，别人看起来也会觉得累。

5.3.3 嵌套的 if 语句

if 语句中，在 if 子句或 else 子句的位置上，都可以再使用一条 if 语句，使得 if 语句嵌套。一条 if 语句最多只能处理两种情况，嵌套的 if 语句可用来处理 3 种以上的情况。

一般来说，若有 3 种情况，则需要 2 条 if 语句嵌套；若有 4 种情况，则需要 3 条 if 语句嵌套；……；若有 n 种情况，则需用 $n-1$ 条 if 语句嵌套。if 的嵌套可以有多层。

常见的嵌套方式有如下 4 种：

<pre>if(表达式 1) if(表达式 2) ... [else ...]</pre>	<pre>if(表达式 1) if(表达式 2) ... else ... [else ...]</pre>	<pre>if(表达式 1) if(表达式 2) ... else ... else if(表达式 3) ... [else ...]</pre>	<pre>if(表达式 1) ... else if(表达式 2) ... [else if(表达式 3) ... [else ...]]</pre>
---	--	---	---

说明：嵌套的 if 语句的书写风格，应该是 else 跟与它配对的那个 if 对齐，各个子句也应该跟与它同级别的其他子句对齐，以使程序结构清晰，易于理解。



编程经验：编程时，通常都是使用最后一种嵌套方式，即第一个 if 语句只处理一种情况，把剩下的情况都放在 else 后面处理。用这种方式的好处是可以避免 if 和 else 配对错误。

下面的程序段用来求解 y 的值：当 $x > 0$ 时， $y = 1$ ；当 $x = 0$ 时， $y = 0$ ；当 $x < 0$ 时， $y = -1$ 。程序中的 if 语句原本应该是右侧的代码，因为其中一个 else 分支什么都不做，所以被省略了，但由于使用的嵌套方式不当，当程序省略 else 分支时，运行结果出现了错误。

```
#include <stdio.h>
int main()
{
    int x, y = 0;
    scanf("%d", &x);
```



嵌套的 if 语句及编程的常见问题



```

if(x>=0)
    if(x>0)
        y=1;
    else
        y=-1;
printf("%d\n", y);
return 0;
}

```

省略 else 分支

```

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

```

运行程序,当输入 0 时,结果如下:

```

0
-1

```

这段代码中的 else 看起来是跟第一个 if 配对,实际上它是跟第二个 if 配对。因为 C 语言规定: else 总是与它前面最近的、尚未配对的 if 配对。

 **说明:** else 和 if 的配对关系与 else 跟哪个 if 对齐无关,即不要以为 else 跟哪个 if 对齐它就跟哪个 if 配对,缩进只是为了方便阅读,编译时所有缩进的空白都被忽略。

在编译器看来,上面的 if 语句相当于是这样写的:

```

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

```

若采用最后一种嵌套方式,让第一个 if 只处理其中一种情况,把剩下的情况都放在 else 后面处理,就不会出现配对错误的问题。

```

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

```

 **说明:** 这段代码的最后也省略了 else 分支。

例 5.5 判断一个学生成绩属于哪个分数段。程序代码如下:

```

#include <stdio.h>
int main()
{
    int x;
    scanf("%d", &x);
    if(x>=90)
        printf("优");

```



```
else
    if(x>=80)
        printf("良");
    else
        if(x>=70)
            printf("中");
        else
            if(x>=60)
                printf("及格");
            else
                printf("差");
return 0;
}
```

本例用的也是前面推荐的嵌套方式。

 **想一想：**为什么代码中“if(x>=80) printf("良");”中的条件是“x>=80”而不是“x>=80&& x<90”? 若运行时输入的成绩是 95, 运行结果会不会是“良”或“优良”或“优良中及格差”?

 **提示：**在 if(x>=80) 前面有个 else。

 **编程经验：**尽量简化条件, 能省则省, 可以缩短表达式的求解时间。

有些编程者喜欢把上面的代码写成下面的格式, 但写成下面的格式后, 程序在逻辑关系上不如上面的清晰, 故不推荐使用。

```
#include <stdio.h>
int main()
{
    int x;
    scanf("%d", &x);
    if(x>=90)
        printf("优");
    else if(x>=80)
        printf("良");
    else if(x>=70)
        printf("中");
    else if(x>=60)
        printf("及格");
    else
        printf("差");
    return 0;
}
```

5.3.4 if 语句应用举例

例 5.6 从键盘输入 4 个整数, 找出其中的最大值。