C语言 scanf的工作原理

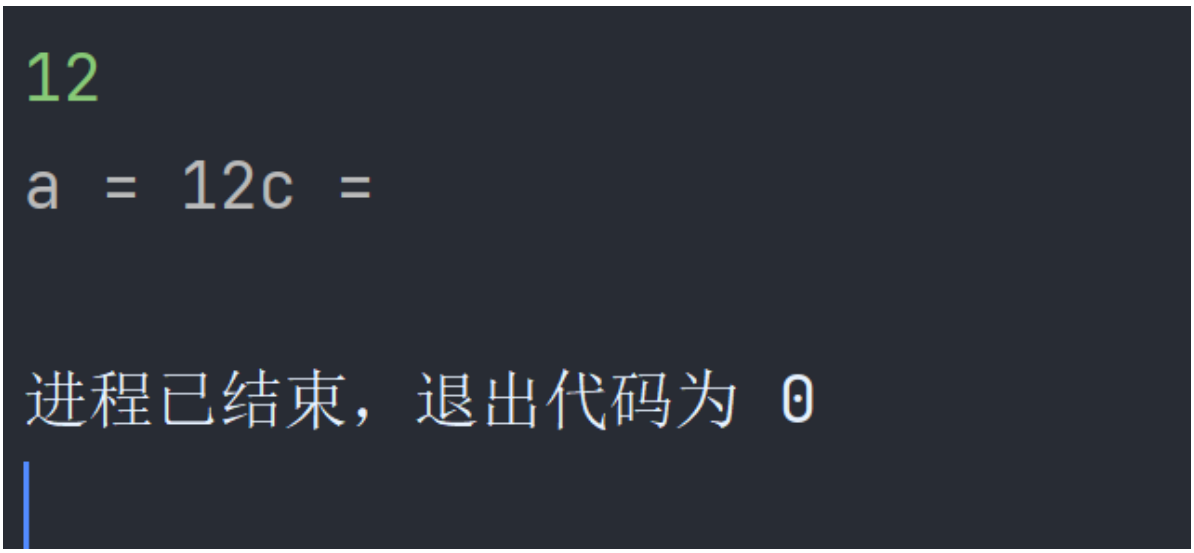
转载自https://blog.csdn.net/qq_43650934/article/details/122293681

原理解释

先来观察一段代码和运行结果：

```
#include <>

int main() {
    int a;
    char c;
    scanf("%d", &a);
    printf("a = %d", a);
    scanf("%c", &c);
    printf("c = %c", c);
}
```



这段代码明明有两个scanf，但是当输入12后按下enter键，代码直接停止，没有执行下一个scanf，why?

行缓冲：在这种情况下，当在输入和输出中遇到换行符时，将执行真正的IO操作。这时,我们输入的字符先存放到缓冲区中，等按下回车键换行时才进行实际的IO操作．典型代表是标准输入缓冲区（stdin）和标准输出缓冲区（stdout）。

如上面例子所示，我们向标准输人缓冲区中放入的字符为 ‘12\n’，输入’\n’（回车）后，scanf 函数才开始匹配，scanf 函数中的%d 匹配整型数 20，然后放入变量 i 中，接着进行打印输出，这时 ‘\n’ 仍然在标准输入缓冲区（stdin）内，如果第二个 scanf 函数为 scanf("%d",&i)，那么依然会发生阻塞

因为 scanf 函数在读取整型数、浮点数、字符串时，会忽略 '\n’（回车符）、空格符等字符（忽略是指scanf函数执行时会首先删除这些字符，然后再阻塞）

scanf 函数匹配一个字符时，会在缓冲区删除对应的字符。因为在执行 scanf("%c",&c) 语句时，不会忽略任何字符，所以 scanf("%c",&c) 读取了还在缓冲区中残留的‘\n’。

如果 scanf 接收的是其他类型的数据，则会忽略这个‘\n’，继续运行下面的代码，再举一个例子：

```c
#include <stdio.h>
int main() {
    int a;
    int c;
    scanf("%d", &a);
    printf("a = %d", a);
    scanf("%d", &c);
    printf("c = %d", c);
}
```

```
12
a = 12

        20
c = 20
进程已结束，退出代码为 0
```

如以上代码，我输入了好多个空格，但根本不影响实际的运行结果，因为它们都被 scanf 在缓冲区内忽略并删除掉了，scanf 不会随意忽略并删除缓冲区内的内容，除了上面列举的第一种情况，绝大多数情况下它会忽略空格和换行符

```c
#include <stdio.h>
#define EOF (-1)

int main() {
    int i;
    while (scanf("%d", &i) != EOF) {
        printf("i=%d\n", i);
    }
}
```

```
"A:\C learn\cmake-build-debug\C_learn.exe"
10
i=10
20
i=20
a
i=20
i=20
```

以上的 scanf 输入，是 10，20，a 的顺序输入，在输入 a 之后，代码一直打印上一个 printf 的内容，这是因为： scanf 返回的是成功读入的数据项数，在我的输入中输入了一个 a ， a 是无法匹配 %d 的，scanf 也不会删除 a ，所以 scanf 的返回值是 0（没有成功匹配），不等于 -1 ，此时就会一直 while 循环。

并且，在 scanf 返回值为 0 的情况下，没有读取 i 的值，此时 i 的值还是上一次输入的 20，这就会导致 while 循环一直打印上一次的 i=20。

解决办法

rewind(stdin) 是把文件指针回绕到文件起始处，stdin 是标准输入设备(输入流)

```c
#include <stdio.h>
#define EOF (-1)

int main() {
    int i;
    // scanf("%c", &c);  // 该方法也可以，指的是用一个字符类型数据读取缓冲区的 \n
    while (rewind(stdin), scanf("%d", &i) != EOF) {
        printf("i=%d\n", i);
    }
}
```

```
10
i=10
20
i=20
a
i=20
10
i=10
20
i=20
```

```c
#include <stdio.h>
#define EOF (-1)

int main() {
    int i;

    while (scanf("%d", &i) != EOF) {
        scanf("%c", &c);
        printf("i=%d\n", i);
    }
}
```