

关于 '\r' 的那些事

如果对本内容有相关疑问，欢迎联系 1706zcy@buaa.edu.cn

0. 开头语

这是一个十分令人头疼的问题，头疼你我他，即便是老师和助教，一个不小心也不可避免地会犯下这样的错误。

同学们在做题的时候，一般在 windows 系统下本地测试，助教们也普遍在 windows 系统下进行数据的生成。虽然OJ一般来说会自动去掉行末的空白字符再进行评测，但是有的时候也同样会出现问题。

然后导致的问题就是，同学们要无缘无故地为这样的错误买单，部分的助教和老师也会认为这是一个大家本就该考虑到的问题。这也会导致双方无法在这个问题上达成共识，在很多题目上助教和老师一个给不出解决的方法。以 Mid-2021spr-两航-C期中考试 的 F 题为例，很多使用 gets 的同学都只拿到了 0.04 分。而本题目测也是从2018级的两航C语言的考试题搬运过来的，这说明这个问题其实从很久以前就有了，但是每一年都要因为这个事情闹一次争执一次，一直得不到有效的解决方法，故笔者打算写一篇说明，针对这个问题进行一下科普。

0.1 关于 '\r' 和 '\n' 的来历

以下内容摘自该博客：<https://www.cnblogs.com/the-tops/p/5626828.html>

在计算机还没有出现之前，有一种叫做电传打字机（Teletype Model 33）的玩意，每秒钟可以打10个字符。但是它有一个问题，就是打完一行换行的时候，要用去0.2秒，正好可以打两个字符。要是在这0.2秒里面，又有新的字符传过来，那么这个字符将丢失。

于是，研制人员想了个办法解决这个问题，就是在每行后面加两个表示结束的字符。一个叫做“回车”，告诉打字机把打印头定位在左边界；另一个叫做“换行”，告诉打字机把纸向下移一行。

这就是“换行”和“回车”的来历，从它们的英语名字上也可以看出一二。

后来，计算机发明了，这两个概念也就被搬到了计算机上。那时，存储器很贵，一些科学家认为在每行结尾加两个字符太浪费了，加一个就可以。于是，就出现了分歧。

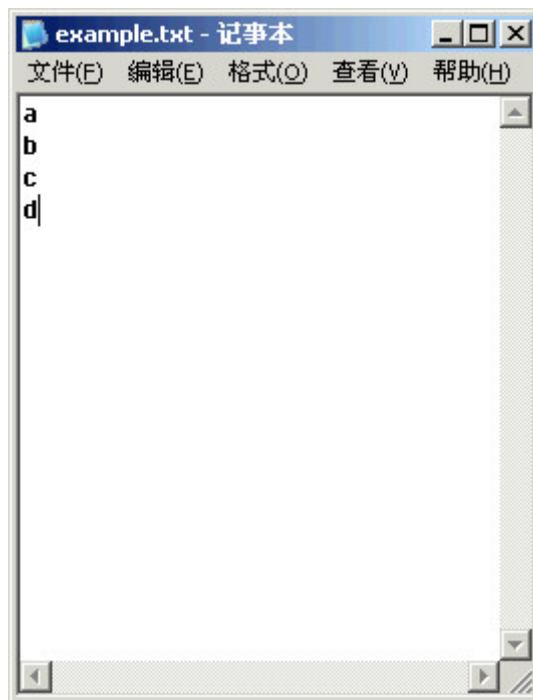
Unix系统里，每行结尾只有“<换行>”，即“\n”；Windows系统里面，每行结尾是“<换行><回车>”，即“\r\n”；Mac系统里，每行结尾是“<回车>”。一个直接后果是，Unix/Mac系统下的文件在Windows里打开的话，所有文字会变成一行；而Windows里的文件在Unix/Mac下打开的话，在每行的结尾可能会多出一个^M符号。

c语言编程时（windows系统）\r 就是return 回到 本行 行首 这就会把这一行以前的输出 覆盖掉

以下内容摘自该博客：<https://blog.csdn.net/lgouc/article/details/7815523>

记得在Windows下学X86汇编语言时，用0DH(\r)和0AH(\n)来输出回车(跳到下一行的开始处)。问题来了，在Windows下是先回车再换行呢还是先换行再回车呢？在Unix系统下换行只有\n，MAC OS下只有\r(网上是这么说的，没用过Mac OS，无从证实)，都不会出现上述的问题。

现在新建一个文本文档，其内容如下：



现在用C语言二进制形式将其读入字符串并按十进制输出。结果如下：

```
97 13 10 98 13 10 99 13 10 100
```

可以看出回车是 13 和 10 也就是 `'\r'` 和 `'\n'`，即先回车后换行。

一言以蔽之，有如下的区别

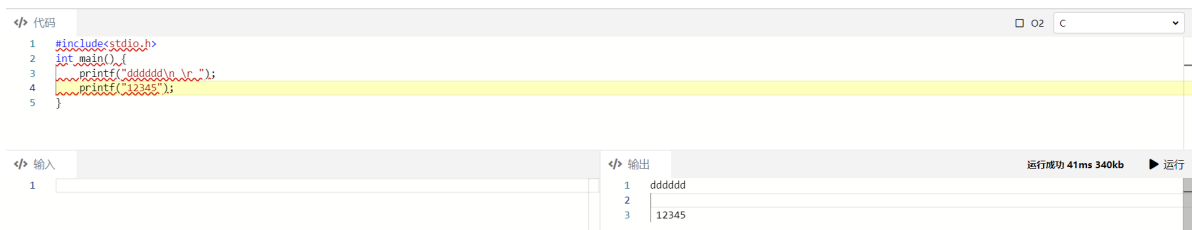
- 回车符 `\r` 和换行符 `\n`，是 2 个符。一个回车，一个换行。`\r`仅仅是回车，`\n`是换行。一个是控制屏幕或者从键盘的Enter键输入。另一个是控制“打印机”！
- 回车 = 光标到达最左侧，换行 = 移到下一行。如果只回车，打印的东西会覆盖同行以前的内容，如果只换行，打印的东西会在下一行的先一个位置继续。
- `'\r'` 实际是回到行首。`'\n'` 如果下一行已经有了一些内容的话它会在那些内容的后边.因为一般情况下下一行是没有数据的,很多时候 `'\n'` 也就成了 `"\r\n"` 作用一样。

0.2 在Linux系统下 `'\r'` 的具体表现形式

输出 `'\r'`

我们用不同的程序来演示一下

```
#include<stdio.h>
int main() {
    printf("dddddd\n \r ");
    printf("12345");
}
```



可以看到其输出为

dddddd

12345

输入 '\r'

我们以某一个题目的样例输入为例

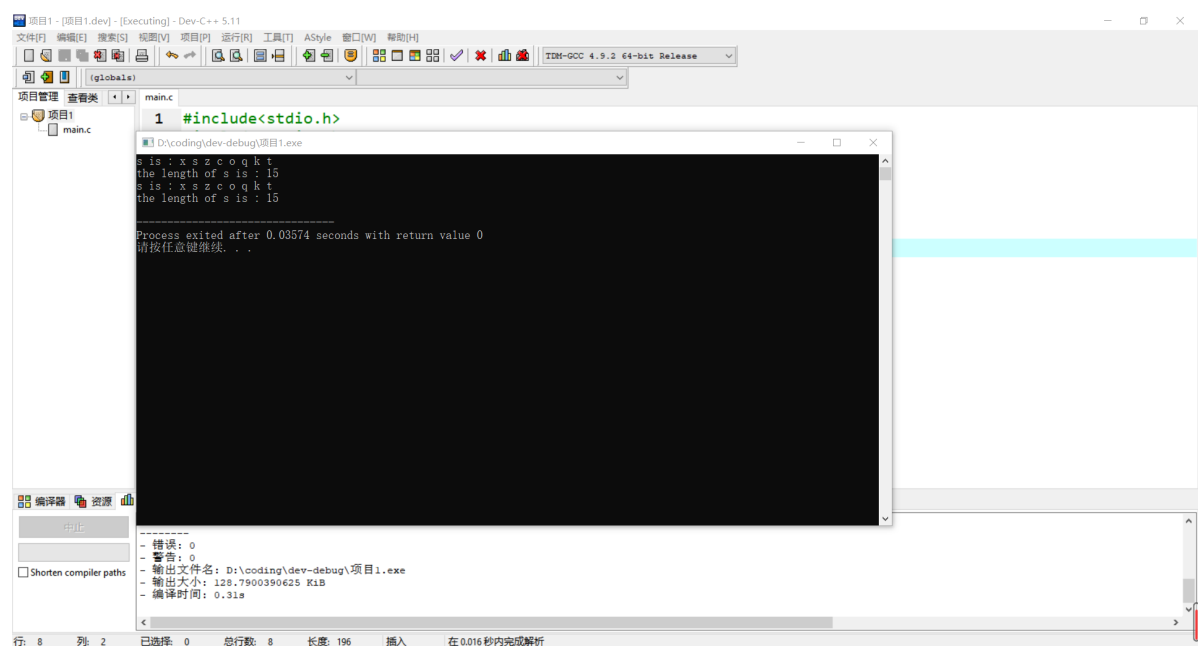
```
x s z c o q k t
x s z c o q k t
```

可以看到，这两行的长度是**完全一致**的，且行末都没有空格。

但是如果该数据是在 Windows 系统下，写程序，通过文件重定向生成的，以最后一行作为文件的结尾，我们采用以下程序来查看每一个字符串的长度。

```
#include<stdio.h>
#include<string.h>
char s[1100];
int main() {
    while (gets(s))
        printf("s is : %s\nthe length of s is : %d\n", s, strlen(s));
}
```

以下 windows 10 系统下的 devcpp 查看到的输出:



```
s is : x s z c o q k t
the length of s is : 15
s is : x s z c o q k t
the length of s is : 15
```

而在以 Linux 环境下部署的在线IDE当中所看到的输出却是这样的:

```
</> 代码
1 #include<stdio.h>
2 #include<string.h>
3 char s[1100];
4 int main() {
5     while (gets(s))
6         printf("s is : %s\nthe length of s is : %d\n", s, strlen(s));
7 }
8

</> 输入
1 x s z c o q k t
2 x s z c o q k t

</> 输出
1 s is : x s z c o q k t
2 the length of s is : 16
3 s is : x s z c o q k t
4 the length of s is : 15
5

运行成功 41ms 368kb ▶ 运行
```

```
s is : x s z c o q k t
the length of s is : 16
s is : x s z c o q k t
the length of s is : 15
```

看到差别了吗？第一行和第二行的长度，在 Linux 下竟然是不同的！

说白了，这就是第一行的 '\r' 在作祟

（而第二行的末尾没有换行，直接就是 EOF 了，EOF 也不会计算在字符串内）

0.3 是否有必要"声讨"OJ？提议OJ更换部署环境？

事实上，完全没有必要。而且无论是作为学生还是助教，也无权要求OJ更换部署环境。

要知道，OJ的部署环境各异，但是无论是 Windows 还是 Linux，都是业界经常使用的环境之一。倒不如说，现在的各大权威性的OJ，基本上采用的都是 Linux 环境

以高校内使用的OJ为例，北航OJ自不必多说，采用的是64位的 Linux 系统

清华大学校内授课的OJ <https://dsa.cs.tsinghua.edu.cn/oj/index.shtml>

用于进行程序设计、数据结构、离散数学、计算几何等课程的教学

采用的是 64 位 Linux 系统 Ubuntu 18.04

其用于举办各种比赛的TUOJ（thusaac）也基本是要求在 Ubuntu 系统下进行编写与提交。

以信息竞赛相关的一些权威OJ为例

LOJ <https://loj.ac/>

采用的评测机内核为：Linux 5.4.106-1-pve

洛谷 <https://www.luogu.com.cn/>

评测C/C++时的环境如下：gcc version 8.3.0 (Debian 8.3.0-6.1)

值得注意的是 Debian 同样是一个 Linux 系统。

NOI全国青少年信息学奥林匹克竞赛

其推出的 NOI Linux 就是专门为信息学奥林匹克竞赛选手设计的操作系统，是NOI系列赛事指定的操作系统。

1. '\r' 的解决方法——致同学

作为同学，日后如果依旧要写代码，无论是做什么工作，都会不可避免地 and '\r' 打交道。为此，也有必要了解处理 '\r' 的办法。

一般来说，直接使用 `scanf` 函数的话，除了采用 `%c` 读取单个字符之外，都会直接略过所有的空白字符（显然，`'\r'` 也在此列），基本不需要处理这样的问题。

但是如果采用 `gets` 或者 `fgets` 函数，则需要考虑行末 `\r` 的问题。

而鉴于 `gets` 函数已经在现有的 C/C++ 标准当中被废除（只有 `devcpp` 等老旧的 IDE 才可以使用），所以推荐大家在日后写代码的时候，采用 `fgets` 函数。

采用 `gets` 函数的解决方法

```
#include<stdio.h>
#include<string.h>
char s[110];
int len;
int main() {
    gets(s);
    len = strlen(s);
    while (s[len - 1] == '\r') s[--len] = '\0';
    //通过对len前置自减，可以保证字符串s去掉\r之后，len依旧等于strlen(s)
}
```

采用 `fgets` 函数的解决方法

```
#include<stdio.h>
#include<string.h>
char s[110];
int len;
int main() {
    fgets(s, 110, stdin);
    len = strlen(s);
    while (s[len - 1] == '\n' || s[len - 1] == '\r') s[--len] = '\0';
    //通过对len前置自减，可以保证字符串s去掉\r和\n之后，len依旧等于strlen(s)
}
```

C++ 语言采用 `getline` 函数的解决方法

```
#include<iostream>
#include<string>
using namespace std;
string in;
int main() {
    getline(cin, in);
    while (in.back() == '\r') in.pop_back();
}
```

2. `'\r'` 的解决方法——致出题的助教和老师

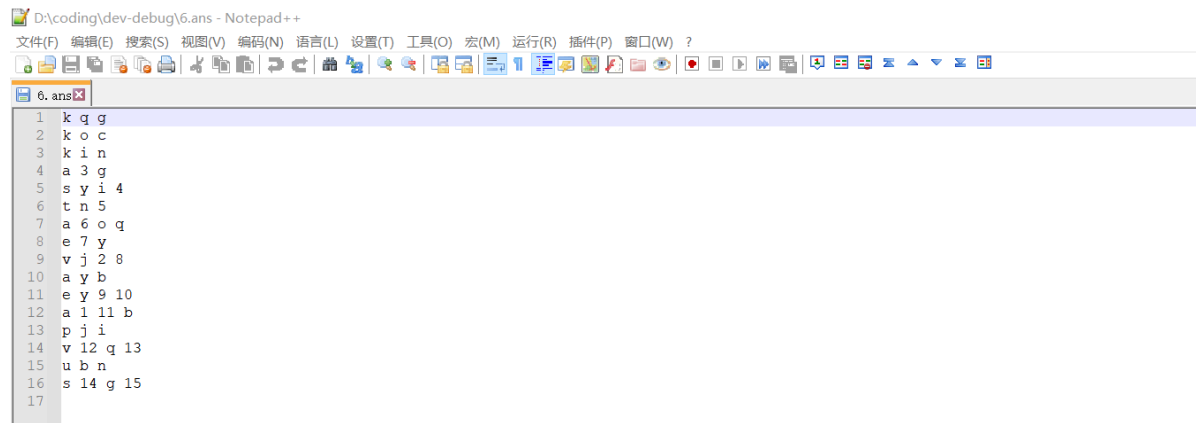
作为助教和老师，我们要考虑到，对于绝大部分使用 `windows` 和 `MaxOS` 做题的学生，使用 `Linux` 环境进行评测，在一门只是在介绍 C 语言程序设计的课程当中，强迫他们自发地考虑到跨平台带来的问题，这毫无疑问对他们是不公平的。

这样的问题，显然不可以让他们来平白无故地为之买单。

那么在出题的时候，如果在制造数据的时候没有去掉 `'\r'`，至少将上述的方法普及给大家。

但是如果出某道题目的重点**不在于字符串处理**，那么这个问题完全可以由出数据的助教和老师们自行解决。

对于 windows 环境，我们可以采用 notepad++ 或者 Subline Text 等文本编辑器。以下以 notepad++ 为例。



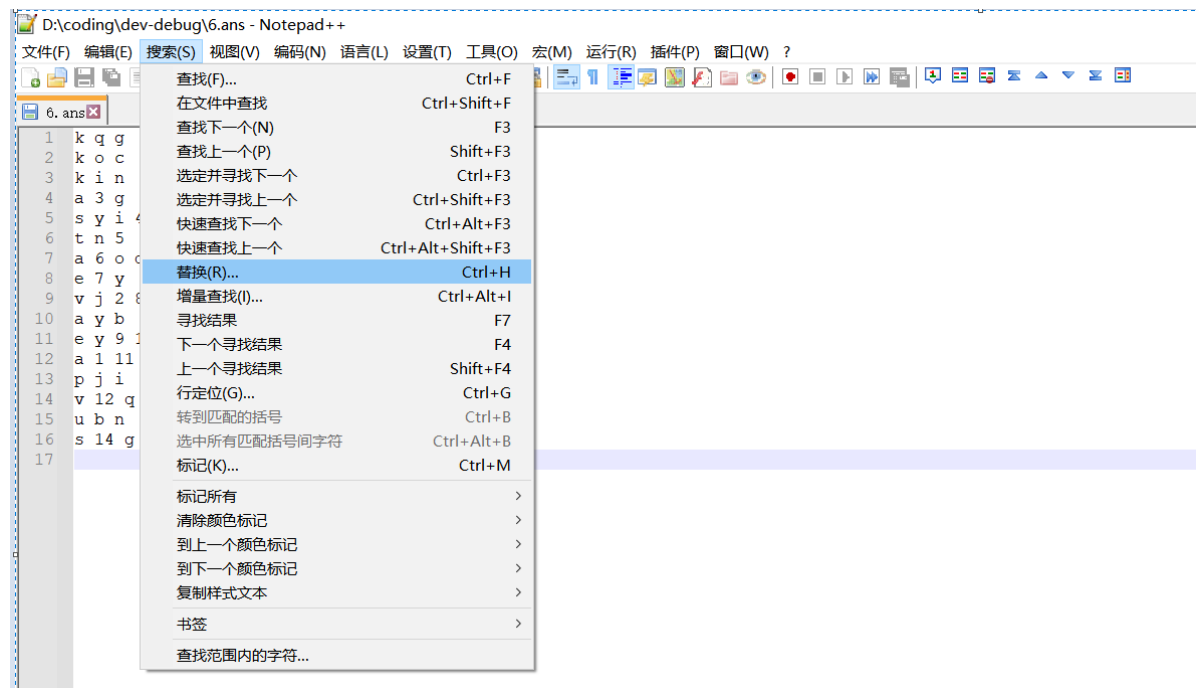
D:\coding\dev-debug\6.ans - Notepad++

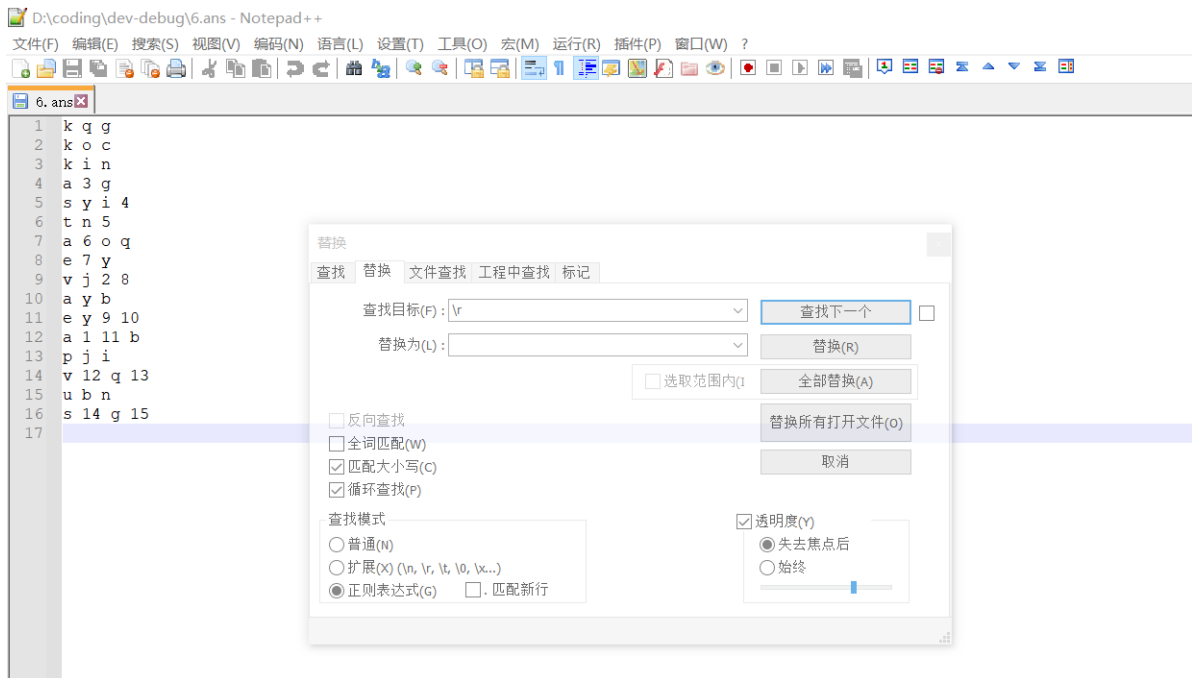
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(M) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?

6.ans

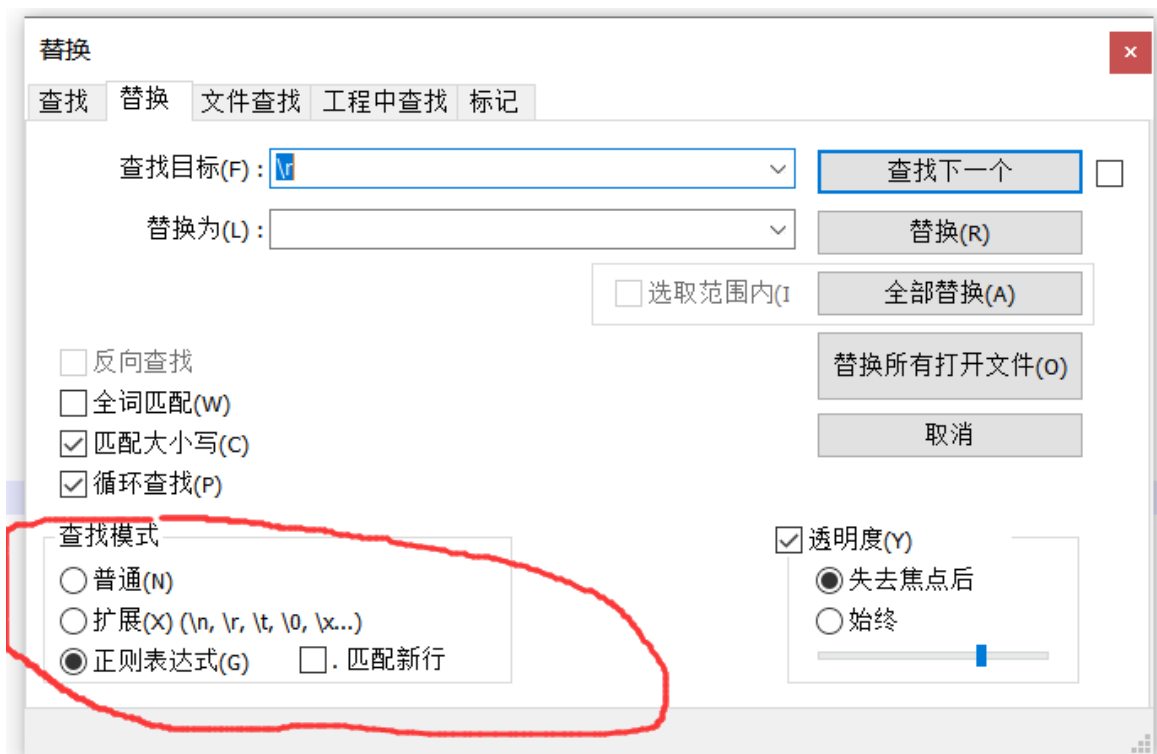
```
1 k q g
2 k o c
3 k i n
4 a 3 g
5 s y i 4
6 t n 5
7 a 6 o q
8 e 7 y
9 v j 2 8
10 a y b
11 e y 9 10
12 a l 11 b
13 p j i
14 v 12 q 13
15 u b n
16 s 14 g 15
17
```

找到替换



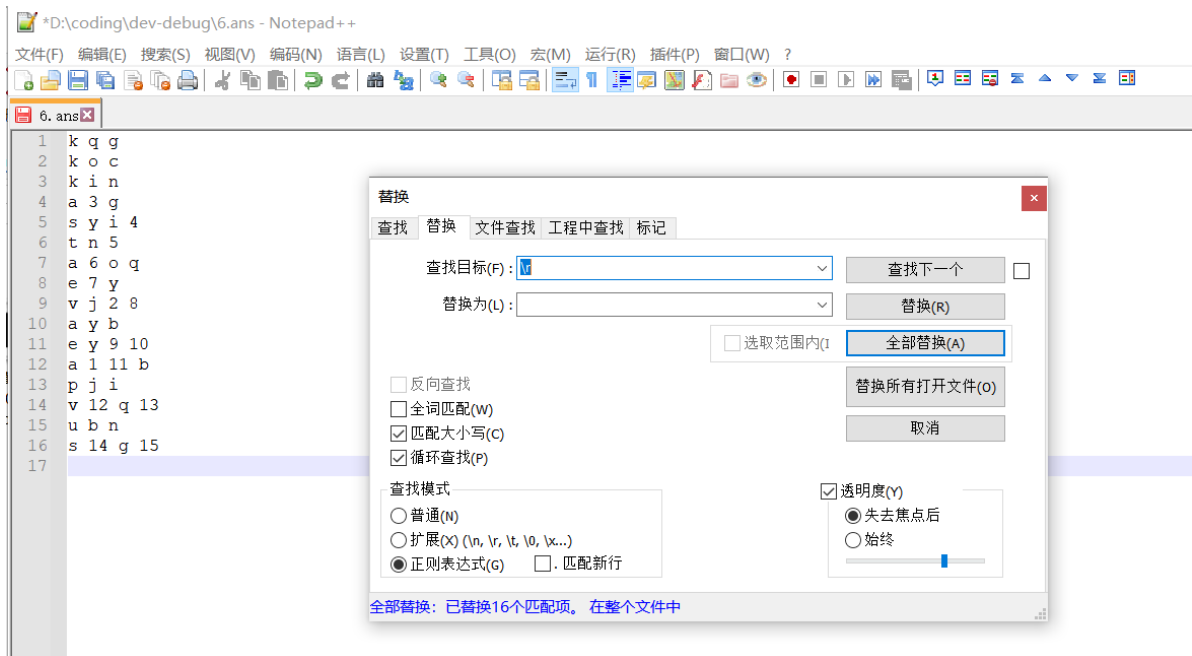


在“查找目标”一栏填写 \\r , “替换为”一栏什么都不填，空着



切记：在查找模式当中改为“正则表达式”！！

接下来，点击“全部替换”



这时我们看到，文件被修改，16 个 '\r' 已经全部被去掉，至此，数据的 '\r' 已经全部清除完毕！

3.节后语

没什么想说的，只希望这篇文章能够帮助大家，让同学们少一些疑惑，减少老师、助教与学生之间在这个问题上年复一年的纷争。