A FORTH programmer will code FORTH in any language

Even SuperBasic.

FORTH is not a RPN calculator. It's a way to dialog with a computer through sentences composed of words he understands, and eventually teach him new words. If your FORTH understands the words **STEPS** , **RIGHT** and **LEFT**, you can tell him :

```
5 STEPS RIGHT 5 STEPS RIGHT 5 STEPS RIGHT 5 STEPS RIGHT
```

and he might stroll through a square five steps wide (like a disciplined soldier).

But you may also tell him :

```
: SIDE 5 STEPS RIGHT ;

: SQUARE 4 TIMES( SIDE ) ;

SQUARE
```

thus, having taught him two new words : **SIDE** and **SQUARE** . Simply teaching FORTH new words is coding. You start teaching him with word **:** and a new name, and end it with word **;** .

The only rule is that the words must be separated by spaces. Any word he does not understand, he will try to see it as a number and pass it to the next word that expects a parameter, like **STEPS**.

BASIC was designed as a simplified FORTRAN perusing its numeric labels by giving them to all statements by numbering them. So, you could use (and abuse) the keywords **GOTO** and **GOSUB** to freely jump here and there in sometimes monstruous “spaghetti programs”.

Sinclair SuperBasic introduced **PROCedure** and **FuNtion** constructs that you may simply call by their name like FORTH words. But still kept the line numbering to let you cook spaghetti code.

A FORTH programmer will like to use and peruse **PROCedure** and **FuNtion** constructs to add his new words to all the keywords of SuperBasic. He will consider that coding is not to type in a huge spaghetti program and tell him to run, but to name a word that will name words, etc. Like in FORTH, programming is to use SuperBasic to add to its existing potentialities.

As a FORTH programmer, writing a BOOT program for my QL, I wanted it to include two of my prefered FORTH words : **BYE** and **OK**. I thus coded this short generic procedure into my boot file :

```
32757 DEFine PROCedure BYE
32758 PAPER#3,0:INK#3,4:CSIZE#3,0,0:CLS#3:AT#3,0,7
32759 PRINT#3,' F1',,'F2',,'F3',,'F4',,'F5':RESTORE OK
32760 FOR i = 5 TO 69 STEP 16
32761    READ p,q,p$,Q$:PAPER#3,p:INK#3,q
32762    AT#3,1,i:PRINT#3,p$:AT#3,2,i:PRINT#3,Q$
32763 END FOR i
32764 p = CODE(INKEY$(-1))/4
32765 IF p<58 OR p>62 : GO TO 32764
32766 CLS#0:CLS:CLS#2:GO TO p+OK-53
32767 END DEFine BYE
```

Each time you name this procedure from within a SuperBasic console window or if this procedure is part of a statement in a SuperBasic program, it will display at the bottom of the screen a banner with five function key labels for **F1** to **F5**. This one is the **TOP_MENU** banner :

As you may note, to do this, **BYE** needs an initial value for the variable **OK(32702)** and a **MERGE** of this following little file with 5 **DATA** statements and 5 "target" statements :

```
32702 DATA   208,7,'   PSION   ','   SUITE   '
32703 DATA   208,7,'  PROLOG   ',' and FORTH '
32704 DATA   208,7,'    C68    ','programming'
32705 DATA     2,7,'   QJUMP   ','  Q R A M  '
32706 DATA    80,7,'   SUPER   ','   BASIC   '
32707 OK=32712 : PAPER#2,0:INK#2,4:CLS#2 : bye
32708 OK=32722 : PAPER#2,0:INK#2,4:CLS#2 : bye
32709 OK=32732 : PAPER#2,0:INK#2,4:CLS#2 : C68 : bye
32710 EXEC DEV$ & 'QRAM' : bye
32711 CLS:CLS#0:PRINT#0,'Say BYE to quit SUPER BASIC':STOP
```

The 5 **DATA** statements tell you that any **PAPER** and **INK** colors can be adopted for displaying the function key labels. I chose to paint "purple" access to submenus, "red" executable targets and "lined red" for going back to father menus. **TOP_MENU**'s father is the SuperBasic console, in which you can pass any commands, including **BYE** to go back to the function keys management. The target statement of this option (**F5**) ends with **STOP**. All the other target statements finish recalling **BYE**.

The statements 32707 to 32709 are targets for three submenus ; they simply update **OK** before calling **BYE**. Here are the banner and menu file for the PSION suite :

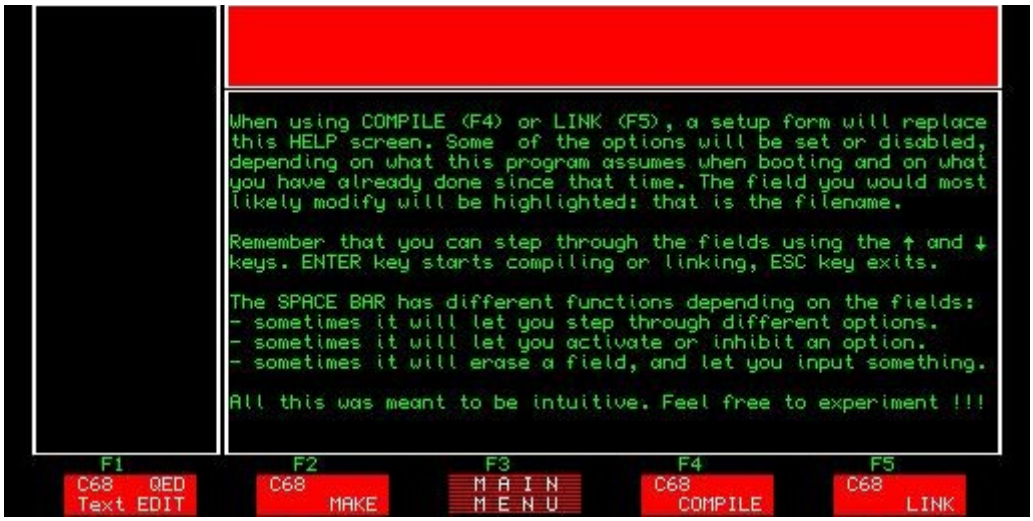

```
32712 DATA  80,7,'  M A I N  ','  M E N U  '
32713 DATA   2,7,'   PSION   ','   QUILL   '
32714 DATA   2,7,'   PSION   ','  ABACUS   '
32715 DATA   2,7,'   PSION   ','  ARCHIVE  '
32716 DATA   2,7,'   PSION   ','   EASEL   '
32717 OK=32702 : bye : REM back to main menu
32718 EXEC DEV$ & 'quill'      : bye
32719 EXEC DEV$ & 'abacus'     : bye
32720 EXEC DEV$ & 'archive'    : bye
32721 EXEC DEV$ & 'easel'      : bye
```

From lines 32718 to 32721 the targets will **EXEC** the four PSION programs before recalling **BYE**. Line 32717 will simply update the **OK** variable to let **BYE** return to the main menu.

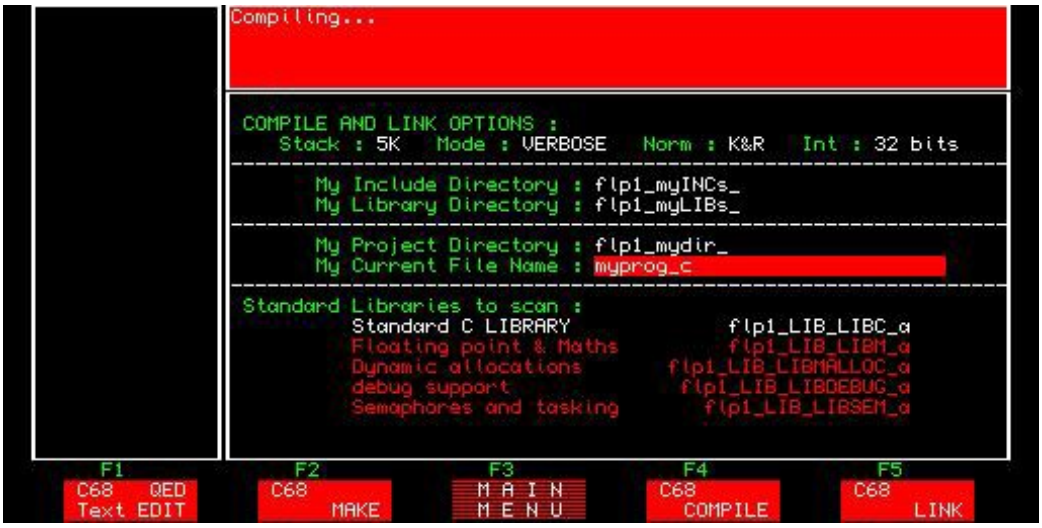
Here are the banner and menu file for starting either a PROLOG or one of two FORTH languages. **F1** will let use the simple text editor Qed, to create and update source files for these programs :

```
32722 DATA   2,7,'    QED    ',' Text EDIT '
32723 DATA  80,7,'  M A I N  ','  M E N U  '
32724 DATA   2,7,' Edimburgh ','   PROLOG  '
32725 DATA   2,7,'ComputerOne','   FORTH   '
32726 DATA   2,7,'DigitalPrec','Super FORTH'
32727 EXEC DEV$ &'qed':bye
32728 OK=32702 : bye : REM back to main menu
32729 EXEC DEV$ &'prolog':bye
32730 EXEC DEV$ &'forth':bye
32731 EXEC DEV$ &'forth83_job':bye
```

The menu file for C68 is special in that it will link to a more complex form editor for **F4** and **F5** :




```
32732 DATA   2,7,' C68   QED ',' Text EDIT '
32733 DATA   2,7,' C68       ','      MAKE '
32734 DATA  80,7,'  M A I N  ','  M E N U  '
32735 DATA   2,7,' C68       ','   COMPILE '
32736 DATA   2,7,' C68       ','      LINK '
32737 EXEC QED:C68:bye
32738 EXEC_W MAKE,#2,#2,#2;p$ & " -t " & Q$:C68:bye
32739 OK=32702 : CLS#2 : bye : REM back to main menu
32740 CLS#0:PRINT#0,"Compiling...";:PRINT#0,CCC$(0):C68:bye
32741 CLS#0:PRINT#0,"Linking...";:PRINT#0,CCC$(1):C68:bye
```



I deported this longer chunk of code at the end of this paper, because it is obsolete, and was just cut out of a previous "spaghetti" boot, so you can play with it but surely not use it (you would need the version 2.00 of the C68 tools on which it was based – compacted on a 720k floppy).

What I want to show you now is the evolution of the ways the **BYE** procedure, the menu files and others may be put together when the QL is booting. Basically, all has to be merged. This is done the simplest way with some SuperBasic code in a "**MANIFEST**" file which is called by a single **MRUN** command statement at the end of the **BOOT** file, following the **BYE** procedure. Here was my first and most basic version of this **MANIFEST** file.

```
 1 LRESPR 'mdv1_ptr_gen'
 2 LRESPR 'mdv1_MacMouse11'
 3 LRESPR 'mdv1_wman'
 4 LRESPR 'mdv1_Qptr'

 5 window#0,512,256,0,0:paper#0,0;:ink#0,0;: cls#0
 6 CLS#0:WINDOW#0,388,42,110,0:PAPER#0,2;:INK#0,7:BORDER#0,1,7
 7 WINDOW#1,94,225,14,0:PAPER#1,0;:INK#1,7:BORDER#1,1,7
 8 WINDOW#2,388,182,110,43:PAPER#2,0;:INK#2,4:BORDER#2,1,7
 9 OPEN#3,'scr_512x31a0x225':PAPER#3,0;:INK#3,4

10 MERGE mdv1_TOP_MENU
11 MERGE mdv1_PSION_MENU
12 MERGE mdv1_LANG_MENU
13 MERGE mdv1_C68_MENU
14 MERGE mdv1_CFORMS

15 OK = 32702 : KO = 15 : INITFORM
```

Before going on, let's look at the way the SuperBasic engine treats different kinds of statements :
1/ Statements without line numbers are called "commands" : these will be interpreted/executed immediately by the SuperBasic parser (also when they are "compound" statements with colon character separations).
2/ Statements with line numbers but outside any procedure or function blocks : these will be interpreted/executed after SuperBasic has finished loading (or merging) the files that contains them, following the numbering order.
3/ Numbered statements within procedures or functions : these will only be interpreted/executed when the procedures or functions are called (simply by their name – eventually with parameters). The functions and procedures can be named/called in (un-numbered) command statements or else in any numbered statement of type 2 or 3. Almost like (indeed) FORTH builds progressively on itself.
4/ The **DATA** keyword controls a special type of statements that must be numbered (outside procedures or functions) but that are not executable in a proper sense.

Here we have a first issue : in our menu files they are mixed with "target" statements that are also outside procedures and functions, but that are executable because they are numbered. In the **BYE** procedure these "target" statements are selected by a "computed" **GOTO** at line number 32766, but to execute only one of them, they must end calling **BYE** again. That' OK.

A second issue arises : after **MERGE**ing all the modules of my boot process, SuperBasic will look for the first executable statement outside any procedure or function. He will find this one :

```
32701 MRUN mdv1_TOOB
```

The little SuperBasic file "TOOB" has to conclude all the **MERGE** of the boot process...

By deleting the **MANIFEST** statements : so in its simplest form, it has only one command statement (ending with **BYE** !) :

```
DLINE 1 to KO : BYE
```

This could be all. BUT…

Last year, a thread on the Sinclair QL forum, inspired me a challenge : find the simplest way to code SuperBasic without line numbers : I wrote this 15 lines SuperBasic function :

```
 1 DEFine FuNction add_line_numbers(f$,first,interval)
 2 input_file$  = dev$ & f$ & "_txt"
 3 output_file$ = dev$ & f$ & "_bas"
 4 OPEN_IN#5,input_file$
 5 OPEN_OVER#6,output_file$
 6 l = first : i = interval
 7 REPeat number
 8    INPUT#5,l$ : k = LEN(l$) : IF k > 1 THEN
 9       IF CODE(l$(k))=13 THEN k = k-1
10       IF k > 0 THEN PRINT#6,IDEC$(l,5,0);" ";l$( 1 TO k )
11       ELSE i = 0 : END IF
12    IF EOF(#5) THEN EXIT number : ELSE l = l+i : i = interval
13 END REPeat number
14 CLOSE#6 : CLOSE#5 : RETurn l
15 END DEFine add_line_numbers
```

This function will take a file of un-numbered SuperBasic statements on **dev$_** with suffix **_txt** and output on the same **dev$_** a file with suffix **_bas** were all the lines will be numbered starting with **first** and paced by **interval**. The function will return the line number of the last statement. It will dismiss the blank lines and the spurious **CR** (code13) characters when needed.

As long you work on the same **_txt** file you will only store the latest corresponding **_bas** file.

This is my new boot file including the **ALN** (add_line_numbers) function and some commands :

```
DEV$="FLP1_" : MODE 4 : PROG_USE DEV$ : DATA_USE DEV$

32741 MRUN DEV$ & "TOOB"

32742 DEFine FuNction aln(f$,f,d)
32743 input_file$  = DEV$ & f$ & "_txt"
32744 output_file$ = DEV$ & f$ & "_bas"
32745 OPEN_IN#5,input_file$:OPEN_OVER#6,output_file$
32746 l = f : i = d
32747 REPeat number
32748    INPUT#5,l$ : k = LEN(l$) : IF k > 1 THEN
32749    IF CODE(l$(k)) = 13 THEN k = k - 1
32750    IF k > 0 THEN PRINT#6,IDEC$(l,5,0);" ";l$(1 TO k)
32751    ELSE i = 0 : ENDIF
32752    IF EOF(#5) THEN EXIT number : ELSE l = l+i : i = d
32753 END REPeat number
32754 CLOSE#6 : CLOSE#5 : RETurn l
32755 END DEFine aln
```

```
32757 DEFine PROCedure BYE
32758 PAPER#3,0:INK#3,4:CSIZE#3,0,0:CLS#3:AT#3,0,7
32759 PRINT#3,' F1',,'F2',,'F3',,'F4',,'F5':RESTORE OK
32760 FOR i = 5 TO 69 STEP 16
32761    READ p,q,p$,Q$:PAPER#3,p:INK#3,q
32762    AT#3,1,i:PRINT#3,p$:AT#3,2,i:PRINT#3,Q$
32763 END FOR i
32764 p = CODE(INKEY$(-1))/4
32765 IF p<58 OR p>62 : GO TO 32764
32766 CLS#0:CLS:CLS#2:GO TO p+OK-53
32767 END DEFine BYE

PRINT aln("MANIFEST",1,1)
MRUN DEV$ & "MANIFEST_bas"
```

The last lines of this **BOOT** file are commands calling **ALN** to add line numbers to **MANIFEST_txt** an un-numbered text file and then **MRUN** the resulting **MANIFEST_BAS** file ; in this file, **ALN** will be used too, to add line numbers to all the **_txt** modules (menus etc.) before merging them :

```
LRESPR DEV$ & 'ptr_gen'
LRESPR DEV$ & 'MacMouse11'
LRESPR DEV$ & 'wman'
LRESPR DEV$ & 'Qptr'

WINDOW#0,512,256,0,0:PAPER#0,0:INK#0,0:CLS#0
WINDOW#0,388,42,110,0:PAPER#0,2;:INK#0,7:BORDER#0,1,7
WINDOW#1,94,225,14,0:PAPER#1,0;:INK#1,7:BORDER#1,1,7
WINDOW#2,388,182,110,43:PAPER#2,0;:INK#2,4:BORDER#2,1,7
OPEN#3,'scr_512x31a0x225':PAPER#3,0;:INK#3,4

CFO = 32601
R = ALN("CFORMS",CFO,1)
TMO = R+1
R = ALN("TOP_MENU",TMO,1)
PMO = R+1
R = aln("PSION_MENU",PMO,1)
LMO = R+1
R = ALN("LANG_MENU",LMO,1)
CMO = R+1
R = ALN("C68_MENU",CMO,1)

MERGE DEV$ & "C68_MENU_bas"
MERGE DEV$ & "LANG_MENU_bas"
MERGE DEV$ & "PSION_MENU_bas"
MERGE DEV$ & "TOP_MENU_bas"
MERGE DEV$ & "CFORMS_bas"
```

Note that because of the **DATA** statements, **MANIFEST** has to remember the first line numbers of all the text files **ALN** will add line numbers to, before chain-merging them ; it does with the variables :

```
CFO = 32601
TMO = 32702
PMO = 32712
LMO = 32722
CMO = 32732
```

The **TOOB** file has to manage these variable line numbers :

```
KO = CFO-1 : OK = TMO
DLINE 1 TO KO
INITFORM : BYE
```

but all the files (menus, etc.) should also have their target statements adapted ; in the **TOP_MENU** for instance, the statements jumping to sub-menus :

```
OK = PMO : PAPER#2,0:INK#2,4:CLS#2 : bye
OK = LMO : PAPER#2,0:INK#2,4:CLS#2 : bye
OK = CMO : PAPER#2,0:INK#2,4:CLS#2 : C68 : bye
```

and in all the sub-menus, to return to the **TOP_MENU** :

```
OK = TMO : bye : REM back to main menu
```

Meaning that each time you boot the QL, the **MANIFEST** and all the modules it will **MERGE** as **_txt** files may be modified, because **ALN** will always treat them. However, when no files have been modified and **MANIFEST** has run once, all the **_bas** files are up to date. So, on all the next boot runs, the **ALN** sequence of the **MANIFEST** is no longer needed. This sequence could then simply be framed out by a SuperBasic **IF** structure :

```
CFO = 0
IF CFO > 0 THEN
OPEN_OVER#4,DEV$ & "TOOB"
PRINT#4,"CFO = "; : PRINT#4,CFO
R = ALN("CFORMS",CFO,1)
TMO = R+1
PRINT#4,"TMO = "; : PRINT#4,TMO
R = ALN("TOP_MENU",TMO,1)
PMO = R+1
PRINT#4,"PMO = "; : PRINT#4,PMO
R = aln("PSION_MENU",PMO,1)
LMO = R+1
PRINT#4,"LMO = "; : PRINT#4,LMO
R = ALN("LANG_MENU",LMO,1)
CMO = R+1
PRINT#4,"CMO = "; : print#4,CMO
R = ALN("C68_MENU",CMO,1)
PRINT#4,"KO = CFO - 1 : OK = TMO"
PRINT#4,"DLINE 1 TO KO"
PRINT#4,"INITFORM : BYE"
CLOSE#4
END IF
```

Needing only the update of one statement of the **MANIFEST** file : **CFO=0** instead of **CFO=32601**

The only problem when you skip the **ALN** sequence is that the QL does not know the line number variables. This is easily solved by updating **TOOB** - as you can notice - in the ALN sequence itself.

With this simple example of a boot process, I wanted to surline how a plain SuperBasic **MANIFEST** file may be used to control modular and conditionnal loading of un-numbered basic programs. The software modules should only contain procedures and functions and the **DATA** statements have only to be the first ones in the modules. Even **GOTO** statements are allowed if they are “computed”.

As a further example of this technique, my (obsolete) **CFORMS** module that controls edition of the C68 compiler and linker options has two "computed" **GOTO** statements at lines 32642 and 32695.

```
32601 DATA 18,4,0,2,0,2,2,0,7,7,12
32602 DATA 12,2,"3K"," -=3072",5,18,2
32603 DATA 12,2,"5K"," -=5120",5,18,3
32604 DATA 12,2,"7K"," -=7168",5,18,4
32605 DATA 12,2,"9K"," -=9216",5,18,1
32606 DATA 24,2,"VERBOSE"," -v",7,1,6
32607 DATA 24,2,"LACONIC","",7,1,5
32608 DATA 41,2,"ANSI"," -unproto",9,5,8
32609 DATA 41,2,"K&R ","",9,5,7
32610 DATA 53,2,"16 bits"," -Qshort",13,7,10
32611 DATA 53,2,"32 bits","",13,7,9
32612 DATA 30,7,""," flp1_mydir_",12,14,4
32613 DATA 30,8,""," myprog_c",15,11,1
32614 DATA 30,4,""," -Iflp1_myINCs_",14,9,2
32615 DATA 30,5,""," -Lflp1_myLIBs_",11,13,3
32616 DATA 10,12,"Floating point & Maths         flp1_LIB_LIBM_a"," -
lm",16,12,15
32617 DATA 10,13,"Dynamic allocations       flp1_LIB_LIBMALLOC_a"," -
lmalloc",17,15,16
32618 DATA 10,14,"debug support              flp1_LIB_LIBDEBUG_a"," -
ldebug",18,16,17
32619 DATA 10,15,"Semaphores and tasking       flp1_LIB_LIBSEM_a"," -
lsem",1,17,18
32620 DATA 0,1,0,0,1,0,0,1,0,1,1,1,1,1,0,0,0,0,11
32621 DATA 1,1,0,4,"COMPILE AND LINK OPTIONS :"
32622 DATA 4,2,0,4,"Stack :      Mode :           Norm :       Int :"
32623 DATA
0,3,0,7,"-----------------------------------------------------------------
"
32624 DATA 7,4,0,4,"My Include Directory :"
32625 DATA 7,5,0,4,"My Library Directory :"
32626 DATA
0,6,0,7,"-----------------------------------------------------------------
"
32627 DATA 7,7,0,4,"My Project Directory :"
32628 DATA 7,8,0,4,"My Current File Name :"
32629 DATA
0,9,0,7,"-----------------------------------------------------------------
"
32630 DATA 1,10,0,4,"Standard Libraries to scan :"
32631 DATA 10,11,0,7,"Standard C LIBRARY             flp1_LIB_LIBC_a"
32632 DEFine PROCedure INITFORM
32633 o=CFO:RESTORE o:READ m:DIM f(m):READ n:DIM r$(n,32)
32634 DIM CC(7):FOR i=0 TO 7:READ CC(i):NEXT i:READ cp
32635 RESTORE o+m+1:FOR i=1 TO m:READ f(i):NEXT i:READ MMM
32636 FOR i=1 TO m:READFLD(o+i):IF m$="":r$(jt)=n$
32637 END DEFine
32638 DEFine PROCedure EDITFORM
32639 o=CFO:CLS#2:RESTORE o+m+2:p$=""
32640 FOR i=1 TO MMM:READ
x,y,jp,jt,m$:PAPER#2,jp:INK#2,jt:AT#2,y,x:PRINT#2,m$
32641 c=cp:READFLD(c+o)
32642 c=NEXTFLD(c):IF c<>cp:GO TO CFO + 41
32643 REPeat ScanKeyb
```

```
32644 a=CODE(INKEY$(-1))
32645 SELect ON a
32646 ON a=208:c=PREVFLD(c)
32647 ON a=216:c=NEXTFLD(c)
32648 ON a=32:c=TOGGLE(c)
32649 ON a=10:DISPFLD c,0:EXIT ScanKeyb
32650 ON a=27:C68:bye
32651 END SELect
32652 END REPeat ScanKeyb
32653 FOR i=1 TO m
32654 READFLD(o+i):IF m$="":n$=r$(jt)
32655 IF ((m$<>"") OR ((jt>1) AND (jt<n))) AND (f(i)=1):p$=p$&n$
32656 END FOR i
32657 dat$=r$(n,4 TO LEN(r$(n))): REM p$ = p$ & " -tmp" & dat$
32658 END DEFine
32659 DEFine FuNction CCC$(opt)
32660 EDITFORM:INK#2,4:CLS#2:nnn="_c"INSTR r$(1)
32661 IF nnn=0:nnn=len(r$(1))+1:r$(1)=r$(1)&"_c"
32662 IF opt=0:p$="-c "& dat$ & r$(1,4 TO nnn+1)&" "&p$
32663 IF opt=1:p$="-o"&dat$&r$(1,4 TO nnn-1)&" "&dat$&"*_o "&p$
32664 EXEC_W CC,#2,#2,#2;p$
32665 PRINT#0," Done... Hit any key !";:return inkey$(-1)
32666 END DEFine
32667 DEFine FuNction TOGGLE(i)
32668 IF m$="" THEN
32669 EDITFLD(jt)
32670 ELSE
32671 IF jt=i :f(i)=1-f(i)
32672 IF jt<>i:f(i)=0:i=jt:f(i)=1:READFLD(i+o)
32673 END IF
32674 DISPFLD i,1:RETurn i
32675 END DEFine
32676 DEFine FuNction PREVFLD(i)
32677 DISPFLD i,0:i=jp:i=FINDFLD(i):DISPFLD i,1:RETurn i
32678 END DEFine
32679 DEFine FuNction NEXTFLD(i)
32680 DISPFLD i,0:i=jn:i=FINDFLD(i):DISPFLD i,1:RETurn i
32681 END DEFine
32682 DEFine PROCedure DISPFLD(i,s)
32683 PAPER#2,CC(2*f(i)+s):INK#2,CC(4+2*f(i)+s)
32684 AT#2,y,x:PRINT#2,m$;:IF m$="":PRINT#2,r$(jt,4 TO)
32685 END DEFine
32686 DEFine PROCedure EDITFLD(ii)
32687 PAPER#2,CC(2*f(i)):INK#2,CC(4+2*f(i))
32688 AT#2,y,x:PRINT#2,FILL$(" ",29)
32689 AT#2,y,x:INPUT#2,O$:r$(ii)=r$(ii,1 TO 3)&O$
32690 END DEFine
32691 DEFine PROCedure READFLD(l)
32692 RESTORE l:READ x:READ y:READ m$:READ n$:READ jn:READ jp:READ jt
32693 END DEFine
32694 DEFine FuNction FINDFLD(i)
32695 READFLD(i+o):IF (jt<>i)AND(f(i)=0):i=jt:GO TO CFO + 94
32696 RETurn i
32697 END DEFine
32698 DEFine PROCedure C68
32699 PAPER#2,0:INK#2,4:CLS#2:VIEW#2,dev$ &'C68_help'
32700 END DEFine C68
32701 MRUN mdv1_TOOB
```