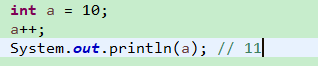
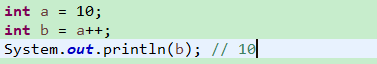
# 关于自增问题的深入解析

关于a++操作，我们都知道如果单独出现，其作用是将a的值加1，而参与运算的时候却是使用的a原来的值进行操作，然后将a的值进行加1。如下:



以上代码的输出为11.



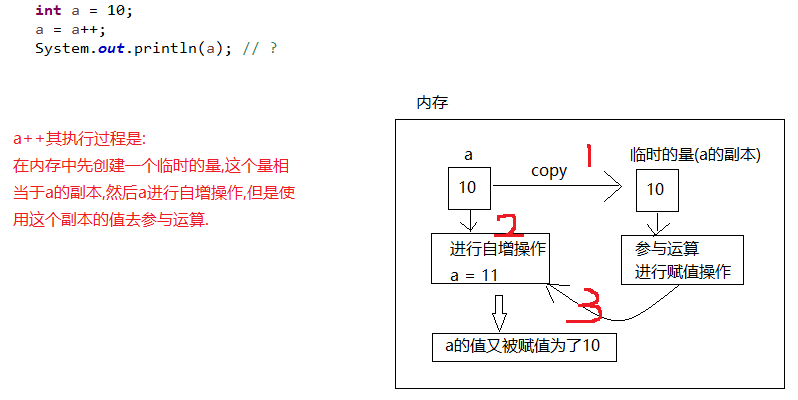
而以上代码输出结果却是10.

以上没什么问题,那么有意思的代码如下:



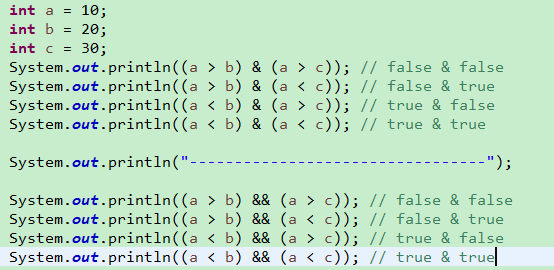
此时a的值到底是多少呢？ 通过运行我们发现，输出值为10。那么问题来了，不管它是先进行自增操作还是先进行赋值操作，a都应该增加1才对啊，那到底在这个运算里面++操作做了没有？

解答此问题，就需要解释一下a++操作在内存中的执行过程了。如下图：

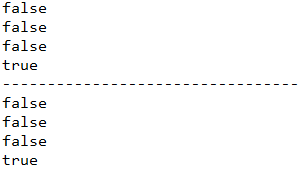


# 逻辑运算符中&和&&的区别

先观察如下代码:



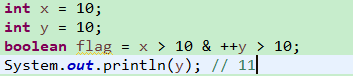
我们发现分隔线两边输出的结果是一样的:



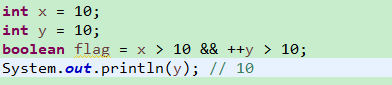
也就是说，不论&还是&&，其运算结果相同，都是对多个boolean表达式进行判断，如果都为true，结果为true；如果有一个表达式为false，其结果就是false。

那么，问题来了，如果判断左边的表达式已经为false了，那么整个表达式的结果其实就可以确定了，一定为false，此时右边的表达式无需再进行判断了，此时使用&和&&就产生了区别。

&&连接的表达式，会有一个短路的功能，如果发现左边的表达式为false，那么右边的表达式将不会被执行，而&却没有这个作用。我们用一段代码进行测试：



此时y值输出输出为11，因为不管左边的表达式是否为false，右边都要进行判断。



而以上代码的结果却是10，右边将不会再进行运算。

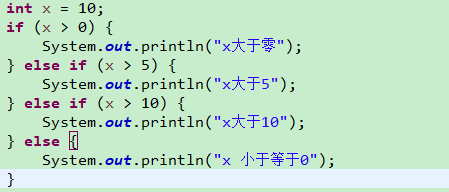
而短路||也是类似的操作，如果判断左边的表达式为true，结果就可以确定，右边的表达式将不再参与判断。

# if语句执行流程的注意事项

if语句的执行流程为：

自上往下，依次判断每个if表达式的值，发现为true，就执行该if语句，然后if语句结束。如果都不成立，执行else语句（如果有的话）。所以即使多个判断条件同时成立，也只会执行其中的第一个，所以，建议大家写代码时一定要尽量避免多个条件同时成立，以增强代码的可读性。

例如：



以上代码，其实只有第一行输出语句会执行。

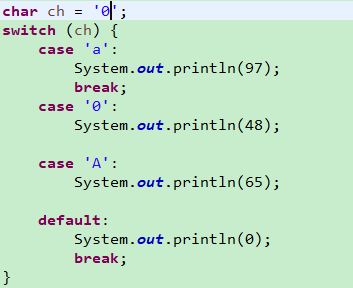
# switch语句的注意事项

1. 执行流程

从上往下，先挨个匹配每一个case，如果都不匹配才会执行default语句（不管default语句是否在case前面或者后面）

语句的结束条件是遇到break，或者执行到switch语句末尾。

1. 建议大家不要省略其中的break，可能会造成case穿透的现象



比如以上代码，输出结果为



并非我们所预期的，只有48。此时就产生了case穿透现象，匹配第二个case后，由于没有遇到break，也没有执行到switch语句的末尾，后面的case语句将不再判断而直接执行。

1. 建议大家也不要省略最终的default语句，对所有不匹配case的数据做一个默认的说明。

总结来说：请大家尽量不要省略和随意改动语句的标准格式，语法可能不会出问题，但可能会带来不必要的麻烦。