Viaggio all'interno del Basic

Come alterare (quasi) a piacimento i puntatori vitali dell'interprete Basic

(seconda e ultima parte)

di Alessandro de Simone

Secondo esperimento

Fase N.1
Spegnete e riaccendete il computer. Digitate, cosi come è pubblicato (senza alcuna modifica), il programma "Prova stringhe 1/a" privo della riga 180 in modo da evitare un "File not found error". Verificatelo e registratelo dopo aver aggiunto la riga 180.

```
100 rem prova stringhe 1/a
110 a$=" a. de simone "
120 b$="c.c.club"
130 c$= "somma "+a$+b$
140 print a$,b$;e$: print
150 rem riga di riempitivo
160 rem riga di riempitivo
170 rem riga di riempitivo
180 load"stringhe n.1/b",8
```

Fase N.2
Spegnete e riaccendete il computer. Digitate e registrate (col nome "Stringhe n.1/b"), dopo un opportuno controllo, il listato dal nome corrispondente avendo 1'accortezza di trascriverlo senza alcuna modifica. In particolare facciamo notare la coppia di doppi punti (::) presenti, dopo i REM, nelle righe 110 e 120.

```
100 rem prova stringhe 1/b
110 rem:: "attenzione!!!!"
120 rem:: "guarda!!"
130 print a$,b$,c$
```

Fase N.3
Spegnete e riaccendete il computer. Caricate e lanciate il programma "Prova stringhe 1/a" e lanciatelo: Sorpresa! Invece di visualizzare, in seguito al caricamento automatico di riga 180, ciò che ingenuamente ci saremmo aspettati (le stringhe A$ e B$ di riga 110 e 120 del *primo* programma), compaiono sullo schermo i caratteri delle righe 110 e 120 del *secondo* programma nonostante siano preceduti da due istruzioni REM. Che cosa e successo?

Esaminiamo con attenzione il listato 1/A. Le righe 110 e 120 definiscono una stringa, mentre la 130 effettua una concatenazione tra stringhe. L'interprete, di conseguenza, individuerà A$ e B$ mediante puntatori che punteranno all'interno del programma. Per C$, invece, sia i puntatori, che i caratteri costituenti C$, si troveranno al di fuori dell'area Basic.
Quando la riga 180 carica il secondo listato, tutti i puntatori, come già visto net primo esperimento, non vengono modificati, e se nel visualizzare C$ non sorgono problemi, ne sorgono, eccome, nel far apparire A$ e B$.
I puntatori di queste variabili, infatti, puntano ancora alle locazioni di inizio stringhe del primo programma che, dopo il caricamento del secondo listato, non contengono più i caratteri di riga 110 e 120, ma quelli situati dopo le REM del nuovo programma.
Ecco spiegato il perchè dell'insolita visualizzazione, come pure i1 motivo dei due caratteri di doppio punto situati dopo i REM: in questo modo i due programmi di figura 5 e 6 risultano (per ciò che riguarda le righe 100,110 e 120) perfettamente identici agli effetti del funzionamento dei puntatori.

Conclusioni sul secondo esperimento

1) 11 programma chiamante non deve contenere istruzioni del tipo: A$="NOME".
In caso contrario, se la variabile stringa viene interessata dal programma chiamato in overlay, nel migliore dei casi compaiono caratteri incomprensibili, nel peggiore possono verificarsi malfunzionamenti di tale entità da essere costretti ad interrompere 1'elaborazione.
2) E' pertanto necessario, purtroppo, esaminare riga per riga il programma chiamante e modificare tutte le variabili stringa definite nel modo A$="nome". Un suggerimento: 1'istruzione....

```
A$= "NOME"
```
...modificatela in...

A$= "" + “NOME” (vale a dire stringa nulla + “NOME”
...oppure...
READ A$: A$=“”+A$: DATA "NOME"

Terzo esperimento

Come caricare dapprima un programma breve e poi uno più lungo?
Alcuni testi suggeriscono di alterare i puntatori di inizio e fine Basic prima di effettuare l'operazione che, in effetti, è "proibita", dato che, come abbiamo visto, distrugge parte delle variabili già elaborate.
E' pero possibile effettuare 1'operazione in modo semplicissimo: supponiamo di voler utilizzare dapprima il programma "Breve" che, giunto alla riga 120, riceve l'ordine di caricare il programma "Lungo" che occupa un numero di byte decisamente superiore.
Ebbene, le fasi da seguire sono le seguenti:

1) Digitare il programma più breve e registrarlo col nome “Breve”.

100 rem programma breve
110 a=10: b=20: c=30
120 load “lungo”,8

2) Digitare 1'altro programma e registrarlo col nome "Lungo".

100 rem programma lungo
110 :
120 print a: print b: print c
130 :
140 rem queste righe
150 rem servono solo per
160 rem allungare il
170 rem listato
180: -
190 end: rem penultima riga
200 eseguire : run 210
210 load "breve",8

A questo punto (avendo cioè "Lungo" in memoria) digitare RUN 210. Questa operazione caricherà il programma "Breve" e lo renderà, come abbiamo avuto modo di studiare, di lunghezza eguale a "Lungo". Dopo 1'elaborazione verrà richiamato (riga 120) nuovamente il listato di figura 8 che risulta essere di lunghezza pari (ma guarda un po'...) al programma chiamante!

I puntatori di inizio e fine Basic

Alcuni (presunti) malfunzionamenti del programma "Append" pubblicato sul N.42.di C.C.C. (pagina 5) offrono il pretesto per parlare ancora di quattro locazioni fondamentali per il corretto funzionamento del sistema operativo.
Allo scopo di meglio comprendere gli argomenti trattati nel presente inserto, si suggerisce al lettore di rileggere attentamente quanto pubblicato nell'inserto dello scorso numero.

Introduzione

I computer Vic 20, C-16, Commodore 64 allocano i programmi Basic, digitati da tastiera o caricati da unità magnetiche, a partire dall'indirizzo contenuto nelle due locazioni di memoria 43 e 44.
Riferendoci al Commodore 64, in tali locazioni sono normalmente contenuti i valori "1" (in 43) e "8" (in 44). L'indirizzo (I) della prima locazione in cui verrà allocato il programma Basic sarà dato dal semplice calcolo:
*I=1 +8*256=2049*

...oppure da un più immediato...

*Print Peek(43)+ Peek(44)*256*

Per far funzionare correttamente un programma Basic è però necessario che il byte precedente tale locazione sia posto a zero; all'annullamento di tale byte provvede automaticamente il computer al momento dell'accensione.
Ne potete avere facilmente conferma chiedendo, mediante Peek, il valore ivi contenuto.
Provando, infatti, a digitare:

*Poke Peek(43)+Peek(44)*256-1;1*

Oppure, più semplicemente (ma è valido, ovviamente, per il solo C-64):

Poke 2048,1

noterete che alcuni comandi (NEW, RUN e altri) non vengono riconosciuti e viene emessa la segnalazione Syntax Error.
Deriva, pertanto, la...

Prima regola

La locazione precedente un programma Basic deve SEMPRE contenere il valore nullo; qualsiasi altro valore provoca infatti inconvenienti di varia natura.
Inoltre...

Seconda regola

L'indirizzo della locazione di inizio Basic è data dal prodotto del contenuto di 43 sommato al contenuto di 44 moltiplicato per 256.
Omettendo tale verifica, di cui si è ampiamente parlato nel1'inserto precedente, si possono verificare guai di vario tipo, per cui insistiamo net ricordare la...

Terza regola

Ogni linea Basic e formata da:

• due byte di Link (= puntatori al prossimo byte di memoria Ram che rappresenta 1'inizio della successiva linea Basic).

• due byte che rappresentano la numerazione della riga Basic stessa.

• un gruppo di (al massimo) 80 caratteri costituenti istruzioni, comandi e dati della linea Basic.

• un ultimo byte nullo. L'indirizzo di questo risulta quindi inferiore di un'unità al valore risultante dal calcolo dei puntatori di Link.
Se, ad esempio, la coppia di Link dovesse puntare alla locazione 10321, il byte nullo sarà rintracciabile all'indirizzo 10320.

Quarta regola

L'ultima linea di un programma Basic termina, invece che con un byte nullo, con tre byte nulli, in successione.

Quinta regola

L'indirizzo (I) dell'ultima locazione riservata al programma Basic è anche data dal calcolo dei puntatori 45 e 46:

*I= Peek(45)+ Peek(46)*256-1*

Tale indirizzo è l'ultimo byte nullo (dei tre) di cui si è parlato poc'anzi.
Le didascalie di figura 1 dovrebbero eliminare ogni incertezza.

Uno strano doppione

Da ciò che abbiamo detto sembrerebbe che il calcolatore dispone di due "sistemi" per individuare la fine di un programma:

• Indirizzo del byte puntato da 45 e 46.
• Tre byte nulli successivi.

In realtà 1'interprete Basic del computer utilizza il primo sistema in alcuni casi ed il secondo in altri.
Quando il programma viene normalmente eseguito (e, in particolare, vengono utilizzate le istruzioni RUN, GOTO, GOSUB, ed altre), il calcolatore provvede a rintracciare la linea giusta iniziando a spulciare 1'intero programma a partire dalla linea puntata da 43 e 44 e FERMANDOSI o nel caso in cui incontra l'indirizzo cercato (ed eseguendo le istruzioni ivi contenute), oppure nel caso in cui individua tre byte nulli in successione (ed emettendo, se del caso, un "Undefined Statement error").
Se, invece, si registra un programma su supporto magnetico, il sistema operativo provvederà a trasferire su disco (o nastro) il contenuto di tutti i byte compresi tra i due indirizzi puntati da 43 e 44 (=inizio) e 45 e 46 (=fine).
Le didascalie della figura 2, ed i due programmi pubblicati, chiariscono meglio quanto esposto.
L'alterazione dei quattro puntatori (43, 44, 45, 46) è la tecnica normalmente adoperata per il caricamento ed il salvataggio di programmi in Linguaggio Macchina.
Un commento a parte meritano i programmi pubblicati, scritti per un Commodore 64:

```
100 a fre(0)
105 if a<0 then a=2        16-abs(a)
110 print a: end
120 poke 46,8: c1r: goto100
130 poke 46,9: c1r: goto100
140 :
150 rem programma n.1
```

I1 primo, decisamente breve, occupa un numero di byte che è possibile calcolare in modo indiretto mediante FRE (0).
Le prime tre righe non fanno altro che calcolare, e visualizzare, il numero di byte RAM rimasti liberi.
Lo stesso risultato si ottiene digitando Run 120.
Con Run 130, invece, l'alterazione "forzata" del byte 46 sottrae artificialmente memoria alla RAM ed il risultato è diverso.
E' ovvio che il programma sembrerà più lungo del precedente, ma solo nel caso di registrazione (e successivo caricamento); al limite sarà possibile ottenere un messaggio di "Out Of Memory Error" nel caso si tentasse di digitare altre righe.
Se, infatti, modificate la riga 130 come segue...

```
130 poke 46,159: c1r: goto100
```

...e tentate di aggiungere altre righe Basic (anche se "piene" di semplici Rem), come ad esempio....

```
200 rem aaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa
210 rem bbbbbbbbbb bbbbbbbbbb bbbbbbbbbb bbbbbbbbbb bbbbbbbbbb
```

... vi accorgerete motto presto di non riuscire ad aggiungere altre righe.

Se poi, addirittura, modificate come segue...

130 poke 46,160: c1r : goto100

...otterrete subito un Out of memory con un semplice Run 120 ed il motivo è semplice da spiegare: il prodotto 160*256 indica una locazione il cui indirizzo (40960) è situato oltre quello consentito dal Basic.
Nel caso in cui non si sia verificato un Out of Memory, invece, la fine del programma verrà sempre individuata dall'interprete, solo dalla presenza di tre byte nulli che sono ancora al loro posto e che non sono stati alterati minimamente dalle operazioni descritte in precedenza.
Il secondo listato altera uno dei due puntatori (i lettori più pignoli possono alterare anche quello contenuto nella locazione 45, in modo da puntare esattamente all'indirizzo desiderato).

```
100 print fre(0): rem verifica memoria prima
110 poke 46,15: clr: restore
120 print fre(0): rem verifica dopo
130 get a$- ifa$= “” then 130
140 f$= "commodore computer club ": rem 23 caratteri
150 for i=1 to 23
160 poke IS*256+i, asc (mid$(f$ i,1)): next
170 print chr$(14): rem maiuscolo minuscolo
180 clr: restore: print chr$(147)
190 for i=1 to 23
200 poke 1024+i, peek(15*256+i): next
210 :
220 rem programma n.2
```

Il listato agisce in modo che il puntatore di fine Basic punti OLTRE il necessario e deposita, in questa zona, i caratteri delta stringa F$ (cioè: "Commodore Computer Club"). In tal modo sarà possibile avere a disposizione una zona di memoria *inattaccabile* dal Basic in cui potrete trascrivere (ricorrendo al semplice algoritmo di righe 110-120) un messaggio di qualsiasi lunghezza.
Per rendervi conto del suo funzionamento, trascrivete il programma N.2, così come è pubblicato, e registratelo dopo averlo fatto girare almeno una volta.
Spegnete pure il computer, riaccendetelo e ricaricate il programma. Digitate RUN 170. Vi accorgete che il messaggio viene egualmente visualizzato (a dispetto di Run e CLR che dovrebbero cancellare ogni variabile senza pietà), dal momento che è rimasto "incorporato", al momento della registrazione, nel programma stesso!
Tale tecnica viene normalmente adoperata per salvare, insieme al programma Basic, brevi listati in linguaggio macchina, oppure il nome degli autori dei programmi (in modo da renderli "indelebili") o per tecniche di protezione.

Malfunzionamenti di append

Da quanto esposto dovrebbe risultare chiaro che la tecnica di Append, descritta nel N.42, non fa altro che seguire la procedura descritta nelle didascalie di figura 3.
Come mai, dunque, in alcuni casi il programma non funziona?
E' presto detto: sono molto diffusi, tra gli appassionati di computer, particolari utility (renumber, delete ed altre) il cui funzionamento non soddisfa pienamente la quarta e quinta regola precedentemente esposte.
In altre parole tali utility, al termine del loro lavoro, non fanno più coincidere 1'indirizzo del byte successivo ai tre byte nulli (di fine programma) con 1'indirizzo dei puntatori 45 e 46. In genere, con tali utility, i puntatori 45 e 46 puntano qualche byte più "in la" dell'ultimo byte nullo col risultato, disastroso, di alterare completamente il sistema di linkaggio del programma Slave caricato in Append.
Il programma 3 risulta utile per individuare i programmi che possono generare malfunzionamenti in fase di Append.

```
63994 a1 = peek(43) +peek(44)*256: a2 = peek(45) +peek(46)*256
63995 for i=a1 to a2: if peek(i)=0 and peek(i+1) =0 and peek(i+2)=0 then 63997
```

```
63996 next
63997 il =(i+3)/256: i2=int(il): il =(i1- i2)*256
63998 print chr$(147)"poke45, "il ": poke 46, "i2": crl: restore"
63999 rem programma n.3
```

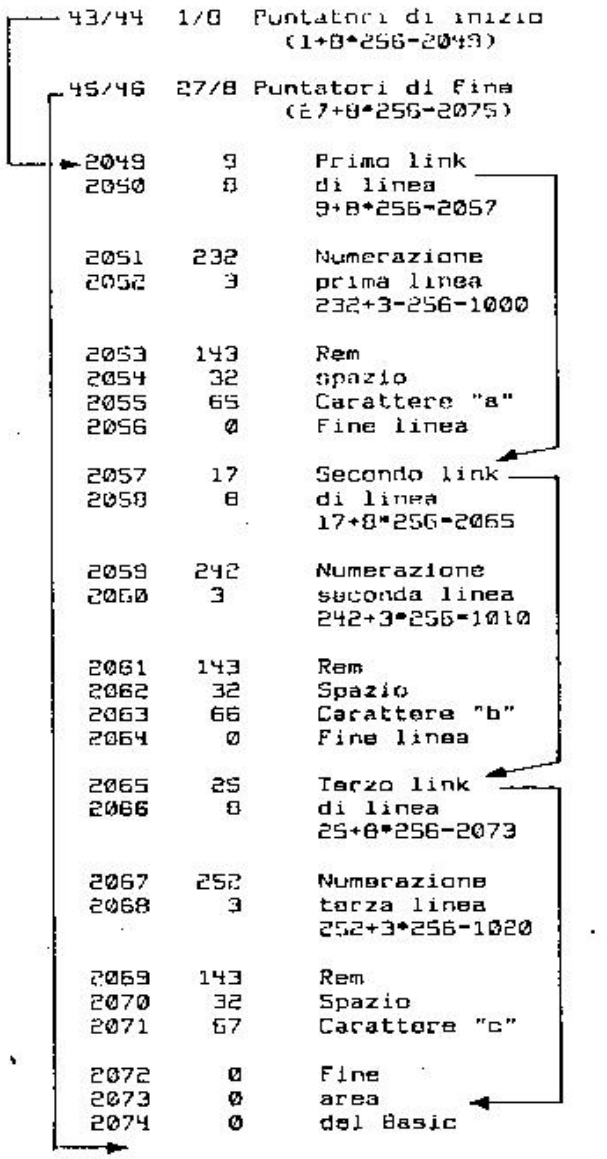
Come si può notare. il listato è numerato con gli ultimi numeri disponibili con un C/64: tentando. infatti, di digitare...

```
64000 Rem
```

...si ottiene un Syntax Error:
Tale numerazione consentirà al lettore di inserire il listato stesso "in coda" ad un qualsiasi listato Basic, a patto, ovviamente, che non possegga eguali numeri di linea.
In conclusione:
• Il programma Append funziona correttamente a patto che almeno il programma Master sia "ortodosso". Questo, in altre parole, deve avere coincidenti i puntatori di fine Basic (45 e 46) con l'indirizzo successivo all'ultimo dei tre byte nulli.
• Nel caso in cui la fusione non avvenga, vuol dire che il programma Master non è ortodosso. In tal caso:
• Digitare il breve listato N.3 di queste pagine (o caricarlo da nastro o disco), e visualizzatelo con un semplice List.
• Mentre tale programma è ancora visualizzato sullo schermo, caricate il programma Master non ortodosso facendo in modo, ovviamente, che lo schermo contenga ancora il breve listato N.3.
• Terminato il caricamento risalite col cursore (senza cancellare lo schermo!) e premete il tasto Return su ciascuna linea Basic del programma 3: in tal modo esso verra incolonnato in fondo al Master.
• Fate partire il programma con RUN 63994. Dopo un po' di tempo, proporzionale alla lunghezza del programma (a volte parecchie decine di secondi), verrà visualizzato un rigo contenente quattro istruzioni (vedi riga 63998). Posizionatevi sopra con il cursore e premete il tasto Return: in questo modo i puntatori 45 e 46 punteranno come di dovere.
• Registrate il programma cosi modificato, magari dopo aver cancellato le ultime righe (che confluiscono il programma N.3).
• Il programma, così registrato, è ora utilizzabile come programma Master in fusioni di append.

```
100 Rem append
130 Rem per C164 e drive Commodore
140 print chr$(147) "nome prog.master":input x$
150 if len(x$)>16 then 140
160 print: print: print "nome prog.slave": input y$
170 if len(y$)>16 then 160
180 printchr$(147): fori = 1tol9: print: next
190 print "poke43, "peek(43) chr$(157)":poke44, "peek(44);
200 printchr$(19)"load" chr$(34) x$ chr$(34) ",8"
210 printchr$(19): fori=lto4: print: next
220 print "a=peek(43): b=peek(44):";
230 print "c=256*peek(46) +peek(45)-2": print: print
240 print "poke43,c and 255: poke44,int(c/256): new"
250 print: print: rem alessandro de simone
260 print "load" chr$(34) y$ chr$(34) ",8"
270 poke 198,10
280 for i=1 to 10: read a: poke 630+i,a: next
290 data 19,13,13,13,13,13,13,13,13,13,13
```

Figura 1

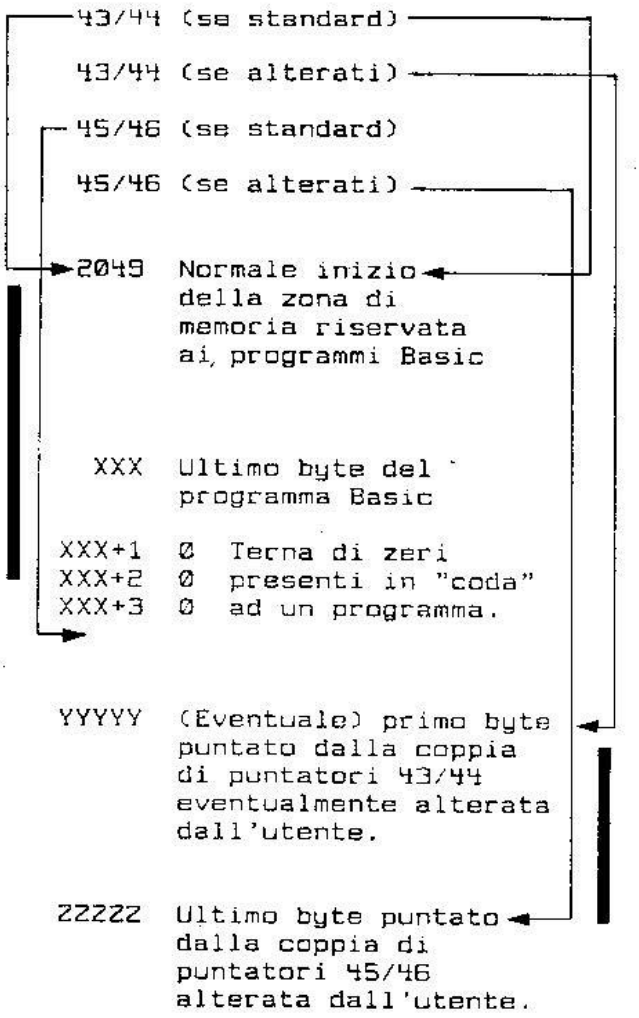


Zona Basic coincidente con zona Load e Save: A mano a mano che vengono digitate le linee Basic, i due puntatori 45 e 46 vengono automaticamente aggiornati. Analogamente una routine provvede ad incorporare nella RAM la nuova fines Basic e ad inserire tre zeri, in successione, in coda ad essa (a patto che sia 1'ultima, come numerazione). Si noti come il numero di linea Basic, qualunque esso sia, occupa sempre due byte come, pure i due byte di link. Ogni linea Basic termina sempre con uno zero. Nella figura, per motivi di sempficità, è rappresentato il banale listato...

1000 Rem a
1010 Rem b
1020 Rem c

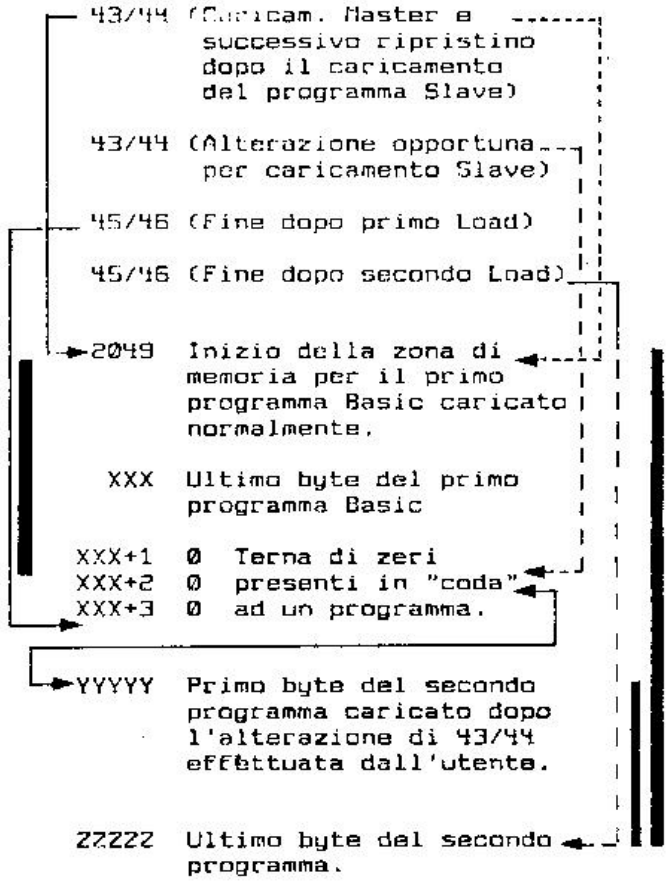
...nel caso sia digitato su un C/64.

Figura 2



Manipolando opportunamente i byte 45 e 46 (=fine Basic) è possibile alterare le informazioni relative alla zona RAM interessata da Load e Save che, in caso contrario, coinciderebbe con la zona contenente il solo programma Basic. Nel caso illustrato in figura è possibile salvare (registrare) mediante Save, non solo il programma Basic ma anche la zona di RAM presente dopo di esso che può contenere eventualmente messaggi, variabili e, addirittura, programmi L.M. o sistemi di protezione.

Figura 3



Funzionamento dell'utility Append:
Fase 1: Accensione del computer e caricamento del programma Master. L'aggiomamento dei puntatori di fine Basic è automatico (frecce scure).
Fase 2: L'utente, altera i puntatori di inizio Basic in modo che "puntino" allo stesso indirizzo di fine Basic (45-46) meno due byte (a causa dei tre zeri successivi). (freccia tratteggiata in basso).
Fase 3: Il caricamento, automatico, "appende" in coda al Master, il programma desiderato (freccia tratteggiata in alto).
Fase 4: Terminato il caricamento l'utente ripristina i precedenti valori di inizio Basic (freccia punteggiata).
L'intera procedura è valida solo nel caso in cui il programma Basic sia ortodosso, simile, in altre parole, a quello di figura 1 e non di figura 2.

Oltre i puntatori

Vedremo, ora, di sfruttare la possibilità di alterare a piacimento i puntatori del Basic per realizzare insolite procedure.
Tra le varie applicazioni esamineremo in dettaglio la coesistenza di programmi Basic, l'utilizzo dell'area di schermo per allocare programmi, e la conseguente estrapolazione di tecniche insolite di protezione.
Su alcuni aspetti ci soffermeremo peù a lungo di altri, anche per non privare i nostri lettori del piacere (!) di scoprire, da soli, altre possibilità.
Verrà ora descritta una tecnica, utile in più di un caso, che consente di "segmentare" a piacere la memoria Basic a disposizione dell'utente.

La segmentazione dell'area Basic

Abbiamo, dunque, imparato che un programma Basic, per il computer, è allocato a partire dalla locazione data da...

*Peek(43)+ Peek(44)*256*

...fino alla...

*Peek(45)+ Peek(46)*256*

... a patto, come ricorderete, che la locazione precedente l'inizio contenga un valore nullo.
Di ciò possiamo chiedere conferma con un semplice:

Print Peek(43); Peek(44)

che risponde con 1 e 8 (caso di un C/64 all'accensione). Alterando i puntatori con...

*Poke 44,10: Poke (10*256+1)-1,0: New*

"costringiamo" l'interprete Basic a riconoscere, come spostata in avanti, l'area destinata ai programmi; in altre parole la prima locazione destinata al Basic sarà posizionata non più a partire da 2049 (dato da 8*256+1), ma da 2561 (=10*256+1). Con tale sistema abbiamo, come si dice in gergo, "spostato avanti il Basic con la conseguenza di riservare 512 byte (2561- 2049) per i nostri scopi più vari.
Potremo, tra l'altro, allocare routine in linguaggio macchina, (mezze) schermate, messaggi e, perchè no?, un altro programma Basic. Vedremo, una per una, le varie applicazioni.

Memorizza messaggi

Abbiamo già visto come "incorporate" un messaggio "in coda" ad un programma Basic; la tecnica descritta, però, impedisce un'eventuale, successiva, aggiunta di righe Basic, o la loro modifica, senza modificare anche il messaggio stesso, se non ricorrendo a complicate tecniche di puntamento.
Vi consigliamo di seguire alla lettera, come al solito, le note che seguono: è l'unico modo di non incorrere in errori:

• spegnete e riaccendete il C/64
• digitate:

Poke 44,10: Poke 2560, 0: New

In questo modo sposterete l'inizio del Basic, come già detto, dall'originario 2049 a 2561.
• digitate il seguente programma:

100 print "digita un messaggio"

```
110 input a$: x=len(a$)
120 poke 2050,x
130 for i=1 to x
140 y$=mid$(a$,i,1)
150 poke 2050+i, asc(y$)
160 next
170 :
180 for i=1 to peek(2050)
190 y$ = chr$(peek(2050+i))
200 print y$;: next
```

Commentiamolo brevemente:
- Le righe 100/160 allocano da 2051 in poi, uno per uno, i caratteri che costituiscono la stringa A$, da digitare al momento dell'Input (rip 110). Si noti che la locazione 2050 servirà, in seguito, per individuare la lunghezza del messaggio stesso.
- Le righe 180/200 operano in modo inverso: estraggono dalla memoria Ram i valori precedentemente allocati, e ne stampano i codici Ascii.

• Verificato il corretto funzionamento (la stringa digitata viene visualizzata sullo schermo), digitate quanto segue:

Poke 44,8: Clr

In tal modo, infatti, si ripristinano i puntatori standard del Basic. Non consigliamo, a questo punto, di listare il programma nè, tantomeno, di modificarlo! I codici da noi inseriti, presenti da 2049, non hanno alcun significato per l'interprete, e correte il rischio di dover spegnere il computer.
Provvedete, invece, a registrare il programma con un banale:

Save "Programma",8

...oppure...

Save "Programma"

... se non disponete del drive, ma solo del registratore.
Con la modifica della locazione 44, e con la successiva operazione di registrazione, non abbiamo fatto altro che riportare, su supporto magnetico, "tutto" ciò che è presente in memoria Ram dalla locazione 2049 fino al valore puntato dalla coppia di locazioni 45 e 46; queste, dal momento che non sono state modificate, coincidono con la fine del nostro listato Basic.

• spegnete e riaccendete il C/64
• digitate:

Poke 44,10.- Poke 2560, 0: New

ripetendo, in partica, l'ordine visto in preccdenza.
• caricate il programma di prima con il suffisso ".1" che obbliga il computer, come è noto, ad allocare il programma in oggetto nelle stesse identiche locazioni in cui si trovava al momento della registrazione:

Load "programma",8,1

Se dimenticate il suffisso ",1" sarete costretti a spegnere e ricominciare daccapo.

• Digitate:

Run 180

il messaggio, memorizzato nella prima fase, verrà ora visualizzato, dimostrando, in tal modo, che lo stesso messaggio è stato memorizzato insieme al programma al momento della registrazione.
• Provate ad aggiungere righe, modificarle, cancellarle: tutto funzionerà come di consueto ed un Run 180 darà sempre gli stessi risultati.
E' intuitivo che è possibile allocare routine in linguaggio macchina nella zona di memoria precedente al Basic "ufficiale"; la loro lunghezza potrà essere qualunque, a patto di inserire, nella locazione 44, il giusto valore.
Va da se che, nei successivi caricamenti, e doveroso ricordarsi di alterare la locazione 44 (ed eventualmcnte la 43), e di annullare il byte precedente la locazione puntata; ciò va eseguito PRIMA di provvedere al caricamento stesso.
Analogamente, al momento della registrazione, è di vitale importanza ripristinare il valore standard della locazione 44.

Due programmi in memoria

Sofisticando la tecnica descritta è possibile avere in memoria due (o più) programmi Basic, ognuno indipendente dall'altro. Provate a seguirci nell'interessante esperimento:

• spegnete e riaccendete il C/64.
• digitate con la massima attenzione (soprattutto la riga 130) il seguente mini-programma:

```
100 input “digita tre numeri”: x, y, z
110 print "i numeri sono: " x; y; z
120 print "La stringa è:" a$
130 poke 44,10: poke2560, 0: goto 100
```

Non appena darete il Run, verrà chiesto di digitare tre numeri; rispondete digitando tre valori qualunque, separati da una virgola.
Subito dopo (riga 110) verrà data conferma dell'avvenuta memorizzazione dei tre numeri e comparirà un messaggio apparentemente strano ("La stringa è:") relativo ad una stringa (A$) nulla, dal momento che abbiamo impartito un Run.
Ma cio che è più grave è sicuramente la presenza del messaggio "Undef d Statment Error in 130".
A questo punto non tentate nemmeno lontanamente di chiedere un List, ma prowedete ad impartire il comando: New.'
* Digitate, ora, il seguente listato:

```
100 input “digita la stringa ”; a$
110 print  “i numeri sono:” x; y; z
120 poke 44,8: goto 100
```

...e solo alla fine, dopo aver verificato la corretta trascrizione, date il Run. Possiamo, ora, capire ciò che accade.
I1 primo programma non fa altro che chiedere tre numeri e associarli alle variabili X, Y e Z; subito dopo stampa il contenuto di A$ (che, per ora, non c'è).
L'ultima riga del primo listato impone la modifica del puntatore di inizio Basic (per semplicità non è stato considerato anche 43, ma i più pignoli, volendo...).
II Goto 100 (vedi riga 130), nonostante l'impostazione della nuova "mappa" della memoria, "deve" essere interpretato, perchè rappresenta l'ultima istruzione della riga Basic attivata. In questo momento, però, l'interprete Basic è indotto a rintracciare la riga 100 e si prepara a farlo considerando l'area di memoria ram in cui ritiene "esistente" i1 programma basic! Questa, però, non e più la stessa di prima (a causa del Poke 44,10) e ne consegue la visualizzazione del messaggio di errore.
Quando, poi, digitate il secondo listato, questo viene allocato a partire da 2561 e attivato, alla fine, con un semplice Run.. In questo caso, però, giunto ad elaborare la riga 120 (dopo aver spostato la mappa della memoria al valore originario), trova davvero "una" riga numerata con 100, ma non è più quella del secondo programma, bensì quella del primo.
Quest'ultimo, infine, non trovera più contraddizione giunto alla riga 130 perchè, stavolta, la riga 100 (pur se del secondo programma) "esiste" realmente.

La corretta visualizzazione delle quattro variabili interessate (X, Y, Z, A$) dimostra che, nei passaggi da un banco di memoria ad un altro, queste non vengono affette da alcuna variazione.
II discorso seguito finora, come è intuitivo, può essere applicato a tre, quattro e, almeno in teoria, a un numero qualsiasi di programmi Basic, mutuamente richiamabili tra loro.
In pratica, però, il discorso si complica terribilmente: non meravigliatevi, quindi, se durante i vostri tentativi il computer si blocca senza rimedio.
Allo stesso modo una minima disattenzione nel registrare (o caricare) programmi scritti nel modo suggeriro può provocare vere catastrofi.
Il lettore, ad ogni modo, non tema di danneggiare il proprio calcolatore: nessuna operazione di Poke, per quanto ardita sia, riuscirà mai a provocar danni (se non la perdita di tempo del digitare nuovamente i vari programmi).
È superfluo ricordare che l'imposizione di Poke 44,10 è valida a patto di accontentarsi, per il primo programma, di soli 512 byte; per programmi più lunghi agire di conseguenza sulla locazione 44.

Sbatti il programma sullo schermo

Ciò che ora faremo servirà per meglio comprendere non solo il modo di operare dell'interprete Basic all'interno dell'area ad esso destinata, ma anche per suggerire nuove, inedite procedure software che possano costituire una valida base per intraprendere, tra 1'altro, lo studio di inediti sistemi di protezione.
Per meglio comprendere 1'argomento, e per consentire anche ai principianti di seguire il discorso, saremo costretti a riprendere argomenti forse già noti, sui quali, peraltro, promettiamo di intrattenerci il minimo indispensabile.

Una premessa

L'area di schermo è costituita da una successione di 1000 byte (25 righe * 40 colonne, caso del C/64), il cui indirizzo iniziale è 1024.
E' possibile scrivere caratteri sullo schermo non solo con istruzioni del tipo Print, ma anche con Poke e, in seguito, leggerne i codici con istruzioni Peek.

• Spegnete e riaccendete il C/64.
• Cancellate to schermo (Shift + Clr/Home).
• Premete tre volte il tasto Return (il cursore, quindi, dovrebbe lampeggiare sul terzo rigo).
• Digitate: Poke 1024,1
• Nella prima cella in alto a sinistra dovrebbe apparire il carattere maiuscolo "A". Se ciò non avviene significa che il vostro C/64 è un modello dotato di una versione particolare di Rom che necessita della colorazione della cella video. Provate, in questo caso, con:

Poke 1024,0: Poke 55296,1

La memoria colore, come è noto, occupa le 1000 celle numerate da 55296 a 56295.
• Cancellate lo schermo e, rimanendo sulla prima cella video in alto a sinistra, battete il tasto "B" e scendete (SENZA premere il tasto Return, ma ricorrendo ai tasti cursore) di qualche riga, in modo da poter digitare liberamente:

Print Peek(1024)

• Otterrete il valore 2. Ciò significa che ad ogni cella video è possibile associare uno dei 256 valori (numerati da 0 a 255) e che, viceversa, ad ogni carattere visualizzabile corrisponde uno dei 256 valori possibili.

Prima di proseguire ricordiamo che è possibile passare dal set maiuscolo-grafico a quello maiuscolo-minuscolo premendo i tasti Commodore e Shift. Lo stesso effetto si può ottenere con:

Print Chr$(14)

...per passare da maiuscolo-grafico a maiuscolo-minuscolo e con...

Print Chr$(142)

...per ritornare al maiuscolo-grafico.
Per fare in modo da rendere inefficiente la pressione contemporanea dei tasti Commodore e Shift, digitate:

Print Chr$(14); Chr$(8)

In questo modo non solo si imposta il set minuscolo, ma si impedisce di ritornare all'altro set premendo inavvertitamente i due tasti Shift e Commodore, a meno di premere Run/Stop e Restore, oppure di battere:

Print Chr$(9)

Alteriamo i puntatori

Se la vostra mente non è labile, dovreste ricordare che è possibile "spostare" 1'area di memoria dedicata al Basic, limitandosi ad alterare i suoi puntatori di inizio (e magari, come vedremo tra breve, di fine).
Poichè l'area di schermo è una zona Ram come qualsiasi altra zona, faremo in modo che 1'interprete Basic consideri proprio 1'area video come zona in cui allocare eventuali programmi Basic!
Da questo momento è indispensabile seguire alla lettera, più che mai, le note che esporremo: un MINIMO errore vi costringerà a spegnere, riaccendere e ricominciare daccapo.

• Spegnete e riaccendete il C/64.
• Digitate...:

Print Chr$(14); Chr$(8)

...e cancellate to schermo (Shift + Clr/Home).
• Premete dieci volte il tasto Return (non una di più, non una di meno).
• Battete, su una sola riga, i seguenti comandi:

Poke 44,4: Poke 1024,0: New

Chi, invece, dispone di un C/64 da "colorare", dovrà digitare alcuni comandi in più:

For i=55296 to 56295: Poke i,1: Next: Poke 44,4: Poke 1024,0: New

Immediatamente compariranno, in alto a sinistra, tre caratteri di "chiocciolina" (simbolo corrispondente al tasto situato tra 1'asterisco e "P"). Che cosa è successo?
Il primo comando (Poke 44,4) sposta il Basic a partire da 4*256, cioè da 1024 che coincide, come abbiamo visto prima, proprio con l'inizio dell'area video.
II successivo Poke 1024,0 è la "solita" locazione nulla che deve precedere 1'inizio del Basic.
L'ultimo comando (New) ci evita di mettere a posto manualmente i rimamenti puntatori e, in pratica, obbliga il computer a sistemare tre zeri in successione all'inizio dell'area destinata al Basic (che coincide, lo ripetiamo, con 1'area video).
Inutile dire che la chiocciolina rappresenta il codice zero nella rappresentazione delle Poke dello schermo!
Da questo momento è assolutamente necessario rendersi conto che non possiamo pasticciare sul video come siamo abituati di solito: dobbiamo tener presente, che ora, ad esempio, 1a cancellazione del video coincide con una cancellazione ben più grave.
Per digitare i vari comandi che suggeriremo tra breve, saremo costretti ad utilizzare sempre la stessa riga (la decima) per evitare disastrosi scrolling del video o, peggio, la comparsa di messaggi di errore che "sporcherebbero" lo schermo (e quindi, involontariamente, 1'area del Basic).
In altre parole, tutti i comandi che indicheremo, dovrete digitarli sulla decima riga di schermo, armandosi di pazienza e salendo con il cursore su tale linea, cancellando i comandi precedenti (mediante la barra spaziatrice o il tasto Del) e, soprattutto, stando bene attenti ad evitare errori di battitura.
Per assicurarsi di essere sulla decima riga di schermo potete, fortunatamente, premere il tasto Home (e non Shift + Home) e premere disci volte il tasto cursore in basso.
Abbiamo, insomma, attribuito alle 1000 locazioni di schermo l'ingrato compito di fungere, contemporaneamente, sia da area video che da area basic, con la conseguenza di dover rispettare tutte lc esigenze di una doppia, delicata e, soprattutto, pesante responsabilità.

Le variabili

Abbiamo detto che è necessario stare attenti a non cancellare lo schermo per evitare guai. In effetti possiamo ripristinare le condizioni iniziali cancellando lo schermo con i tasti Shift e Clr/Home e digitando in successione, nelle prime tre celle video, altrettanti caratteri di chiocciolina.
Tale modo di comportarsi corrisponderà alla cancellazione del programma Basic eventualmente presente in memoria e avrà (quasi) lo stesso effetto di un New.
Supponendo, dunque, di aver impostato il modo maiuscolo-minuscolo, di aver spostato 1'area Basic a partire dall'inizio del video, di avere lo schermo "pulito" (tranne le tre chioccioline in alto a sinistra) e di veder lampeggiare il cursore sulla decima riga, digitate il seguente comando:

AA=0

che associa, alla variabile in virgola mobile "AA", il valore nullo.
Vedrete subito una piccola rivoluzione in alto a sinistra dello stesso schermo:
• le tre chioccioline sono rimaste allo stesso posto (non vi sono ancora, infatti, programmi Basic).
• Subito dopo si notano i due caratteri "AA" che stanno a rappresentare il nome della prima varabile dichiarata (vedi inserto del N.43) che è anche l'unica.
• Dopo "AA" sono presenti cinque locazioni contenenti altrettante chioccioline che rappresentano il valore nullo. Queste cambieranno a seconda del valore successivamente associato ad "AA".

Salite, ora, con il cursore sulla riga che contiene AA=0 e modificatene il valore a piacimento (AA =123 AA =2.45 AA=123.67): dopo ogni pressione del tasto Return noterete 1'immediata modifica delle cinque locazioni poste dopo "AA" in alto sul video. Ciò è perfettamente in linea con quanto studiato nel primo inserto.
Naturalmente, ma agendo con 1a massima prudenza, potete posizionarvi, agendo con i tasti cursore, alla destra di "AA" (primo rigo del video), e digitare, a casaccio, vari caratteri (magari anche in reverse).
In seguito, sempre con i tasti cursore, posizionatevi sulla solita decima riga e impartite Print AA per verificare il valore che corrisponde a ciò che avete battuto.

Per esernpio, se dopo AA (ci riferiamo ancora al primo rigo di schermo), battete "aaaaa" (cinque "a" minuscole in successione) otterrete, come risposta a Print AA, il valore 5.9696674e-38. Con "bbbbb", invece, il valore: 5.9696674e-39 e così via.

Non nominare invano le variabili

Facciamo, ora, una nuova esperienza:
• Cancellate lo schermo (Shift + Clr/Home)
• digitate tre chioccioline in successione (equivale, come visto, a un New) e scendete, con i tasti cursore, sul decimo rigo.
• Battete il comando CLR, che annulla le variabili; tale comando è indispensabile. Provate, infatti, a farne a meno... (siete in grado di spiegarvi il motivo del successivo, strano comportamento?).
• digitate, sempre sul decimo rigo, il comando di prima:
AA=0
• Chiedete, ora, di stampare il contenuto della variabile BB:

Print BB

La risposta, ovviamente, sarà zero dal momento che non abbiamo dichiarato tale variabile in precedenza. Guardate in alto a sinistra: tutto è come prima.
Ma provate, ora, a commettere un errore di sintassi. Digitate, ad esempio, BB e premete il tasto Return: oltre al Syntax Error (che ci aspettiamo), qualcosa è accaduto in alto, sullo schermo: subito dopo i sette byte relativi alla variabile "AA" sono comparsi altri sette byte relativi alla variabile ("ufficialmente" inesistente) "BB", che vale zero dal momento che vi sono cinque chioccioline posizionate dopo i caratteri stessi "BB".
Questo fenomeno dimostra che, commettendo particolari errori di sintassi, costringiamo involontariamente 1'interprete Basic a riservare spazio a variabili non richieste esplicitamente.
Con ciò si spiega, dunque, la presenza di variabili non dichiarate attivando particolari procedure Dump, presenti in Tool evoluti che aggiungono altri comandi a quelli standard Commodore.

Il primo programma

Vediamo, ora, che cosa accade digitando un "vero" programma Basic:

• Cancellate lo schermo e digitate in successione tre chioccioline.
• Scendete, con i tasti cursore, sul decimo rigo e digitate la seguente riga Basic:

255 rem commodore computer club

Sul primo rigo di schermo, non appena premiamo il tasto Return, vedremo il messaggio "Commodore Computer Club" (scritto tutto in maisucolo) a distanza di sette locazioni da sinistra e, subito dopo, le "solite" tre chioccioline (che indicano la fine del listato Basic).
11 primo carattere del video è una chiocciolina (zero) richiesta dall'interprete Basic. I successivi due byte sono i byte di Link alla prossima linea, e la coppia successiva rappresenta la numerazione Basic dell'unica riga digitata: si noti che, di questa coppia di byte, il primo è il carattere grafico 255 e il secondo la chiocciolina (0).

• Salite sulla riga Basic digitata sul decimo rigo di schermo
• Cancellatela con la barra spaziatrice
• Salite sul carattere semigrafico che rappresenta il valore 255 (due quadratini contrapposti).
• Sostituitelo con una chiocciolina.
• Posizionatevi nuovamente sul decimo rigo e chiedete il List: la numerazione non e più 255 ma zero (0).
In modo analogo, seguendo la falsariga di quando fatto in precedenza, divertitevi a salire sulla prima riga, a sostituire il messaggio "Commodore Computer Club" con altri (facendo attenzione a non invadere gli altri byte) e a chiedere il List: ne vedrete delle belle; in alcuni casi, addirittura, sarete costretti a spegnere e riaccendere il computer.

Cancellate lo schermo, digitate le tre chioccioline e, posizionati sul decimo rigo, battete, dopo un opportuno New, il seguente programma:

0 For AA=1 to 10: For BB=1 to 10: For CC=1 to 10: Next CC, BB, AA

Non appena premerete il tasto Return, la prima riga di schermo, e qualche carattere della seconda, saranno riempiti da un apparente guazzabuglio di caratteri; lasciamo al lettore il compito di individuare i codici relativi ai comandi, alle variabili, al termine della riga e così via.
Battete Run e osservate lo schermo: subito dopo le tre chioccioline di fine Basic compaiono, all'improvviso, altri 21 caratteri che rappresentano, a gruppi di sette, le tre variabili "AA" "BB" "CC" dichiarate dallo stesso programma; non solo, ma noterete che, grazie ai tre cicli For ... Next nidificati tra loro, il valore (e quindi i corrispondenti caratteri visualizzati sullo schermo) cambia ad ogni ciclo. Il risultato, insomma, consiste nell'osservare comodamente, sullo schermo, ciò che avviene, in tempo reale, all'interno del computer quando vengono elaborate variabili numeriche.

Il Top di Memoria

Prima di studiare ciò che accade alle stringhe, consigliamo caldamente di rileggere le notizie che le riguardano (inserto N.43).
E' bene ricordare che l'ultima locazione utilizzabile dall'interprete Basic è puntata dalla coppia 55 e 56. Per sincerarcene effettuiamo un semplice esperimento:

• Spegnete e riaccendete il computer.
• Cancellate lo schermo, "scendete" sulla ventesima riga con i tasti cursore e digitate il seguente gruppo di comandi:

Print Chr$(14),Chr$(8): Poke 44,4: Poke 1024,0: Poke 56,6: New

In tal modo non solo facciamo coincidere l'inizio del Basic con l'inizio dell'area di schermo (come nelle esperienze di prima), ma limitiamo il cosiddetto "Top di memoria" alla locazione 6*256=1536.
Se, infatti, salite con il cursore sulla ventesima riga, la cancellate con la barra spaziatrice e chiedete un banale:

Print Fre (0)

la risposta laconica è un modesto 509.
Risalite, con pazienza, sul ventesimo rigo, cancellate ciò che eventualmente rimane e battete:

AA$="commodore computer club"

Al momento della pressione del tasto Return notiamo che alle tre chioccioline presenti in alto sul video si affiancano i sette byte che rappresentano le peculiarità della variabile stringa appena dichiarata.
Ma ciò che più attira la nostra attenzione è il fatto che il messaggio "Commodore Computer Club" compare nel bel mezzo dello schermo; i più pignoli potranno verificare che 1'ultimo carattere del messaggio coincide con la cella video 1535, che precede immediatamente la cella 1536 prima calcolata.
Le sorprese non sono finite: se saliamo ancora con il cursore sulla riga che contiene...

AA$="commodore computer club"

...e battiamo nuovamente il tasto Return, ci accorgiamo che un nuovo messaggio viene visualizzato nelle locazioni di memoria che precedono quello già visibile.
Ne deduciamo che:

• Ogni volta che si dichiara una variabile stringa, il suo contenuto "sale" all'interno della memoria Ram disponibile per 1'interprete Basic, fino a giungere all'ultima locazione che contiene l'ultima variabile dichiarata; a partire da questo istante eventuali altre variabili stringa ricominciano il ciclo a partire dal "fondo" disponibile a meno che non si impartisca un Run, un Clr oppure si richieda un Fre(0).

Supponendo di NON aver spento e riacceso il computer, cancellate lo schermo, digitate le familiari tre chioccioline, scendete sul ventesimo rigo e impartite un opportuno New. Subito dopo, risaliti sul ventesimo rigo, battete il microprogramma che segue:

100 For II=1 to 100: AA$=""+" pippo": For JJ=1 to 300: Next JJ, II

Se non ricorressimo al trucco della concatenazione delle stringhe, infatti (vedi N.43), non potremmo osservare il fenomeno della "risalita" delle stringhe, visibile con il solito Run.
Impartendo il comando Run, infatti, vedremo, come prima, la manipolazione in tempo reale della variabile II (che varia tra 1 e 100), di JJ (tra 1 e 300) e di AA$; in cui cambiano continuamente i puntatori che individuano, istante dopo istante, la posizione esatta, all'interno della Ram, della stringa elaborata.
L'esperienza effettuata consente di meglio comprendere la necessità di impostare un Top di memoria nel caso in cui si desideri allocare programmi in linguaggio macchina (o messaggi) in una zona di memoria a torto ritenuta libera: in caso contrario, infatti, la "salita" delle stringhe cancella inesorabilmente qualsiasi "cosa" incontri nel suo distruttivo cammino.

...E altre idee

Con queste ultime considerazioni termina la camminata (è il caso di dirlo) all'interno della Ram gestita dall'interprete Basic.
Ci limiteremo a sollecitare i lettori ad esplorare nuovi sistemi di protezione che utilizzino lo spostamento dell'area destinata al Basic, spostandola magari nello schermo, durante la registrazione, e il caricamento, di programmi.
Come al solito i migliori lavori che dovessero pervenire in Redazione saranno adeguatamente compensati.

Vale, ovviamente, il solito consiglio di telefonare (tel. 02/8467348) prima di inviare il frutto del vostro lavoro.