DEBUG 和改向功能结合所得文件的有关问题

湖南省双峰工商银行 罗 辉

摘要：用 *DEBUG* 和 *DOS* 改向输入输出功能结合得到文件在各种字处理软件中编辑分析时，存在许多的麻烦，本文分析其原因并提供消除这一问题的方法。

许多报刊杂志上经常介绍一种分 COM、EXE 等文件的方法，即用 DEBUG 命令结合操作系统的改向输入(<)、输出(>)功能，得到被分析文件的二进制映象或反汇编程序到某一文本输出文件中，然后再通过各种字处理软件对该文件进行阅读编辑分析等操作。譬如如下命令：

```
C>debug command. com <input. tmp> output. tmp
```

其中输入文件 input.tmp 的内容(DEBUG 操作命令行)为：

```
d100,900
u100,900
q
```

那么上述命令就得到一个 output.tmp 文件，其内容是 command.com 文件的开始 800H 个字节的二进制映象及其反汇编程序。

但是，在用各种字处理软件比如 Wordstar、WPS 等对所得到的输出文件 output.tmp 调用编辑分析时，会出现一些暮有其妙的问题，譬如：在 WPS 中用 PhDn 键翻屏到文件尾后，再用"↑"键上移几行，然后欲用"↓"键下移几行，却不能如愿；或者用"→"或"END"键不能移到行尾；或者欲将一行折成两行时，下一行却无故消失不见，好象被删除了一般，重新调用该文件，却又恢复原状，用调试成功的自编程序对这类输出文件进行某种处理时，发现程序却不能正常终止，好象被处理文件无限长似的……等，给操作带来诸多不便。

用上述方法生成输出文件时，输入文件中的 DEBUG 操作命令行(如 input.tmp 中的"d100,900"、"u100,900-、"q-等)都会出现在输出文件中。通过用 DEBUG 对输出文件进行分析后发现其"作怪"的原因：

1.在输出文件中出现的 DEBUG 操作命令行的行尾，并没如理想的直接附加上行尾标志——回车换行码(ODH、OAG)，而是附加上了两个回车码加一个换行码(ODH、ODH、OAH)的行尾标志，多了一个(ODH)！而一般字处理软件的行尾处理都是直接对回车换行码(ODOAH)成对处理的，因而在处理(ODODOAH)时程序失控，从而出现莫明其妙的不便。

2.用这一方法生成的文本输出文件，文件结尾并没有文件结束标志(1AH)！文件是否结束，仅仅由文件目录表项的文件长度予以限制。这对于某些以文件结束符为文件结束判别标志的字处理软件或自编的文件处理程序来说，这类文件将没有结尾，无疑将达不到理想的目的。

面对这样的困境时，您不妨在 TURBO C2.0 环境下编译下面的 DBGO.C 程序，用如下的格式调用它：DBGMOD＜匹配文件名＞＜CR＞

它将匹配的文件中出现的不正常的(ODODOAH)行尾标志换为正常的(ODOAH)行尾标志，同时在文件尾附加上结束标志(1AH)。这样，用 DEBUG 命令结合改向功能得到的输出文件，再经过本程序处理后，将可以正常在各种字处理软件中使用和分析了。

```
C>type DBGMOD.C
#include "stdio.h"
#include "dir.h"
main(argc,argv)
int argc;
char * argv[];
{
struct ffblk f;
FILE * fp, * fp1;
```

```
    int ch, count = 0,error;
    long int sizecount, size;
    if (argc < 2) {
    printf("\n\t\tUsage: %s 匹配文件名\n", argv[0]);
    exit(0)
    }
    error=findfirst(argv[1],&f, 23);
    if (! error)
    printf("\n\t\t 处理本目录如下文件:\");
    slse
    printf("\n\t\t 没有匹配文件!\n"), exit(1);
    while (!error( {      / * 文件批处理 * /
    count++;
    size = f.ff__fsize;   / * 置被处理文件长度 * /
    fp1 = fopen(f.ff__name, "r+b");
    fp = fopen(f.ff__name, "r+b");
    if ((fp1 = = NULL)     (fp = = NULL))
    printf("\007\t%14s 文件打不开!\t\n",f.ff__name);
    else
    {
    printf("\n%d--%14slength              %dbytes",count,
f.ff__name, size);
    sizecount = 1
    while (sizecount < = sinze)      / * 一个文件尚未处理完
* /
    {
    ch = fgetc(fp1);
    sizecount++;
    fputc(ch,fp);
    if((ch = = 0x0d)&&(sizecount < = size))
    {   / * 如果出现"ODH"码,下面过滤多余的"ODH"码
* /
    ch = fgetc(fph);
    sizecount++;
    if ((ch! = 0x0d)&&(sizecount < = size))
    fputc (ch, fp); / * 如无多余的"ODH"码复写原文 * /
    }
    }
    ih)ch! = 0x1a) fputc (0x1a,fp; / * 加文件结束标志 * /
    fcolse (fp);
    fclose(fph);
    }
    error = findnext(&f);      / * 下一文件 * /
    }
    prinft{"\n\n\t\t\t----Finished!----!\n);
```

(上接第 43 页)

```
write (fp,al,ah):   {保存屏幕点的 ASCII 码值和属性}
end;
ah: = 2 dl: = col1;
dh: = row1; bh: = 0; intr( $ 10,reg)  {置光标回到起始点)
close (fp)
end;
end

program restore;  {屏幕信息恢复程序 RESTORE.PAS}
uses dos;
var
reg;registers;
fp:file of byte;
col, row, col1, row1, col2, row2:byte;
errcode:integer;
begin
if paramcount < > 1 then {命令行参数的合法性检查}
begin writeln('Paramte Error!", #7); exit; end;
assign(fp, paramstr(2)); reset(fp);
if IOResult < > 0 then exit;
read(fp, col1, row1, col2, row2);  {读屏幕坐标范围值}
with reg do
begin
for row: = row1 to row2 do
for col: = col1 to col2 do
begin ah: = 2 dl: = col; dh: = row; bh: = 0;
intr( $ 10, reg);   {循环置光标历经要保存的点}
ah: = 9; bh: = 0;read)fp,al, bl); cx: = 1;
intr( $ 10, reg);  {写各屏幕点的 ASCII 码和属性}
end;
ah: = 2; dl: = col1; dh: = row1; bh: = 0
intr( $ 10, reg);{置光标回到起始点}
close)fp);
end;
end.
```