if 语句之后的表达式运算结果为 true 时，条件成立，于是紧接其后的语句便会被执行。此例中如果 usr_rsp 等于'N'，则 usr_more 会被设置为 false。反之，如果 urs_rsp 不等于'N'，那就什么事也不会发生。不等运算符（inequality operator）的逻辑恰恰相反，例如：

```
if ( usr_rsp != 'Y' )
     usr_more = false;
```

麻烦的是，程序只检验 usr_rsp 的值是否为'N'，而用户却可能输入小写的'n'。因此我们必须能够辨识两者。解决方法之一就是加上 else 子句：

```
if ( usr_rsp == 'N' )
     usr_more = false;
else
if ( usr_rsp == 'n' )
     usr_more = false;
```

如果 usr_rsp 的值为'N'，usr_more 将被设置为 false，并结束整个 if 语句。如果其值不等于'N'，那么便开始对接下来的 else 子句求值。换句话说，当 usr_rsp 等于'n'时，usr_more 也会被设置成 false。如果两个条件都不符合，则 usr_more 不会被赋值。

新手常犯的错误便是将赋值（assignment）运算符误当作相等（equality）使用，例如：

```
// 呃，这会造成将常量字符'N'赋值给 usr_rsp
// 而整个表达式的运算结果为 true
if ( usr_rsp = 'N' )
     // ...
```

OR 逻辑运算符（||）提供了上述问题的另一个解法。它让我们得以同时检验多个表达式的结果：

```
if ( usr_rsp == 'N' || usr_rsp == 'n' )
     usr_more = false;
```

只需左右两个表达式中的一个为 true，OR 逻辑运算符的求值结果便为 true。左侧表达式会先被求值，如果其值为 true，剩下的另一个表达式就不需要再被求值（此所谓短路求值法）。本例中，只有当 usr_rsp 不等于'N'时，才会再检验其值是否为'n'。

AND 逻辑运算符（&&）则在左右两个表达式的结果皆为 true 时，其求值结果方为 true。例如：

```
if ( password &&
     validate( password ) &&
     ( acct = retrieve_acct_info( password ) ))
       // 处理账户（account）相关事务
```

最上面的一条表达式会先被求值。其结果若为 false，则 AND 运算符的求值结果即为 false，其余表达式不会（不需要）再被求值。本例中，当密码已设定，而且有效之后，账户的相关信息才会被取出。