# 变量与历史记录

## 变量格式

### 变量在内存中的位置

我们知道，0xD180开始的0xC8个字节是输入区。紧接着，从0xD248开始的0xC8个字节为回放区。接下来，从0xD310开始有10个字节的随机数种子，0xD31A开始就是变量存储区。变量按照字符表的顺序连续存储，每个变量占10字节。比如，0xD31A～0xD324为变量M，0xD324～0xD32E为变量Ans等，以此类推。注意，字符表上一共有16个“变量”，但是其中只有前12个是真的变量（分别为M Ans A B C D E F 𝒙 𝒚 PreAns @），后面4个@占有的存储空间与即将讲到的历史记录相重合。

### 变量的储存格式

一个变量10字节的储存空间中，第一个字节的前面一半表示了后面九个半字节的类型。在这里我们只需要掌握最常用的浮点数类型即可。对于浮点数类型，这半个字节为0。从第一个字节的后面一半开始，有七个半字节，表示这个浮点数的15位有效数字，每半个字节表示一位，是按十进制直接存储的。最后还有两个字节。倒数第二个字节是指数，负指数用反码表示。最后一个字节的前一半固定为0，后一半表示这个浮点数整体和指数部分的正负。0表示整体正指数负，1表示整体正指数正，5表示整体负指数负，6表示整体负指数正。

以这个数为例：

06 89 47 57 00 00 00 00 00 01

第一个字节的前面一半告诉我们，这是一个浮点数。接下来七个半字节是有效数字：

6 89 47 57 00 00 00 00

第九个字节00表示指数。最后一个字节01表示这个浮点数整体是正的，指数也是正的。所以这个数就是+6.89475700000000E+00，也就是6.894757，大家对这个数一定很熟悉了。

再以这个数为例：

07 46 42 68 41 12 85 46 82 05

与刚才一样，有效数字为

7 46 42 68 41 12 85 46

不同的是，最后一个字节05表示整体是负的，指数也是负的。反码就是用100去减，82-100=-18，所以指数就是-18。这整个数是-7.46426841128546E-18。

### 合法变量与重置

在每次按开机的时候，计算器会检查变量是否合法。如果某个变量第一个字节的后半在0到9之间，而且最后一个字节以0开头，那么它就是合法的。开机时，计算器会挨个检查所有的10个变量，如果遇到了不合法的变量，就会进行完全初始化（初始化的等级与自检相同，都不保留语言设置）。比如，如果我们卡出字符3A并按下等于，此时变量Ans就会设为A，也就是

0A 00 00 00 00 00 00 00 00 01

按下开机，因为变量Ans的第一个字节的后半是A，大于9，不合法，所以就会重置。同样的，在位移较小的an模式中，你会发现有些变量的值是异常的，此时按开机，多半就会触发重置。

如果用q92手动清空存储器，只有前10个变量会清空。后面的PreAns、@，还有后面4个@，都不会受到影响。

## 历史记录格式

### 历史记录在内存中的位置

紧跟着12个变量就是历史记录区。因此历史记录区的起始与“变量”4C重合。

### 历史记录的储存格式

每条历史记录分为3个部分。第一部分长3字节，主要是这条历史记录的储存模式（例如是否有虚部等等）。第二部分长10或20字节，包含了1或2个数值，表示这条历史记录的“答案”。如果答案是虚数，那就有2个数值，第一个是实部，第二个是虚部。类似的，计算余数、Pol／Rec互化等，都会产生两个数值。如果答案是个一般的实数，那么数值就只有一个。第三部分是算式，长度不定。这一部分以字节23（对应的单字节字符为:）结尾。注意，如果在读到字节23之前先读到了字节00，那这条记录就是“错误”的，计算器也不会尝试读取下一条历史记录。

如果你还记得的话，在第一章我曾经提到，计算会导致字符FE23的23被吃掉。这是因为23被当成了这条历史记录的结尾。在后面再多放几个字符，它们就会被当成下一条历史记录。如果这些字符的长度不到13字节，那么后面这条历史记录就是无效的，但是如果字符长度超过13字节，超过的部分就会进入下一条历史记录当中。

## 变量和历史记录的重叠

### 通过变量修改历史记录

如前所述，“变量”的位置和历史记录相重叠，可以通过给这些变量赋值来修改历史记录。有一种广为人知的刷字符方法：刷出字符，然后打@=1.0000（编码）23:𝒙，其中编码是要刷的字符对应的粗体十六进制编码，要刷字符就打81，刷就打B3等等。最后rC，最上面一条历史记录就是想要卡的字符。

现在我们来探究这种方法的原理。假设我们经济困难，要刷四个字符。那么算式即为@=1.00002424242423:𝒙。一般来说，在刷字符之前，我们会用一下字符转换器，因此历史记录里已经有了一条：

xx xx xx:06 89 47 57 00 00 00.00 00 01:31 FE 23|23 ......

接下来的操作会给变量赋值。我们知道，变量与历史记录的开头对齐，那么变量就位于历史记录开始10字节以后，也就是加了点的那个位置。变量会覆盖掉原有的历史记录：

xx xx xx:06 89 47 57 00 00 00.01 00 00:24 24 24 24 23|00 01 ......

现在我们翻到最上面，计算器就会把第一条历史记录理解为，答案是

06 89 47 57 00 00 00 01 00 00

也就是6.89475700000001（由于精度问题，只会显示成6.894757），而算式是

24 24 24 24

也就是我们想要刷的四个。注意，最后一个23是必须要加的，用来分隔历史记录。

这种办法显然效率低下，一次只能刷4个字符，如果我需要更多的字符怎么办呢？回去看一下前面讲的浮点数格式，你会发现，浮点数的倒数第二个字节是指数，也可以利用一下。有一种显而易见的改进方法，就是把刚才打的1.00002424242423改为1.0000242424242423。这样就利用了指数位，可以多打一个字。

单个基本上只能做到这样了。但是为什么只用呢？可以把、、三个变量拼起来一块用。我曾经发过一个在虚数模式下刷38字的方法：

1. 每个浮点数可以放8字节的数据，格式为1.(前7字节) (最后一个字节)
2. 先按照顺序给变量赋值：D=1.0000(6字节)，M=1.(8字节)，F=1.(8字节)，𝒙=1.(8字节)+1.(8字节)𝒊
3. 切换模式以清空历史记录，刷出，r，把两个@都赋值为0，然后C!，改为这个式子：

111:((M)):𝒙:@=D:@=F

1. r=====，向上翻到第一条历史记录

（注：在线性复数an模式中刷字符的方法略有不同，见本章末）

这种方法的原理又是什么呢？首先为了刷和，我们肯定用到了字符转换器，因此现在的历史记录区就是：

xx xx xx:06 89 47 57 00 00 00 00 00 01:31 FE 23|23 ......

然后变量被设成0了，这覆盖了唯一的一条历史记录，使其失效，现在就没有历史记录了:

xx xx xx 06 89 47 57 00 00 00 \*\*00 00 00 00 00 00 00 00 00 00\*\* ......

接下来的111只是用来“垫”的，占了17个字节的空间：

xx xx xx:01 11 00 00 00 00 00 00 02 01:31 31 31 23|00 00 00 ......

((M))就是M，它把变量M中存储的数值放在历史记录里面：

xx xx xx:01 11 00 00 00 00 00 00 02 01:31 31 31 23|xx xx xx:01 mm mm mm mm mm mm mm mm 01:60 60 40 D0 D0 23| ......

又填充了𝒙：

xx xx xx:01 11 00 00 00 00 00 00 02 01:31 31 31 23|xx xx xx:01 mm mm mm mm mm mm mm mm 01:60 60 40 D0 D0 23|xx xx xx:01 xx xx xx xx xx xx xx xx 01 01 xx xx xx xx xx xx xx xx 01:48 23| ......

注意𝒙是虚数，因此虚部也包含在里面，答案区占了20个字节。然后我们给和赋值：

xx xx xx:01 11 00 00 00 00 00.01 00 00:dd dd dd dd dd dd 01.01 mm mm mm mm mm mm mm mm 01.01 ff ff ff ff ff ff ff ff 01.01 xx xx xx xx xx xx xx xx 01 01 xx xx xx xx xx xx xx xx 01 48 23| ......

此时你会发现，D、M、F、𝒙都连在一起了，而且都位于第一条历史记录的算式区。我们只要向上翻到第一条记录，就能看到要刷的字符，多达38个！

但是，这种方法还是有一些限制。第一，它只能在虚数模式用。（计算模式里可以去掉𝒙的虚部，可是那样就只能刷30个字符了。）第二，如果一个浮点数的最后几位有效数字中包含粗体十六进制字母，它可能会触发进位，导致刷字符失败。而且，指数位也不能包含粗体十六进制字母。（其实所有通过这个原理刷字符的方法都有这个缺点。）怎么办呢？

### 通过历史记录填充变量

我们发现，一个变量占了10字节，但是我们只用到了其中的8个字节。当然，这是因为我们把变量作为浮点数来赋值。有没有办法直接给变量“填充”值呢？当然也是有的。假设你要给一个变量填充10字节的值。首先你要做一个准备工作。刷出字符，=J[。这是一个白值，显示不出来，计算它的零次方会得到ERROR。这个ERROR和数学错误不一样，它会使计算器停止运算接下来的式子，不会报错。有了它，就能把任何东西放进历史记录了。

接下来，刷出你想要的10字节（可以通过连续赋值的办法），记得刷的时候一并刷一个。

翻到最上面一条历史记录，先直接r，把赋值为0，这样就清掉了第一条历史记录。然后输入如下算式：

++要刷的10字节

为什么要这样做呢？其实，数学输入中的次方符号长4个字节，因此你输入的式子在内存里长这样：

48 C9 1A 30 1B A6 A6 十字节......

按=，把它读入历史记录中，现在历史记录就是：

xx xx xx:F3 00 00 00 00 00 00.00 00 00:48 C9 1A 30 1B A6 A6.十字节...

你会发现那十个字节刚好和变量对齐了，因此现在变量就是你想要填充的值！刷出，然后Jz，就可以了。（有些时候，J会导致报错，那就只好老老实实用浮点数了，浪费的2个字节我们也拿它没办法。）同理，其实可以一次刷20个字节，然后用和刚才一模一样的步骤，同时填充和，效率直接翻倍。通过这种方法，在计算模式中理论上一次可以刷出37个字符，基本上是够用了。

### 附：线性复数an模式中刷字符的方法

1. 进入an之前，先按照顺序给变量赋值：D=1.0000(6字节)，M=1.(8字节)，F=1.(8字节)，𝒙=1.(8字节)+1.(8字节)𝒊
2. 进入an模式，刷出，输入这个式子：  
   111:((M)):𝒙:@=D:@=F
3. R=====，向上翻到第一条历史记录