TWS Basic

di Tommaso Romanazzi - Livorno

Il TWS Basic (Tomsoft Window System Basic) è l'evoluzione di un mio precedente interprete Basic; questa evoluzione ha seguito due direttive: la realizzazione di un tool che 1) semplificasse la realizzazione di programmi 'scolastici' e 2) mettesse a disposizione un'interfaccia grafica che almeno nelle intenzioni ricordasse quelle di S.O. più seri. Credo quindi di aver realizzato un Basic che pur nelle sue mancanze può essere molto utile, visto e considerato che fino ad ora non esistono per C64 dei linguaggi che gestiscano delle finestre (anche un eventuale GeoBasic avrebbe, nella sua potenza, notevoli limitazioni dal punto di vista della memoria a disposizione).

Dopo questa breve introduzione, analizziamo come il TWS Basic funziona, ovvero esaminiamo come viene modificato il sistema operativo ed il Basic.

Per implementare questo interprete ho usaato i Token; il cui uso è forse più complesso, ma che è certamente più 'pulito' e corretto di altri.

I Token consistono nella compattazione delle parole chiave (costituite da più lettere) in un singolo byte che viene usato come codice per quel comando dalle successive routine del Basic.

Per fare questo è necessario cambiare i vettori del S.O. delle locazioni $0304 (tokenizzazione), $0306 (list), $0308 (esegue comandi) $030a (Valutazione argomenti); quest'ultima modifica si è resa necessaria in quanto il TWS Basic ha alcune nuove funzioni e conversioni numeriche.
Inoltre ho modificato anche le routine di gestione errori (vettore $0300), e il Warm Start (vettore $0302).

Dopo questa scarna introduzione (per ulteriori approfondimenti vedere i numeri relativi di MC), passiamo all'elenco delle nuove istruzioni. Le ho suddivise per praticità in 5 sezioni (titoli in inglese perché la traduzione fa... perdere qualcosa!): Programmers' tools, Program management, Disk management, Graphic management e Advanced graphic management.

Le parentesi quadre, nelle descrizioni, indicano un argomento opzionale.

Il nome degli argomenti è spesso autoesplicante (almeno per chi è un po' esperto), e al loro posto vi può essere sempre sia una variabile, una costante o una espressione. Le espressioni stringa sono indicate da nomi fra doppi apici, ma possono essere anche variabili o espressioni.

Sotto la denominazione di Programmers' tools vanno quei comandi di cui si sente la mancanza scrivendo programmi col C64.

Comandi:

RUN [«filename»[,device]]
Oltre al normale funzionamento, il comando ora può eseguire anche programmi residenti sul dispositivo eventualmente indicato.
OLD
Esegue l'operazione inversa di NEW: rende possibile riottenere il programma Basic cancellato con NEW o con un reeset, purché non siano state inserite nuove linee Basic o usate variabili.
HELP
Lista la linea in cui si è verificato l'ultimo errore, evidenziando gli ultimi caratteri letti in reverse.
DELETE line[—[line]]
Sintassi identica a quella di LIST, solo che le linee specificate saranno cancellate invece che listate.
KILL
Resetta il computer disabilitando la gestione da parte del TWS Basic del RunStop-Restore e del Reset.
Locate col,row
Posiziona il cursore del modo testo alla colonna «col» e riga «row».
RESTORE [line]
È ora possibile spacificare la linea da cui si vuole iniziare a leggere i DATA; «line» è un'espressione qualsiasi e non viene fatto alcun controllo sulla presenza di istruzioni DATA in quella linea.
DOKE addr, word
È la POKE a 16 bit; durante la scrittura della parola «word» vengono disabilitate le interruzioni.
' L'apice è sinonimo di REM.

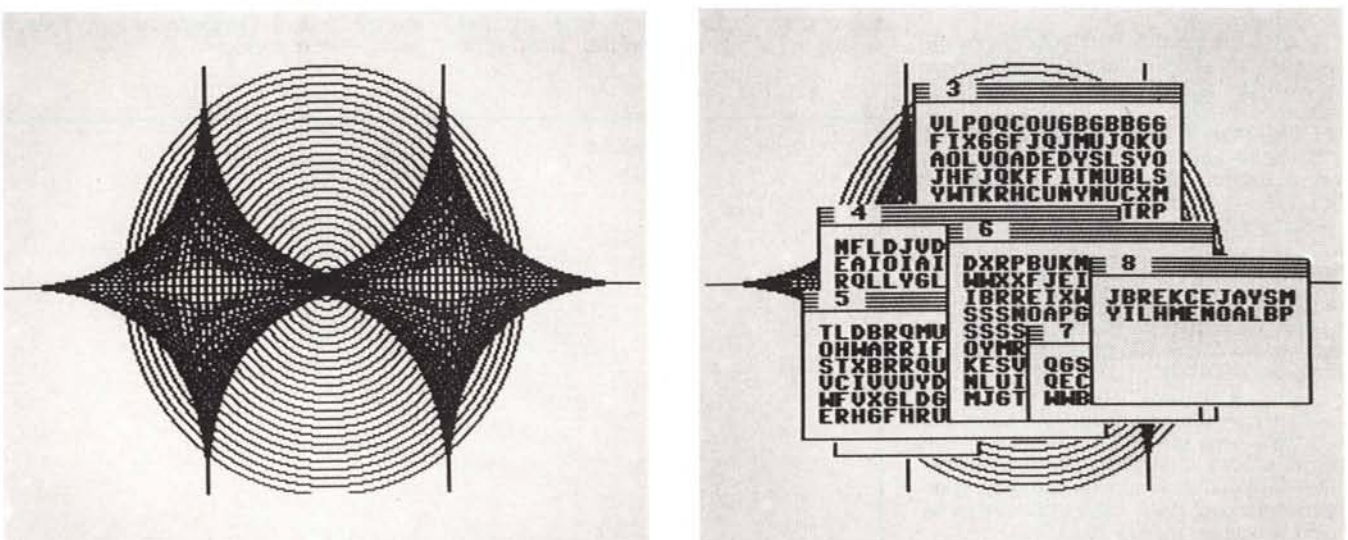
Funzioni:

DEEK (addr)
È la PEEK a 16 bit, ritorna cioè peek(addr)+256*peek(addr+1).
$, &, %: possono essere utilizzati (eccetto che in fase di INPUT) numeri in base diversa

Listato 1 - Window Demo

```
10 INIT:HCLR:MODE1:SCREEN 1,1,1,0
15 FOR X=0 TO 100 STEP 4:CIRCLE 160,100,X:NEXT
20 FOR X=0 TO 100 STEP 4
30 LINE 100-X,100,100,X
32 LINE 100-X,100,100,200-X
34 LINE X+100,100,100,X
36 LINE X+100,100,100,200-X
40 LINE 220-X,100,220,X
42 LINE 220-X,100,220,200-X
44 LINE X+220,100,220,X
46 LINE X+220,100,220,200-X
50 NEXT
99 '--------------
100 REPEAT
101 '
110 REPEAT
115 C=RND(0)*30:R=RND(0)*19
120 W=RND(0)*(34-C)+6:H=RND(0)*(20-R)+5
130 FL=TOPBUF(0)+H*W*8+4<65536
131 '
140 CIF FL THEN
150 WOPEN C,R,W,H:WSET:WCLR:FRAME:MODE3:TEXT C+1,R,WINDOW
160 FOR Y=R+2 TO R+H-2
165 FOR X=C+1 TO C+W-2:TEXT X,Y,"(SWLC)":CHR$(RND(0)*25+65):NEXT
170 NEXT:MODE1
180 ENDIF
190 UNTIL NOT FL
191 '
200 FOR I=WINDOW TO 1 STEP-1:WPUT:WCLOSE:NEXT
210 SET A$
220 UNTIL A$<>""
230 INIT
```

Listato con il quale si ottengono i due esempi della pagina a fronte.



Esempio di grafica e finestre ottenute facilmente con il programma riportato nel listato 1.

da quella decimale: «$» precede un numero esadecimale; «&» precede ottali e «%» indica numeri binari.

Il numero di cifre è libero purché il valore decimale stia all'interno dei limiti del FloatingPoint (circa 1 E 38).

La sezione del Program management contiene i comandi che modificano il flusso del programma avvicinando il Basic al mondo dei linguaggi strutturati.

Comandi:

IF-THEN-ELSE

La struttura IF-THEN è stata estesa con ELSE; può essere annidato, nel qual caso ogni ELSE viene considerato relativo all'ultimo IF.

CIF-THEN-CELSE-ENDIF

Questa nuova «struttura» è identica come principio a quella precedente, ma il suo funzionamento è esteso a più linee di programma; cioè CELSE e ENDIF non necessariamente devono essere nella stessa linea del CIF, ma anche molte linee più avanti, senza limitazioni. Può essere annidato e contenere IF-THEN-ELSE.

REPEAT-UNTIL condition

Ripete le istruzioni comprese tra REPEAT e UNTIL finché «condition» è non nulla (cioè vera).

POP FOR

Esce forzatamente dall'ultimo FOR-NEXT; il programma continua dopo il POP-FOR senza saltare al NEXT.

POP GOSUB

Toglie dallo stack l'ultimo GOSUB, senza ritornarvi.

TRAP(arg)

Abilita la gestione degli errori: in caso di errore, il controllo viene passato alla linea indicata dall'espressione «arg». Se durante l'esecuzione della routine di gestione degli errori si verifica un altro errore, il programma si arresta e viene visualizzato il messaggio «?ERROR IN TRAP MODE ERROR».

Se «arg»=−1 viene disabilitata la gestione degli errori.

Il modo diretto la gestione degli errori è ancora valida a meno che non vengano eseguiti i comandi DELETE o OLD.

RESUME [line]

Dopo un errore (se è abilitata la gestione degli errori con TRAP), il programma ritorna all'istruzione successiva a quella che ha causato l'errore. Se è presente l'espressione «line», allora il programma riparte dalla linea specificata.

Funzioni:

ERRL

Questa variabile contiene il numero della linea in cui si è verificato l'ultimo errore. Se ERRL>63999, l'errore si è verificato in modo diretto.

ERRN

Contiene il codice dell'errore.

Il Disk Management è l'insieme delle istruzioni che servono a semplificare l'uso del drive (device 8), senza incomprensibili sequenze di OPEN e PRINT#.

Comandi:

SEND «disk command»

Invia al drive il messaggio.

DIR [mode,] [«pattern»]

Visualizza la directory del disco; «mode» è la lettera opzionale che può essere 'E' oppure 'N':

'N' (= normal) indica la visualizzazione consueta della directory;

'E' (= extended) indica che la directory del disco deve essere visualizzata sostituendo al tipo del file (SEQ, PRG, ecc...) gli ultimi caratteri del nome del file, se sono al più 3, preceduti da un punto, e il primo di essi è una lettera esempio:

```
DIR
  0 «dir sample» ds 2a
  8 «sort.bas» PRG
  656 blocks free
DIR E
  0 «dir sample» ds 2a
  8 «sort» BAS
  656 blocks free
```

La stringa «pattern» contiene l'eventuale

È disponibile, presso la redazione, il disco con il programma pubblicato in questa rubrica. Le istruzioni per l'acquisto e l'elenco degli altri programmi disponibili sono a pag. 235.

pattern di visualizzazione dei file, necessario per avere la directory selettiva; gli esempi riportati in fondo alla pagina 11 del manuale in inglese del drive corrispondono a:

DIR «TEST»
DIR «T*»
DIR «T?ST»
DIR «T?ST*»

DOPEN filnum, «filnam»
Apre il file 'filnum' su drive (l'indirizzo secondario è gestito in modo automatico: vedere dopo).

SIZE reclen
Specifica la lunghezza (1-255) dei record per l'uso e l'apertura di un file che deve essere fatta da DOPEN filnum, «filnam, L» preceduto da SIZE.

RECORD#filnum, record, position
Posiziona il puntatore del file «filnum» sul record «record» alla posizione «position».

DCLOSE
Chiude il file aperto con DOPEN.

Funzioni:

DISK$
Ogni chiamata di questa variabile causa la lettura dello status del drive che viene memorizzato in DISK$.

Alle sezioni precedenti segue la parte che si occupa della grafica; è suddivisa in due sezioni perché la prima tratta i comandi primitivi della grafica, mentre la seconda riguarda la gestione delle «finestre». Vediamo quindi il Graphic management.

L'uso di valori che eccedono le coordinate di schermo, purché nel range +32767, −32768 non provoca messaggi di errore nei comandi PLOT, LINE, DRAW, BOX e CIRCLE.

Comandi:

MODE n
Seleziona il modo di tracciamento dei successivi comandi grafici:

- n = 0 cancella
- n = 1 traccia (AND)
- n = 2 scambia — toggle — (XOR)
- n = 3 da usare con i caratteri se eventuali informazioni presenti sotto il carattere vadano perse.

SCREEN resmod[,brdr[,paper[,ink]]]
Seleziona la risoluzione ed i colori da utilizzare: si passa nella bassa risoluzione (40*25) se 'resmod'=0; con 'resmod'=1 si passa in alta risoluzione (320*200).
'brdr' è il colore del bordo (0-15)
'paper' è il colore dello sfondo (0-15)
'ink' è il colore dei punti in hires, dei caratteri in lores.

HCLR [pattern]
Riempie lo schermo hires col 'pattern' specificato (0-255); se non è presente, lo schermo viene pulito, azzerato.

PLOT x, y
Opera sul punto di coordinate X (0-319), y (0-199) a seconda del valore specificato con MODE.

LINE x1,y1,x2,y2
Traccia la linea che va da (x1,y1) a (x2,y2).

DRAW x2,y2
Traccia la linea che congiunge l'ultimo punto tracciato a (x2,y2).

BOX x1,y1,x2,y2
Traccia il rettangolo di diagonale (x1,y1)-(x2,y2).

CIRCLE x,y,r
Disegna il cerchio di raggio 'r' e centro in (x,y).

TEXT col, row, argmnt
Stampa in alta risoluzione alla colonna 'col' e riga 'row' il resto indicato da 'argmnt': questo segue le stesse regole della PRINT, cioè accetta stringhe e/o numeri, oltre a TAB, SPC e i caratteri «;» e «,».

Il set di caratteri (maiuscolo/minuscolo) è quello usato in bassa risoluzione, ma può essere modificato con chr$(14), chr$(142) e con chr$(18) — RVS ON 8 — e chr$(146) — RVS OFF —.

Finalmente siamo arrivati alla sezione delle window, forse la più appariscente di tutte.

Comandi:

WOPEN row, col, wid, hgt
Riserva in RAM spazio sufficiente per la window di dimensioni specificate; (row, colr) è l'angolo superiore sinistro della window, mentre 'wid' è l'ampiezza e 'hgt' l'altezza della window stessa.

Listato 2 - Math Pack 64

```
100 INIT:MODE1
200 DIM F(320)
205 MODE1:HCLR:SCREEN 1,1,1,0:PRINT"(SWLC)(DISH)"
210 WOPEN 4,0,33,13:WGET:WCLR:FRAME:MODE3:TEXT 6,0,"  "
220 MODE1:PRINT"(SWLC)(DISH)":SCREEN1,1,1,0:GOSUB530
260 TEXT 5, 2,"      INPUT NUOVA FUNZIONE          "
280 TEXT 5, 3,"    OUTPUT GRAFICO  Y=F(X)          "
300 TEXT 5, 4,"   RICERCA SOLUZIONI REALI          "
320 TEXT 5, 5,"   RICERCA MASSIMI E MINIMI         "
340 TEXT 5, 6,"       INTEGRALE DEFINITO           "
360 TEXT 5, 7,"   DERIVATA PRIMA E SECONDA         "
380 TEXT 5, 8,"          PAGINA GRAFICA            "
400 TEXT 5, 9,"               END                  "
420 TEXT 5,11,"    (C)1987     & SOFT              "
430 BOX 55,15,272,80
440 BOX 47,85,280,97
450 MOUSE 482,7,2,27,8
460 ON INT(SPRTY/8)-1 GOSUB 500,570,1120,1280,1040,950,860,490
470 GOTO 450
489 '--------------------
490 INIT:END
499 '--------------------
500 WOPEN0,10,40,9:WGET:WCLR:FRAME:MODE3:TEXT2,10,"INPUT FUNZIONE":MODE1
505 TEXT 0,12,"FUNZIONE CORRENTE:":TEXT 0,13,FU$
510 TEXT 0,15,"NUOVA FUNZIONE:":TEXT 0,16,"F(X)=":C=5:R=16:L=56:GOSUB60000
512 IF S$=""THEN 510
514 SCREEN 0,1,1,1
515 PRINT"(CLR)(DOWN)(DOWN)(DOWN)530DEFFNY(X)=";S$:PRINT"535FU$="CHR$(34)S$CHR$(
34):PRINT"RUN200(HOME)"
520 POKE631,13:POKE632,13:POKE633,13:POKE198,3:END
529 '--------------------
530 DEFFNY(X)=X*X-4
535 FU$="X*X-4"
540 DEFFNY1(X)=(FNY(X+1E-4)-FNY(X-1E-4))/(2E-4)
550 DEFFNY2(X)=(FNY1(X+1E-4)-FNY1(X-1E-4))/(2E-4)
560 DEFFNAR(Y)=INT(Y*1E6+.5)/1E6:RETURN
569 '--------------------
570 GOSUB 890:X1=T1:X2=T2
610 WOPEN13,14,14,4:WGET:WCLR:FRAME:MODE3:TEXT14,16,"UN ATTIMO...":MODE1
620 KX=320/(X2-X1)
650 Y1=1E+38:Y2=-1E+38
660 FOR X=X1 TO X2 STEP 1/KX
670 Y=FNY(X):F((X-X1)*KX)=Y
680 IF Y<Y1 THEN Y1=Y
690 IF Y>Y2 THEN Y2=Y
700 NEXT
710 WPUT:WCLOSE:WSWAP:HCLR
730 KY=KX:IF Y1<>Y2 THEN KY=200/(Y2-Y1)
760 IF X1<0 AND X2>0 THEN X=-X1*KX:LINE X,0,X,199
790 IF Y1<0 AND Y2>0 THEN Y=-Y1*KY:LINE 0,Y,319,Y
810 FOR X=0 TO 319
820 PLOT X,100-(F(X)+Y1)*KY
850 NEXT:GOTO865
860 WSWAP
865 MOUSE0:SCREEN 1,0,1,0
870 REPEAT:GETA$:UNTIL A$<>""
875 SCREEN1,1,1,0
880 WSWAP:RETURN
889 '--------------
890 MODE1:WOPEN 10,15,20,7:WGET
892 MODE1:WCLR:FRAME:MODE3:TEXT 12,15,"INTERVALLO:":MODE1
895 TEXT 11,17,"ESTREMO INFERIORE:":C=11:R=18:L=15:GOSUB60000:T1=VAL(S$)
900 TEXT 11,19,"ESTREMO SUPERIORE:":C=11:R=20:L=15:GOSUB60000:T2=VAL(S$)
905 IF T1>=T2 THEN 892
910 MODE1:WPUT:WCLOSE:RETURN
919 '--------------
950 WOPEN0,6,40,10:WGET:WCLR:FRAME:MODE3:TEXT 3,6,"DERIVAZIONE":MODE1
952 SP$="                        "
960 TEXT1,8,SP$
965 TEXT1,8,"X=":C=3:R=8:L=15:GOSUB60000:IF S$="" THEN WPUT:WCLOSE:MODE1:RETURN
970 MODE3:TEXT 1,10,SP$
975 X=VAL(S$):Y=FNY(X):TEXT 1,10,"  F("X")="Y
980 TEXT 1,12,SP$
985 Y=FNY1(X):Y=FNAR(Y):TEXT 1,12," F'("X")="Y
990 TEXT 1,14,SP$
995 Y=FNY2(X):Y=FNAR(Y):TEXT 1,14,"F''("X")="Y
1000 GOTO960
1039 '--------------
1040 WOPEN0,6,40,10:WGET:WCLR:FRAME:MODE3:TEXT 3,6,"INTEGRALE DEFINITO":MODE1
1042 GOSUB890
```

Una versione di Math Pack 64 realizzata per mezzo del TWS Basic (vedi listato 2).

```
1045 TEXT 1,8,"SUDDIVISIONI?":C=1:R=9:L=15:GOSUB60000:C=VAL(S$)
1050 J1=FNY(T1):S=(T2-T1)/C:A=0
1055 FOR X=T1+S TO T2 STEP S:J2=FNY(X):A=A+S*(J2-J1)/2:J1=J2:NEXT
1060 TEXT 1,10,"(SWUC)A:"TEXT 1,11,"I":TEXT 1,12,"(SWLC)F(X)DX=":A
1065 TEXT1,13,"I":TEXT1,14,"(SWUC)+"
1070 MOUSE0:REPEAT:GET A$:UNTIL A$<>""
1075 SCREEN1,1,1,0:WPUT:WCLOSE
1080 RETURN
1119 '--------------
1120 WOPEN0,6,40,10:WGET:WCLR:FRAME:MODE3:TEXT 3,6,"-ICERCA SOLUZIONI":MODE1
1122 SP$="                                    "
1125 GOSUB890:FR=0
1130 TEXT 1,8,"INTERVALLINI ?":C=1:R=9:L=15:GOSUB60000:PR=VAL(S$)
1135 MOUSE0:SCREEN 0,1,1,0:PRINT"(CLR) ASCISSA:"
1140 R=(T2-T1)/PR
1150 J1=SGN(FNY(T1)):S=R
1160 REPEAT
1165 T1=T1+S:J3=FNY(T1):J2=-SGN(J3)*(ABS(J3)>1E-8)
1170 :IF J1<>J2 THEN
1190 :IF FR=0THENSS=S:FR=1
1200 ::IF J1=0 OR J2=0 THEN
1210 :::I=T1+S*(J1=0)
1220 :::PRINT TAB(6);I:FR=0:T1=T1+R
1230 :::ELSE:T1=T1-S:S=S/2:J1=SGN(FNY(T1))
1240 :ENDIF
1245 ENDIF
1250 UNTIL T1>=T2
1260 PRINT"(RVS)(RGHT)(DOWN)     FINE INTERVALLO     "
1265 GOTO1070
1279 '--------------
1280 WOPEN0,6,40,10:WGET:WCLR:FRAME:MODE3:TEXT 3,6,"-ICERCA MAX & MIN":MODE1
1285 GOSUB890:FR=0
1290 TEXT 1,8,"INTERVALLINI ?":C=1:R=9:L=15:GOSUB60000:PR=VAL(S$)
1295 MOUSE0:SCREEN0,1,1,0:PRINT"(CLR)(RVS)  ASCISSA   ###  ORDINATA  "
1300 R=(T2-T1)/PR
1310 J1=SGN(FNY1(T1)):S=R
1320 T1=T1+S:J3=FNY1(T1):J2=-SGN(J3)*(ABS(J3)>1E-20)
1325 IFJ1<>J2ORS<1E-15THEN1350
1330 IFT1<T2THEN1320
1340 PRINT"(RVS)(RGHT)(DOWN)     FINE INTERVALLO     "
1345 GOTO1070
1350 IFFR=0THENSS=S:FR=1
1360 IFJ1=0THENZ=T1-S:GOTO1390
1370 IFJ2=0THENZ=T1  :GOTO1390
1380 T1=T1-S:S=S/2:J1=SGN(FNY1(T1)):GOTO 1320
1390 T=FNY(Z):Z$=LEFT$("     ",4-LEN(STR$(INT(T))))+STR$(T)
1400 J3=FNY2(Z)
1410 IF J3<0 THEN PRINT Z;TAB(12)"MIN"TAB(17);Z$
1420 IF J3>0 THEN PRINT Z;TAB(12)"MAX"TAB(17);Z$
1430 FR=0:T1=T1+R
1440 GOTO1310
59997 "" ┌──────────────────────┐
59998 ""|INPUT STRING SUBROUTINE |
59999 "" └──────────────────────┘
60000 MODE3:POKE198,0
60005 S$="":F$=" ":A$="":N=0:CT=0
60010 CT=CT+1:IF CT<8 THEN 60020
60015 CT=0:IF F$=" "THEN F$="(RVS) ":ELSE:F$=" "
60020 IF N<L THEN TEXT C,R,F$
60030 GETA$:IFA$=""THEN60010
60040 IF A$=CHR$(20)THEN 60080
60045 IF A$=CHR$(13)THEN 60095
60050 IF N>L-1 THEN 60010
60060 TEXT C,R,A$
60065 S$=S$+A$:N=N+1:C=C+1:IF C>39 THEN C=0:R=R+1
60070 GOTO 60010
60075 '
60080 IF N<1 THEN 60010
60083 N=N-1:S$=LEFT$(S$,N):C=C-1:IF N<L-1 THEN TEXT C+1,R," "
60087 IF R<0 THEN C=39:R=R-1:IF R<0THEN R=0
60090 GOTO 60010
60092 '
60095 IF N<L THEN TEXT C,R," "
60097 MODE1:RETURN
```

WCLOSE

Rilascia la RAM occupata dall'ultima finestra aperta.

WGET [wn[,row,col]]

Senza alcun parametro, memorizza l'ultima window definita nel corrispondente buffer in RAM. Se è presente 'wn' (1-32), allora l'operazione viene eseguita per la finestra 'wn'. Se sono presenti 'row' e 'col', allora il comando lavora sulla window 'wn' usando però le nuove 'row' e 'col' (ampiezza e altezza sono immutate).

WPUT [wn[,row,col]]

Stessa sintassi di WGET, ma esegue il compito inverso: legge il buffer in RAM e lo mette sullo schermo.

WSWAP [wn[,row,col]]

Stessa sintassi di WGET, però la sua funzione è quella di scambiare la window col suo buffer in RAM.

WCLR [wn[,row,col]]

Sintassi analoga a WGET, ma pulisce la window.

FRAME [wn[,row,col]]

Sintassi analoga a WGET, ma disegna una cornice per la window.

MOUSE mode [,row, col, wid,hgt]

Gestisce la freccia leggendo la porta joystick # a seconda del valore di 'mode' che può essere:

- = 0 disabilita la freccia; non sono necessari i parametri successivi;
- = 1 attende la pressione del fire; non sono necessari i successivi valori;
- = 2 attende la pressione del fire; i 4 parametri delimitano una window in cui evidenziata la linea in cui si trova la freccia;
- = 129 attende la pressione del fire all'interno della window specificata;
- = 130 attende la pressione del fire all'interno della window specificata, evidenziando la linea in cui si forma la freccia.

INIT

Inizializza il chip video in bassa risoluzione, bordo e sfondo blu, caratteri bianchi. La variabile WINDOW viene azzerata e rilasciata tutta la RAM utilizzata da eventuali window (cosa che non avviene con CLR o RUN).

Funzioni:

WINDOW

Ritorna il numero di window correntemente definite; se=0, allora nessuna window è definita.

TOPBUF (arg)

Se 'arg' = 0 ritorna l'indirizzo più alto usato nel buffer per le window, quindi TOPBUF(0)−$c000 dà il numero di byte usati in totale dalle window, e $ffff−TOPBUF(0) dà il numero di byte ancora utilizzabili.

Se 'arg'<>0 allora la funzione ritorna l'indirizzo dal quale inizia il buffer della window 'arg', se definitiva.

SPRTX

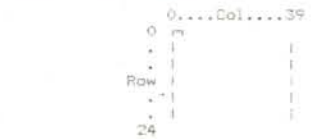
È la coordinata X (0-511) della freccia (se 320<SPRTX<511, la freccia è fuori schermo).

SPRTY

È la coordinata Y (0-255) della freccia (se 200<SPRTY<255, la freccia è fuori schermo).

Ciò che segue sono una serie in schemi che chiariscono i vari sistemi di coordinate:

coordinate in bassa risoluzione (LOCATE):



Coordinate in alta risoluzione (PLOT, LINE, DRAW, ecc...):

Coordinate della freccia:

I Token

Il vantaggio dei Token è che viene eliminata parte della lentezza intrinseca nel processo di interpretazione; per chiarire le idee mettiamo di avere la linea

PRINT peek (1024+RND(0))

l'interprete, dovendo eseguire questa linea, dovrebbe in continuazione cercare uno fra le decine di comandi nella stringa che costituisce la linea. Ma se «avvertiamo» l'interprete dei comandi presenti nella linea, la sua esecuzione risulterà più rapida. Tokenizzando la linea, si otterrà qualcosa del genere:

X Y (1024+Z(0))
↑ ↑ ↑
token

i caratteri X Y Z sono particolari in quanto sono gli unici ad avere il bit7 settato; in altre parole l'interprete dovrà leggere byte per byte la linea: se legge un byte col bit7 a 1, allora è un comando e ne usa il valore per chiamare la routine che lo esegue. In una volta sola viene riconosciuto un comando anche di parecchi caratteri; naturalmente il tempo guadagnato si ripaga (infatti linee piene di comandi richiedono più tempo di quelle corte per la loro immissione in memoria soprattutto per la diversa quantità di confronti con la tavola dei comandi); si complica un po' la gestione delle linee: basti pensare al LIST che «decomprime» i token.

In conclusione viene ottimizzata l'interpretazione a scapito di altre cose (complicazioni — relativa — delle routine che gestiscono le linee del programma).

Vediamo un'altra cosa: se il bit7 è a 1, abbiamo a disposizione 127 diverse parole chiave possibili; e il TWS Basic ha esattamente 127 parole chiave (contando anche i simboli e le parole chiave del Basic 2.0).

Con questo si potrebbe pensare di sapere tutto ciò che serve per realizzare effettivamente un'estensione Basic basata sui token; invece no!!!

Credo che ben pochi abbiano notato una sottile differenza tra il TWS Basic e l'ADP Basic: le parole chiave del primo contengono anche parole chiave del Basic 2.0, mentre il secondo no; ricorderete probabilmente la domanda che fece A.D.P. riguardo il comando FRMAT che non poteva chiamarsi FORMAT.

Il problema sta nell'ordine delle parole nell'apposita tavola; se avete un monitor, vedrete nella zona di ROM da $a09e a $a19d la tavola delle keyword del Basic 2.0 e in essa INPUT è preceduto da INPUT# come PRINT è preceduto da PRINT#, perché altrimenti INPUT# e PRINT# non sarebbero mai riconosciuti come token (compreso il '#'). Avendo quindi il comando FORMAT, esso dovrebbe essere «inserito» nella ROM prima di FOR, altrimenti FORMAT diventerebbe il token di FOR seguito da 'mat' causando un SYNTAX ERROR perché non rispettata la sintassi del FOR-NEXT in cui crederebbe di essersi imbattuto l'interprete.

Ovviamente questo «inserimento» non è possibile: o si riscrive la keyword table inserendo i nuovi comandi, o si prendono i dovuti accorgimenti per tokenizzare i nuovi comandi.

La prima ipotesi è assai fastidiosa da realizzare per vari motivi: il primo è dovuto a sua grazia il 6510 che ha grossi problemi con tavole di oltre 255 byte (spero abbiate notato che $a09e-$a19d sono 255 byte...); un secondo motivo è che così facendo bisogna riscrivere la routine di tokenizzazione (invece io l'ho copiata pari pari cambiando poco più che alcuni indirizzi); un terzo è che si occupa

Valore testo dell'errore di ERRN

1	Too many files	22	Type mismatch
2	File open	23	String too long
3	File not open	24	File data
4	File not found	25	Formula too complex
5	Device not present	26	Can't continue
6	Not input file	27	Undef'd function
7	Not output file	28	Verify
8	Missing file name	29	Load
9	Illegal device number	30	Break
10	Next without for	31	Until without repeat
11	Syntax	32	Resume without error
12	Return without gosub	33	Error in trap mode
13	Out of data	34	Too many cif
14	Illegal quantity	35	Pop without gosub
15	Overflow	36	Pop without for
16	Out of memory	37	Can't help
17	Undef'd statement	38	Window buffer full
18	Bad subscript	39	Undef'd window
19	Redim'd array	40	Too many window
20	Division by zero	41	Celse without cif
21	Illegal direct	42	Endif without cif

L'elenco dei valori di ERRN associati ad ogni tipo di errore che il TWS Basic può segnalare.

Programmi d'esempio in TWS Basic.

Listato 3 - Move Window

```
100 INIT:HCLR #55:MODE1:SCREEN1,1,1,0:PRINTCHR$(14):MODE3
110 TEXT 0, 1,"      QUESTA E' UNA SEMPLICISSIMA      "
115 TEXT 0,3,"    DIMOSTRAZIONE DELLE POSSIBILITA'    "
120 TEXT 0, 5,"              DEL TWS BASIC.            "
125 TEXT 0, 8,"  POSTATI DOVE VUOI NELLO SCHERMO E   "
130 TEXT 0,10,"      PREMI IL FIRE:LA FINESTRA TI     "
135 TEXT 0,12,"        SEGUIRA' OVUNQUE ... !!        "
200 MODE1:XW=17:YW=20:W=6:H=4
210 WOPEN XW,YW,W,H:WGET:WCLR:FRAME:TEXT XW+1,YW+2,"TK ?"
220 REPEAT
230 REPEAT:MOUSE1:UNTIL SPRTY<200-H*8
235 WSWAP WINDOW,XW,YW
240 XW=INT(SPRTX/8):YW=INT(SPRTY/8)
245 WSWAP WINDOW,XW,YW
250 UNTIL 0
```

Listato 4 - Copy Window

```
100 INIT:HCLR #55:MODE1:SCREEN1,1,1,0:PRINTCHR$(14):MODE3
110 TEXT 0, 1,"      QUESTA E' UNA SEMPLICISSIMA      "
115 TEXT 0,3,"    DIMOSTRAZIONE DELLE POSSIBILITA'    "
120 TEXT 0, 5,"              DEL TWS BASIC.            "
125 TEXT 0, 8,"  POSTATI DOVE VUOI NELLO SCHERMO E   "
130 TEXT 0,10,"      PREMI IL FIRE:LA FINESTRA        "
135 TEXT 0,12,"     VERRA' COPIATA OVUNQUE ... !!     "
200 MODE1:XW=17:YW=20:W=6:H=4
210 WOPEN XW,YW,W,H:WCLR:FRAME:TEXT XW+1,YW+2,"TK ?":WGET
220 REPEAT
230 REPEAT:MOUSE1:UNTIL SPRTY<200-H*8
240 XN=INT(SPRTX/8):YN=INT(SPRTY/8)
245 WPUT WINDOW,XN,YN
250 UNTIL 0
```

una maggior quantità di memoria (ben 255 byte in più!!!).

Non resta che l'altra ipotesi di cui si parlava; in questo caso si ha in effetti una tokenizzazione in due passate: la prima normale del Basic 2.0, la seconda del TWS Basic, che però si trova a che fare con una linea per così dire parzialmente tokenizzata. La seconda keyword table conterrà delle parole con al loro interno dei token: FORMAT sarà memorizzato come $81, $4d, $41, $54.

Il separatore tra una keyword è l'altra è lo zero, e due zeri segnalano la fine della keyword table. Ma non finisce qui!!!

Questa specie di doppia tokenizzazione comporta dei problemi anche nel LIST: trovando il token di FORMAT, dovrà leggere tre caratteri (FOR) dalla keyword table in ROM ed i seguenti (MAT) dalla seconda keyword table. È difficile essere più chiari se non si ha familiarità con L.M. e con le routine della ROM.

Una volta superati (!!!) questi problemi si ha un interprete facilmente espandibile: basta aggiungere il nome del comando nella tavola e scrivere la routine che lo esegue effettivamente, ricordandosi che non si possono avere più di 127 comandi in tutto (è possibile superare anche questo scoglio ma non ho intenzione di sconvolgervi ulteriormente).

Ricapitoliamo cosa succede quando premiamo il RETURN: la linea viene messa nel buffer a $0200, viene chiamata la routine di tokenizzazione della ROM e poi quella del TWS Basic; se c'era un numero all'inizio, la linea viene inserita in memoria, altrimenti viene eseguita subito.

Tutto quello detto finora non dice nulla su come funziona la routine che esegue i vari comandi: essa deve riconoscere se sta trattando un comando standard od esteso (basta controllare il valore del token: da $80 a $cc sono del Basic 2.0) e dopo aver controllato che non sia una funzione, chiamare l'appropriata routine.

Dopo il THEN vi può essere qualsiasi comando, anche esteso (non serve ':' dopo il THEN come in Basic meno perfezionisti), e sono accettate senza problemi sequenze di ':' (i cosiddetti comandi vuoti).

Tutto questo è poi sensibilmente complicato dalla gestione degli errori: ogni volta che è chiamata, la routine di esecuzione dei comandi (vettore $0308) deve sapere se e come «maneggiare» un errore, controllando che non sia avvenuto nella routine di gestione degli errori.

Per quanto riguarda gli errori, bisogna dire che vi è un bug proprio nella loro gestione: esso però è dovuto alla Commodore; la causa è sostanzialmente la stessa, ma l'effetto è riscontrabile usando i comandi FN e VAL.

La routine fondamentale del Basic è quella all'indirizzo $0073 (CHRGET) (attraverso la quale l'interprete legge il testo Basic carattere per carattere) che punta con $7a/$7b all'ultimo carattere letto.

Le chiamate di funzioni, così, come il comando VAL, alterano «momentaneamente» il puntatore $7a/$7b per farlo puntare rispettivamente alla definizione della funzione, o alla stringa da VALutare; se avviene un errore durante l'esecuzione di questi due comandi, la gestione degli errori preserverà $7a/$7b per poter poi sapere da dove continuare con RESUME; ma come abbiamo visto, il puntatore punta (brutto, vero?), dopo l'errore, alla linea dopo quella che definisce la funzione o nel bel mezzo dell'area riservata alle stringhe (se va bene...).

È quindi necessario usare il RESUME 'LINE' per forzare il Basic ad aggiornare correttamente $7a/$7b alla linea specificata.

La grafica

Ogni sistema di coordinate grafico del TWS ha l'origine nell'angolo in alto a sinistra, e per snellire notevolmente l'uso della grafica ho reso possibile specificare (solo con i comandi a coordinate in pixel) coordinate fuori dello schermo ed anche negative, nel range −32768/+32767. Oltre, viene segnalato il messaggio «?ILLEGAL QUANTITY ERROR».

È quindi possibile disegnare linee con gli estremi singolarmente o entrambi fuori schermo, od anche cerchi col centro fuori schermo o con raggio maggiore di 200.

L'unico comando di una certa importanza che manca è il FILL, ma non ho trovato algoritmi soddisfacenti; è comunque una mancanza abbastanza sopportabile e che è parzialmente attenuata da HCLR con parametro (un WCLR con parametro è abbastanza fastidiosa da implementare a differenza di HCLR...).

La gestione delle window è molto semplice e consiste semplicemente nel memorizzare una specificata area di schermo in un buffer in memoria. Il primo problema da affrontare è stato proprio quello di cercare il buffer in un modo conveniente: alla fine ho scelto di adibire una parte della RAM del C64 solamente per memorizzare i dati delle window (esattamente la RAM che va da $c000 a $ffff); per fare questo ogni accesso alle window causa la disabilitazione delle ROM e degli I/O per qualche attimo (tra l'altro non sono riuscito a disabilitare la ROM del Kernal indipendentemente da quella del Basic: sono io che sbaglio, è il mio C64 che è particolare oppure è vero???).

La RAM necessaria è H*W*8+4 dove H è l'altezza in caratteri e W l'ampiezza in caratteri; il '+4' si spiega col fatto che insieme ai dati della window vengono memorizzate le dimensioni e l'header della window, costituita appunto da 4 byte che sono nell'ordine ROW, COL, WID e HGT. Dopo seguono i byte effettivi della immagine video. L'indirizzo dato da TOPBUF( ) è quello del primo dei 4 byte dell'header.

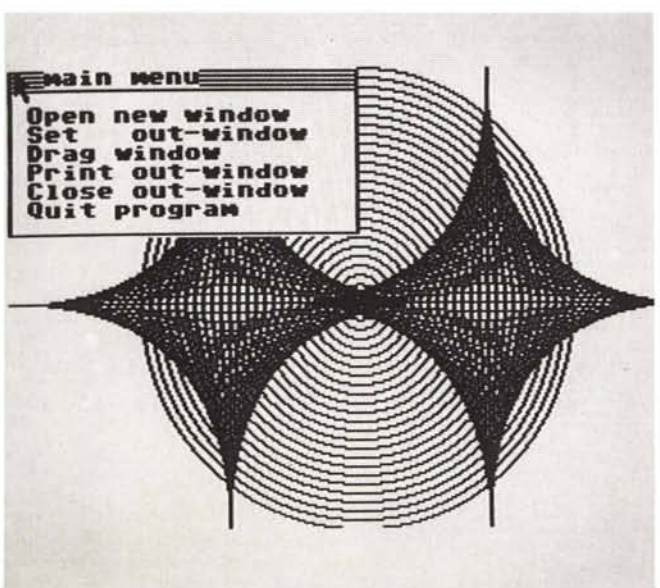
Ad ogni WOPEN viene riservata la zona di memoria necessaria, immediatamente dopo il buffer della window precedente.

La memorizzazione delle finestre è quindi del tutto sequenziale, e non vi è normalmente la possibilità di variare dimensioni e posizione di una singola window senza alterare le altre; scordatevi quindi cose come l'AMIGA o WINDOWS.

Comunque, pur nella loro struttura sostanzialmente sequenziale, è possibile «dinamicizzare» le window, come sarà evidente dalla sintassi dei vari WPUT, WGET, WCLR, WSWAP, WRAME che a differenza di WOPEN e WCLOSE permettono di specificare quale window utilizzare: bisogna però stare attenti a non farne un uso troppo libero, rendendosi sempre conto di quello che si sta facendo, per evitare effetti indesiderati.

I limiti d'uso delle window sono 2: non se ne possono definire più di 32 contemporaneamente, e per quanto riguarda la memoria, vi è a disposizione la RAM che va da $c000 a $ffff (quindi anche quella «sotto» gli I/O device e la Kernal ROM) riempiendola a partire da $c000 in su; ci sono cioè 16k a disposizione, che significa poter aprire al più 2 finestre grandi come lo schermo (o quasi). Nel caso che WOPEN richieda più memoria di quanta ne sia effettivamente disponibile in quel momento, verrà generato il messaggio «?WINDOW BUFFER FULL ERROR» senza però occupare la memoria libera.

L'apertura di una window non comporta alcun cambiamento del sistema di coordinate; purtroppo non mi è stato possibile realizzare una cosa del genere così come un FILL o altre cosette utili (AUTO, RENUMBER, MERGE...) perché la memoria del C64 è satura fino all'ultimo byte, e cercare ulteriore RAM avrebbe notevoli conseguenze su tutta la configurazione di memoria, a meno di rubare RAM utente riducendo i 30k a disposizione a una quantità minore. MC



Programma d'esempio in TWS Basic.