rbf

java层进行了格式的校验

```
if (!(input.length() == 22 && input.startsWith("miniLCTF{") && input.endsWith("}
    Toast.makeText(getApplicationContext(),"Wrong format"  , Toast.LENGTH_SHORT)
    return;
}
```

然后直接查看libmyrust的check方法,Java_com_doctor3_rbf_MainActivity_Check

```
while ( 2 )
{
  switch ( v20[v28] )
  {
    case '+':
      if ( v30 >= 0x400 )
        sub_61365(v30, 1024LL, &off_6B068);
      ++*(_BYTE *)(v27 + v30);
      goto LABEL_42;
    case ',':
      if ( v29 >= *((_QWORD *)&v26 + 1) )
        sub_61365(v29, *((_QWORD *)&v26 + 1), &off_6B110);
      if ( v30 >= 0x400 )
        sub_61365(v30, 1024LL, &off_6B128);
      *(_BYTE *)(v27 + v30) = *(_BYTE *)(v26 + v29++);
      goto LABEL_42;
    case '-':
      if ( v30 >= 0x400 )
        sub_61365(v30, 1024LL, &off_6B080);
      --*(_BYTE *)(v27 + v30);
      goto LABEL_42;
    case '.':
```

往下一翻可以看到很明显的vm结构

这个vm是brainfuck，当然如果没有办法调试得到code也可以根据hint来找

转到init_array

```
006D9F0 ; Segment permissions: Read/Write
006D9F0 _init_array     segment qword public 'DATA' use64
006D9F0                 assume cs:_init_array
006D9F0                 ;org 6D9F0h
006D9F0                 dq offset sub_23820
006D9F0 _init_array     ends
006D9F0
```

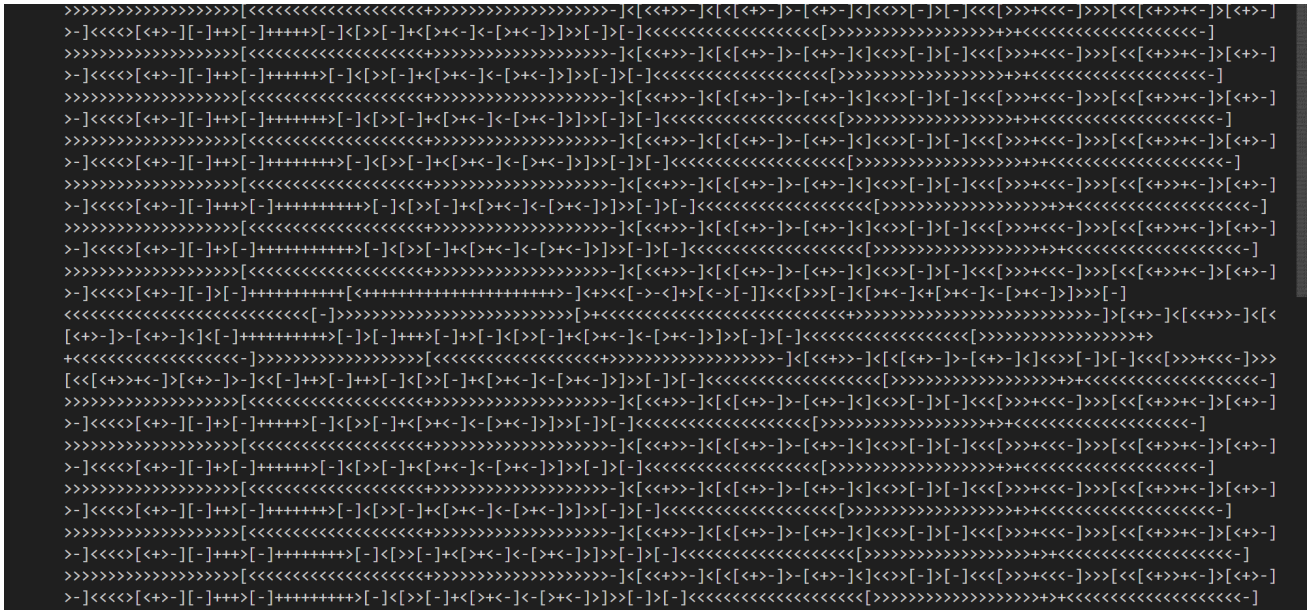
很明显的密钥流生成和异或，判断是rc4，key是myrust

```
31    do
32    {
33      if ( v8 == v9 )
34      {
35        v9 = v6;
36        v8 = a2;
37      }
38      v11 = *((_BYTE *)v13 + v7);
39      v10 += v11 + *v8;
40      *((_BYTE *)v13 + v7) = *((_BYTE *)v13 + v10);
41      ++v8;
42      *((_BYTE *)v13 + v10) = v11;
43      ++v7;
44    }
45    while ( v7 != 256 );
46  }
```

```
v0 = (_BYTE *)sub_243E0(0x7D1EuLL, 1uLL);
if ( !v0 )
  panic(1LL, 32030LL, (__int64)&off_6AEF8);
v1 = v0;
memcpy(v0, &unk_6238, 0x7D1EuLL);
sub_24430(dest, aMyrust, 6LL);
v11 = 0;
for ( i = 0LL; i != 32030; ++i )
  v1[i] ^= sub_245A0((__int64)dest);
```

得到bf代码之后可以直接自行实现一个bf解释器调试，没必要直接在原题中解答

观察bf代码，发现后半部分有大量重复代码出现



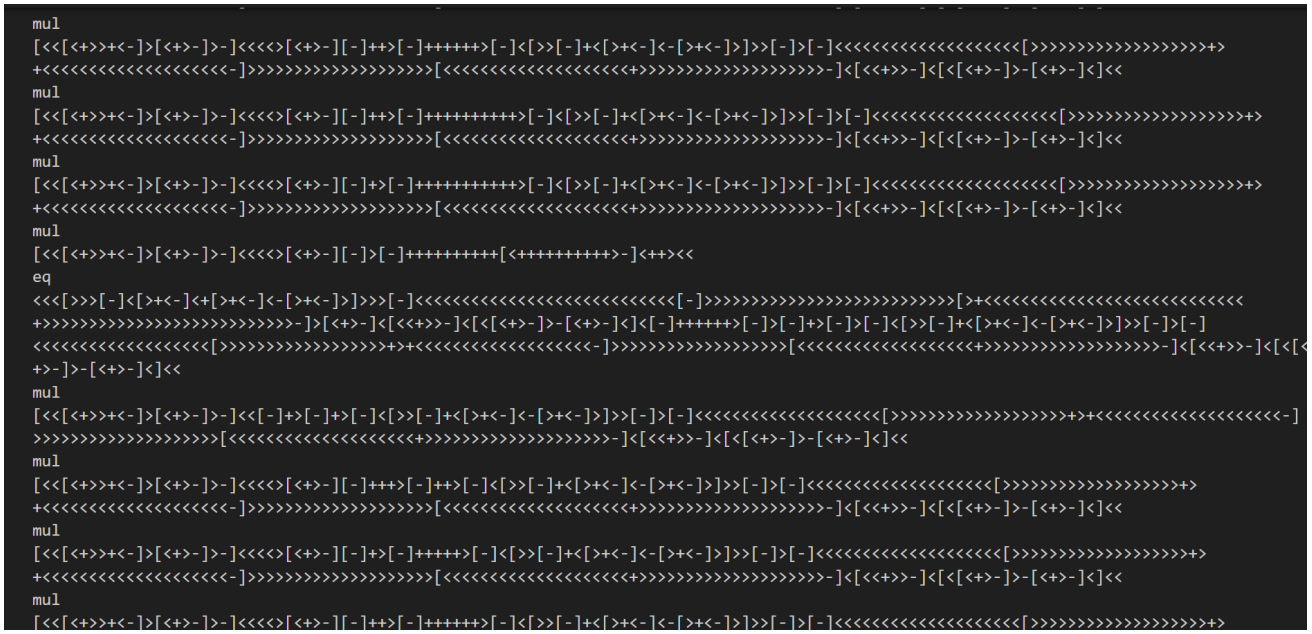
自行调试或者询问ai之后可以发现实际上是一组方程，考虑提取方程使用z3求解

经过分析得到eq和mul的op

```
eq: [->-<]+>[<->[-]]

mul: >>[-]>[-]<<<[>>>+<<<-]>>>
```

替换之后进行分割



再分析取数组位置以及比较，结合调试可以得出最后等式

```
from z3 import *

x = [Int(f'x{i}') for i in range(0, 12)]
```

```
solver = Solver()
solver.add(3 * x[0] + 3 * x[1] + 3 * x[2] + 3 * x[3] + 1 * x[4] + 1 * x[5] + 3 *
solver.add(3 * x[0] + 2 * x[1] + 2 * x[2] + 2 * x[3] + 2 * x[4] + 2 * x[6] + 1 *
solver.add(2 * x[0] + 3 * x[2] + 3 * x[5] + 3 * x[6] + 1 * x[7] + 1 * x[8] + 2 *
solver.add(2 * x[2] + 3 * x[6] + 2 * x[7] + 3 * x[8] + 2 * x[10] == 147)
solver.add(1 * x[0] + 2 * x[1] + 2 * x[2] + 1 * x[3] + 2 * x[5] + 1 * x[8] + 2 *
solver.add(3 * x[1] + 3 * x[5] + 2 * x[6] + 2 * x[10] + 1 * x[11] == 102)
solver.add(1 * x[0] + 1 * x[1] + 3 * x[2] + 1 * x[5] + 2 * x[6] + 2 * x[7] + 2 *
    11] == 192)
solver.add(3 * x[0] + 2 * x[1] + 2 * x[2] + 2 * x[4] + 1 * x[5] + 2 * x[8] + 1 *
solver.add(3 * x[0] + 3 * x[1] + 3 * x[2] + 3 * x[3] + 1 * x[4] + 2 * x[5] + 2 *
solver.add(1 * x[0] + 1 * x[1] + 2 * x[3] + 2 * x[4] + 2 * x[5] + 2 * x[6] + 2 *
    11] == 243)
solver.add(3 * x[1] + 2 * x[2] + 1 * x[5] + 1 * x[6] + 2 * x[7] + 3 * x[8] + 3 *
solver.add(3 * x[0] + 3 * x[1] + 3 * x[3] + 1 * x[4] + 2 * x[5] + 3 * x[6] + 2 *
    11] == 229)

if solver.check() == sat:
    ans = solver.model()
    for i in x:
        print(chr(ans[i].as_long() + ord('a')), end='')
else:
    raise RuntimeError("unsat")
```

miniLCTF{favyxwekppoa}