Viaggio nel Basic

Siete sicuri di conoscere come lavora il Basic durante la dichiarazione delle variabili, la stesura di un programma, la cancellazione di linee? Due inserti appositamente dedicati all'esplorazione dei meandri del vostro computer

di Alessandro de Simone

In queste righe illustreremo in modo approfondito la gestione delle variabili nei calcolatori Commodore servendoci di un gruppo di brevi programmi semplici e di immediata comprensione utilizzabili sul Commodore 64 (e sul Vic 20) e, con minime modifiche, anche sul C/16 e Plus/4.
Teniamo a precisare che gli argomenti affrontati possono sembrare molto difficili da comprendere.
Se, però, leggerete queste pagine avendo al fianco il computer acceso e digitando, volta per volta, quanto suggerito, avrete la possibilità di incrementare le vostre conoscenze nel campo della programmazione.

Organizzarsi

Dovrebbe esser noto che la memoria di un computer è una successione di byte che, nel caso del C/64, sono in quantità di *65536*. Di questa gran massa di dati alcuni sono "fissi", vale a dire che, essendo su Rom, hanno il proprio valore ben definito e non possono esser modificati in alcun modo; per analogia possiamo affermare che la Rom corrisponde alla parte stampata di un'agenda.
La memoria Ram, al contrario, può essere scritta e cancellata a volontà dall'utente, o dallo stesso computer, e corrisponde, seguendo l'analogia precedente, alle pagine bianche della stessa agenda presa in considerazione in cui possiamo scrivere, cancellare, modificare qualsiasi messaggio, a patto, ovviamente, che ci serviamo di una matita (ed, eventualmente, di una gomma).
Come in un'agenda, a differenza di un qualsiasi block notes, ogni pagina rappresenta un giorno dell'anno e, comunque, un riferimento inconfondibile, cosi in un computer non tutte le locazioni hanno lo stesso valore e importanza.
Sappiamo benissimo, ad esempio, che un certo numero, posto nella rubrica telefonica, rappresenta il prefisso di un numero di telefono perché è trascritto in una "cella" specifica. Lo stesso numero, posto altrove, può avere un significato totalmente diverso. La confusione, nel caso di un calcolatore, è decisamente maggiore perché i numeri che è possibile memorizzare sono "soltanto" 256. Oltre a verificare il valore di una cella, quindi, è indispensabile conoscere con precisione la sua posizione all'interno della memoria e, molto spesso, anche il valore delle celle adiacenti.
Se, ad esempio, digitate Print Chr$(65) otterrete la visualizzazione di una "A" maiuscola e ne potete dedurre che il numero 65 genera un carattere. Se, però, digitate Print Tab(65) sembra di non ottenere un bel niente perché il computer si predispone a stampare qualcosa (che però non è indicato) a distanza di 65 celle video.
L'esempio appena riportato è volutamente banale, ma efficace per comprendere la differenza notevole di significato di uno stesso numero in circostanze diverse.
Abbiamo detto che in una memoria Rom (Read Only Memory, vale a dire, appunto, memoria a sola lettura) non è possibile scrivere nulla, ma solo leggerne il contenuto. Le Rom contengono, quindi, programmi scritti e memorizzati permanentemente dal fabbricante (la Commodore, nel nostro caso) idonei a far funzionare correttamente il calcolatore su cui le stesse Rom sono montate.
Una parte delle Rom consente, tra l'altro, di caricare, cancellare, scrivere, modificare programmi: sono in grado, insomma, di "capire" che stiamo digitando qualcosa sulla tastiera e di interpretare la nostra volontà quando battiamo List, Run, Save e in tutte le altre circostanze ben note.
Naturalmente il computer, grazie ai programmi permanenti scritti su Rom, non memonzza a casaccio i tasti che battiamo, ma provvede a inserirli razionalmente in aree di memoria Ram ben precise e in accordo ad una rigida architettura. Il nome Ram, che vuoi dire "Random Access Memory" (memoria ad accesso casuale), non vuoi dire che i dati vengono memorizzati a casaccio, ma che è possibile memorizzarvi un dato in una parte qualunque della Ram, senza seguire un percorso preciso. Ad esempio possiamo memorizzare un dato nella cella 9870 e, subito dopo, un dato nella 567 che non ha nulla a che fare con la precedente.
Il computer (o meglio: le Rom del Basic) hanno un orientamento molto valido per individuare il punto in cui sono memorizzati i dati che interessano nel corso di un'elaborazione. Anzitutto il computer "sa" che la zona Ram che può ospitare un programma Basic da noi scritto è circoscritta e ben delimitata; nel caso del C/64 i byte che possono ospitare un programma Basic sono quelli compresi tra 2049 e 40960 (per il C116 il valore iniziale è 4096). Nella stessa area trovano posto anche le variabili intere, in virgola mobile e stringa; e anche i vettori o le eventuali matrici. Con quale criterio il computer memorizza una riga Basic, una variabile, un vettore, una stringa? E, soprattutto, come fa a rintracciarli in seguito?

I puntatori

Prendiamo in considerazione l'abitazione di un nostro amico: per individuarla abbiamo bisogno di conoscerne l'esatto indirizzo (via e numero civico), ma tali informazioni, da sole, non bastano perché il piano dell'edificio è altrettanto importante per recarci nel suo alloggio.
Allo stesso modo non serve a nulla sapere che abita al terzo piano se non conosciamo la via in cui abita.
Quando ci rechiamo da lui, quindi, "puntiamo" alla via e, individuato il portone d'ingresso, "puntiamo" al piano sul quale è ubicato il suo appartamento. Come si può notare, abbiamo bisogno di due informazioni che si rivelano indispensabili per rintracciare il nostro amico (e non altri). A pensarci bene, l'indirizzo è costituito non da uno, ma

da due puntatori: il nome della via e il suo numero civico; allo stesso modo il puntatore del piano lo possiamo considerare costituito da due informazioni: il numero del piano e il numero della porta sullo stesso piano (la seconda da destra, la prima vicino all'ascensore, la terza in senso orario a partire dalle scale, eccetera).
Anche un computer, per individuare correttamente una locazione (o un gruppo di informazioni) necessita di puntatori, vale a dire di numeri che, opportunamente interpretati, gli consentono di accedere alle informazioni desiderate.
Gli indirizzi di inizio (2049) e di fine (40960) area destinata al Basic si possono ottenere da due puntatori che la individuano in qualsiasi momento.
Provate a digitare, qualunque sia il vostro computer:

*Print Peek(43)+ Peek(44)*256*
*Print Peek(55)+ Peek(56)*256*

Otterrete, nel caso di un C/64, i valori anzidetti, a patto che abbiate appena acceso il computer e che non abbiate combinato operazioni "strane", simili a quelle che verranno tra breve descritte...
Possiamo dedurre, quindi, che due celle adiacenti (43 e 44) contengono altrettanti valori che, combinati opportunamente, rappresentano l'indirizzo di una cella particolare. A tale coppia di byte diamo il nome di "Puntatore".
Con il sistema descritto sarà possibile individuare una qualsiasi tra le 65536 celle disponibili con un C/64, e più in generale, con un computer a 8 bit.
Molto spesso al secondo puntatore (quello, per intenderci, con indirizzo più alto, esempio: 44) si assegna il nome di "pagina" ed il computer, in questi casi, viene suddiviso in 256 pagine (numerate da O a 255); al primo puntatore (quello con il numero più basso: 43) è, invece, riservato il compito di individuare un byte all'interno della pagina indicata dal secondo puntatore. Naturalmente ogni pagina è costituita da 256 byte, numerati anch'essi da O a 255.
Ecco spiegato, quindi, il valore 65536: non è altro che il prodotto di 256 (pagine) x 256 (locazioni).
Elenchiamo, ora, i puntatori più importanti che si incontrano lavorando in Basic:

43/44: inizio del Basic
45/46: fine del Basic (e inizio variabili)
47/48: fine delle variabili (e inizio dei vettori e matrici)
49/50: fine dei vettori e delle variabili
51/52: primo byte in cui verrà depositata la prossima stringa
53/54: utilità
55/56: ultimo byte usato dal Basic

Riferendosi al C/64 potremo notare che, non appena accendiamo il computer, il contenuto dei puntatori, visualizzabile inserendo tra le parentesi di un banale Print Peek() la locazione desiderata, è il seguente:

*43/44: 1,8 (1 +8*256 =2049)*
*45/46: 3.8 (3+8*256=2051)*
47/48: 3,8
49/50: 3,8
*51/52: 0,160 (0+160*256=40960)*
53/54: 0,0
55/56: 0,160

Si noti che l'inizio del programma Basic (che... non c'è dal momento che abbiamo appena acceso il computer) è distante tre byte dalla sua fine (2049 2051): l'area di inizio (45/46) e fine variabili e di inizio (47/48) e fine (49/50) vettori coincidono tra loro (2051).
Tale suddivisione, in parte contraddittoria, verrà studiata ampiamente in seguito. Per ora interessa esaminare in dettaglio ciò che succede quando iniziamo a digitare un qualsiasi programma. Trascrivete, ad esempio, le due righe seguenti:

100 REM ABC
110 REM ABC

facendo attenzione a lasciare un solo spazio tra le Rem e il gruppo di caratteri ABC e, soprattutto, a premere il tasto Return alla fine di ogni riga.
Se avete seguito alla lettera quanto detto, dovreste ritrovarvi (dopo opportuni Print Peek..) una situazione del genere:

*43/44: 1,8 (1+8*256=2049)*
*45/46: 23,8 (23+8*256=2071)*
4 7/48: 23,8
49/50: 23,8
*51/52: 0,160 (0+160*256=40960)*
53/54: 0,0
55/56: 0,160

Alcuni puntatori sono rimasti invariati (43/44, 51...56); gli altri, invece, sono cambiati proprio a causa della digitazione delle due righe Basic. Se i valori che riscontrate sono diversi, vuoi dire che avete inserito più spazi oppure più righe o un numero diverso di caratteri alfanumerici dopo le Rem (oppure che non avete un C/64!).
Digitate ora di seguito la seguente riga, in modo diretto:

FOR 1=2049 TO 2070: PRINT PEEK(I);: NEXT

...non trascurando di digitare il carattere di punto e virgola (;) dopo il Peek. Tale comando visualizzerà il contenuto (Peek) dei 21 byte numerati da 2049 a 2070. Se avete digitato senza errori, dovrebbe apparire:

11, 8, 100, 0, 143, 32, 65, 66, 67, 0
21, 8, 110, 0. 143, 32, 65, 66, 67, 0
0,0

In effetti i valori compaiono tutti di seguito e la suddivisione in Ire righe, che vedete in questa pagina, è riportata solo per motivi di chiarezza.
Noterete, infatti, che i primi due gruppi di dieci valori sono quasi simili tra loro; è poi riportata una coppia di zeri.
In particolare rileviamo che i primi due valori (11 e 8) "tradotti" con il solito sistema (11+8*256) danno come risultato il valore 2059; questo, a sua volta, "punta" alla locazione di memoria che contiene il valore 21.
Se, infatti, 11 è il contenuto della locazione 2049, 8 di 2050, 100 di 2051, si perviene a quanto asserito.
I valori 21 e 8. seguendo la stessa "traduzione", indicano la cella 2069 che contiene uno zero; stavolta, però, la coppia di zeri (2069/2070) "puntano" alla locazione zero (0+0*256= 0), e tale particolarità fa capire al Basic che il programma è finito.
Il sistema appena esaminato è formato da una successione di puntatori "in cascata" detti, più propriamente, "Link" (=legame).
E', questo, un sistema semplice ed efficace per "inseguire" le informazioni che possono risultare disseminate in qualsiasi modo all'interno del computer; è una situazione simile a quella che si verifica in una caccia al tesoro, in cui ciascun luogo da rintracciare contiene un foglietto che, a sua volta, contiene le indicazioni idonee per rintracciare il successivo.., e cosi via, fino al tesoro.
Ma esaminiamo meglio il significato dei 21 byte visualizzati:
subito dopo il byte 2049 e 2050 (i link di prossima linea sui quali ci siamo già soffermati) sono presenti due byte (2051 e 2052) che contengono, rispettivamente, 100 e 0. Tale coppia di byte (successiva, cioè, ai due byte di Link) rappresenta la numerazione della linea Basic: 100+0*256=100. Con il numero 100, infatti, abbiamo numerato la prima riga del nostro microprogramma. Analogamente (vedi byte 2061/2062) la coppia di valori 110/0 indica la numerazione della seconda linea Basic.
Il valore 143 rappresenta, nel codice Commodore, il comando REM; il codice 32 è lo spazio e i valori 65, 66, 67, rispettivamente, i caratteri "a", "b" e "c".
Proviamo a modificare, prestando la massima attenzione, i contenuti di alcune locazioni.
In 2053 è presente il codice 143. Provate a digitare...

Poke 2053,153

...e, subito dopo, chiedete il listato; otterrete:

100 PRINT ABC
110 REM ABC

Il valore 153, infatti, è il codice Commodore di Print. Divertitevi a digitare altri valori (compresi tra 0 e 255) e, subito dopo, a richiedere il listato: in alcuni casi otterrete autentiche sorprese (scomparsa dei caratteri ABC, comparsa di segni semigrafici, istruzioni tipiche del Basic e così via). Alcune Poke hanno consentito, nel passato, di realizzare particolari tipi di protezioni.
Provate, ora, ad alterare il contenuto della locazione 2051 digitando, ad esempio...

Poke 2051,91

...e a chiedere il listato:

91 REM ABC
110 REM ABC

Ciò dimostra che, in effetti, la locazione 2051 è preposta (insieme con la successiva, la 2052) a contenere la numerazione della prima linea.
Divertendovi ad alterare, sempre mediante Poke, le locazioni relative alla numerazione delle due righe, otterrete risultati stranissimi tra cui la visualizzazione di due righe in ordine decrescente anziché crescente.
Naturalmente alcune modifiche possono portare a veri e propri inchiodamenti della macchina che deve, in casi come questo, essere spenta e riaccesa per ripristinare le condizioni iniziali.
Nell'area destinata ad un programma Basic, insomma, è presente un doppio sistema di puntatori: una prima coppia (Link) che consente di conoscere la locazione Ram in cui è allocato il successivo Link; e una seconda coppia relativa alla numerazione Basic vera e propria. Ogni linea Basic termina con uno zero mentre un programma Basic è individuato da tre zeri posti in successione. Ma su questo argomento torneremo in dettaglio in seguito.
Per ora ci accontentiamo di affermare che l'interprete Basic del computer sa' che nella locazione 2048 (puntata dalla coppia 43/44) è situato il primo dei due byte che contengono le informazioni necessarie per individuare l'intera area Basic, grazie alla "cascata" di puntatori posti in successione, finché non incontra due zeri di seguito.

Esame dei puntatori

Abbiamo concluso il paragrafo precedente affermando che i byte 43 e 44 "puntano" all'inizio del programma Basic (se c'è) mentre i byte 45 e 46 puntano alla sua fine. In effetti i byte 45 e 46 puntano all'inizio delle variabili, ma, dato che queste sono allocate a partire dal byte successivo all'ultima locazione Basic, offrono comunque una valida indicazione per individuare il termine di un programma.

```
100 REM LISTATO N.1
110 :
120 REM STUDIO DEI PUNTATORI:
130 REM QUANTITA’ DI MEMORIA OCCUPATA DAL
140 REM BASIC E DA DIVERSE VARIABILI.
150 REM OPEN1,4:CMD1:REM PER STAMPARE I RISULTATI
160 A=12:A$=””+”1234567890”:A%=123
170 DIM A$(19):FOR A=0 TO 19:A$(A)=A$:NEXT
180 DIM A(19):FOR A=0 TO 19:A(A)=A:NEXT
190 DIM A%(19):FOR A=O TO 19:A%(A)=A:NEXT
200 PRINTCHR$(147)”INIZIO BASIC   “PEEK(43)+PEEK(44)*256
210 PRINT”FINE   BASIC   “PEEK(45)+PEEK{46)*256:PRINT
220 PRINT”INIZIO VARIABILI (VEDI FINE BASIC)”
230 PRINT”FINE DELLE VAR.”PEEK(47)+PEEK(48)*256:PRINT
240 PRINT”INIZIO MATRICI (VEDI FINE VARIABILI)”
250 PRINT”FINE MATRICI   ”PEEK(49)+PEEK(50)*256
260 PRINT”ALLOCAZ.(PROSSIMA SIRINGA) DA:”PEEK(51)+PEEK(52)*256
270 PRINT”FINE STRINGHE  “PEEK(53)+PEEK(54)*256
280 PRINT”FINE MEMORIA   “PEEK(55)+PEEK(56)*256
290 PRINT”RAM (EXTRA PRG.) OCCUP.DA STRINGHE”;
300 PRINT(PEEK(55)+PEEK(56)*256)—(PEEK(53)+PEEK(54)*256)-1:PRINT
310 PRINT”1 VARIABILE INTERA, 1 IN VIRG.MOB”
320 PRINT”1 VARIABILE STRINGA DI 10 CARATTERI”
330 PRINT”1 VETTORE DI 20 VALORI INTERI”;
340 PRINT”(2*20+7)”:REM  A. DE SIMONE
350 PRINT” 1 VETTORE DI 20 VALORI DECIM.”;
360 PRINT”(5*20+7)”
370 PRINT”1 VETTORE DI 20 STRINGHE (3*20+7)”;
380 REM PRINT#1:CLOSE1:REM COMANDO PER STAMPANTE
```

Digitate il programma N.1 e dategli il RUN: dovrebbe apparire quanto segue:

inizio Basic 2049
fine Basic 3094

inizio variabili (vedi fine Basic)
fine delle var. 3115

inizio matrici (vedi fine variabili)
fine matrici 3336
allocaz. (prossima stringa) da: 40750
fine stringhe 40760
fine memoria 40960
ram (extra prg.) occup. da stringhe 199

1 variabile intera, 1 in virg. mob
1 variabile stringa di 10 caratteri
*1 vettore di 20 valori interi (2*20+ 7)*
*1 vettore di 20 valori decim. (5*20+ 7)*
*1 vettore di 20 stringhe (3*20+7)*

Esaminiamo ciò che è accaduto, non senza aver dato dapprima una definizione:
Diremo che una variabile è "dichiarata" quando viene nominata per la prima volta nel corso di un'elaborazione all'interno di un programma Basic. Ciò significa, che nel caso del microprogramma che segue,...

100 a=l00: b=3.456: a$="prova"

...la prima variabile ad essere dichiarata è "A", la seconda "B" e la terza A$. Osserviamo, invece, il caso seguente:

```
100 a=100
110 gosub 200
120 b=456: c$="primo"
130 ...
140 ...
200  c=3487: d$="secondo"
210 return
```

Apparentemente l'ordine di dichiarazione delle variabili sembra essere:

A, B, C$, D$

Seguendo, invece, la struttura LOGICA del programma, il computer incontra dapprima la variabile A, ma, subito dopo, grazie a Gosub 200, memorizza "C" e "D$". Solo al "ritorno" dalla subroutine incontra "B" e "C$".
Quanto detto giustifica il motivo per cui, nei listati pubblicati, si è preferito comunicare immediatamente, nelle primissime righe, i nomi delle variabili che saranno adoperate nel corso dell'elaborazione dei singoli programmi. Il Basic, come infatti vedremo, alloca l'una in coda all'altra le variabili che a mano a mano incontra.
Ma torniamo al programma 1, ed esaminiamolo nei dettagli:

Riga 160
Dichiarazioni variabili: si noti la variabile A$ realizzata servendosi di una concatenazione. Ciò serve per "costringere" il Basic ad allocare i byte costituenti la stringa stessa in una zona estranea all'area del programma Basic, come vedremo più avanti-

Righe 170/190
Dimensionamento (e riempimento) di tre vettori di variabili miete, virgola mobile e stringhe lunghi ciascuno 20 elementi (non si dimentichi infatti la posizione "zero").

Righe 200/370
Stampa dei risultati. La visualizzazione (riportata nella precedente tabella) evidenzia in modo chiaro lo spazio occupato dalle variabili all'interno della memoria RAM.
Si precisa che i risultati di figura si riferiscono al Commodore 64. I possessori di Vic 20, C/16 e Plus/4 noteranno altri valori. Ciò è dovuto al fatto che gli indirizzi di partenza dei due computer sono diversi. Gli stessi utenti del C/64 potranno notare valori differenti da quelli riportati in figura. Il motivo deve esser ricercato nel fatto che i valori indicati cambiano a seconda della lunghezza del programma Basic. E' infatti sufficiente che, nella trascrizione del programma, un solo byte venga digitato in più, o in meno, perché la lunghezza cambi (caso dei

REM non trascritti, spazi bianchi tra istruzioni, ecc.).
Proprio a tal proposito il lettore può verificare le differenze esistenti digitando un maggior numero di righe, variabili, matrici, stringhe: in seguito a ciascuna modifica apportata, e dopo aver dato il Run, si potranno notare varie cose interessanti. Ne accenniamo alcune:

• *Aumentando, o diminuendo, il numero di righe Basic, o alterando la loro lunghezza, l'inizio del Basic non varia mai.*
• *Apportando variazioni viene modificato, invece, l'indirizzo dell'ultimo byte Basic.*
• *Se viene variato il puntatore difine Basic (a causa della vari azione apportata al programma stesso), vengono modificati TU7TI gli altri puntatori; tale alterazione è esattamente eguale (in più o in meno) alla variazione della lunghezza del programma.*
• *Ogni variabile (intera, decimale, stringa) occupa SEMPRE sette byte. Si deduce che la differenza tra i valori dei puntatori di fine ed inizio variabili è sempre un multiplo di sette (oppure vale zero nel caso non siano state dichiarate variabili).*
• *I puntatori di inizio e fine stringhe coincidono nel caso in cui le stringhe dichiarate siano allocate all'interno dell'area del programma Basic. Se invece, come nel listato presentato, esse rappresentano il risultato di una elaborazione di stringhe (Left$, Right$, somme, eccetera), la differenza tra i puntatori coinciderà con il numero dei caratteri costituenti le stringhe dichiarate, come avremo modo di studiare nel paragrafo relativo all'Overlay.*
• *I vettori occupano uno spazio diverso a seconda della propria tipologia:*

Vettori di variabili intere

Ogni vettore occupa il numero di byte seguenti:
2 per il nome del vettore (o matrice pluridimensionale).
2 per indicare il numero di byte occupati dall'intero vettore.
1 per indicare il numero delle dimensioni (max~256).
2 per ciascuna dimensione: ognuna di tali coppie ha il compito di indicare il numero di valori occupati dalla dimensione interessata.
Per ogni vettore dichiarato di variabili intere si ha pertanto un minimo di sette byte (con funzioni di "indice") necessari, al Basic, per ottenere informazioni sul vettore stesso. Oltre a quelli esaminati bisogna, ovviamente, aggiungere due byte per la memorizzazione di ciascun valore del vettore. Come è noto con due byte è possibile memorizzare valori interi compresi tra -32768 e +32767.

Vettori, o matrici, di valori decimali

Il numero di byte di "indice" sono gli stessi di quelli delle matrici intere (minimo 7). Cambia il numero dei byte per ciascun valore decimale: cinque invece di due.

Vettori stringhe

Idem come sopra per i byte di indice. Il numero per ciascun elemento del vettore è fissato invece in tre: il primo indica il numero dei caratteri della stringa esaminata, gli altri due individuano l'indirizzo del primo byte in cui è allocato il primo di essi. A questi vanno aggiunti, o meno (vedi dopo), i byte che costituiscono effettivamente la stringa stessa.

Il lettore, per verificare quanto asserito, può modificare a piacimento le linee 160-190 (listato n.l) inserendo, o cancellando, linee, dichiarando altre variabili, dimensionando in modo vario più matrici. Dopo aver apportato la variazione, digitando RUN sarà facile effettuare un controllo sui mutamenti avvenuti, specialmente per ciò che riguarda gli indirizzi di inizio e fine Basic, variabili, matrici e stringhe.
Nel caso particolare del programma N.1, le tre variabili dichiarate (A, A$, A%) rendono, appunto, pari a 3*7=21 la zona RAM dedicata ad esse. Il vettore A$(19), d'altra parte, occupa 7+3*20=67 byte, mentre A(19) richiede 7+20*5 = 107 e A%(19) 7+20*2=47 locazioni di memoria per un totale di 221 byte. Tale valore, aggiunto all'indirizzo di inizio matrici, fornisce, appunto, il valore del puntatore di fine matrici.

Esame variazione puntatori

```
100 REM LISTATO N.2
110 :
120 REM ALLOCAZIONE DELLE VARIABILI NUMERICHE
130 REM PRIMA FASE: ESAME PUNTATORI DEL BASIC E DELLE
140 REM VARIABILI NEL CASO DI 6 DICHIARAZIONI
150 :
160 PRINTCHR$(14)CHR$(147)"PRIMA DI DICHIARAZIONI":GOSUB 380
170 AA%=100: BB%=32767
180 PRINT:PRINT"DOPO DUE DICH."
190 GOSUB 380:PI=PI:PF=PF:I=I:J=J
200 PRINT:PRINT"DOPO 6 DICHIARAZIONI":GOSUB 380
210 PRINT:PRINT"DOPO MODIFICA A 2 DELLE 6 DICHIARAZ."
220 AA%=111 : BB%=-546: GOSUB 380
230 PI=PEEK(45)+PEEK(46)*256
240 PF=PEEK(47)+PEEK(48)*256
250 PRINT
260 PRINTCHRS$(18)"ELENCO VARIABILI:"
270 :
280 FOR I=PI TO PF—7 STEP 7:PRINTCHR$(18);
290 PRINTCHR$(PEEK(I))CHR$(PEEK(I+1))CHR$(146)"(";
300 PRINT PEEK(I)PEEK(I+1)")";
310 PRINT" "CHR$(18)I
320 FOR J=I TO I+6:PRINTPEEK(J);:NEXTJ
330 PRINT: NEXTI
340 PRINT"VALORI DELLE VARIABILI:"
350 PRINT:PRINT"AA%:"AA%;" BB%:"BB%;  PI:"PI;" PF:"PF;"  I:"I;"  J:"J
360 END
370 REM SUBROUTINE DI ESAME PUNTATORI
380 PRINTCHR$(18)"INIZIO DEL BASIC"PEEK(43)+PEEK(44)*256
390 PRINT"FINE BA.=IN.VAR."PEEK(45)+PEEK(46)*256
400 PRINT"FINE VARIABILI  "PEEK(47)+PEEK(48)*256
410 PRINT"N. VARIABILI DICHIAR.";
420 PRINT((PEEK(47)+PEEK(48)*256)-(PEEK(45)+PEEK(46)*256))/7
430 PRINT"N.BYTE PER VAR.DICHIAR.=";
440 PRINT(PEEK(47)+PEEK(48)*256)—(PEEK(45)+PEEK(46)*256)
450 PRINT TAB(25)CHRS(18)"PREMI UN TASTO"
460 IF PEEK(197)=64 THEN 460
470 RETURN
480 END
```

Il secondo listato presentato ha lo scopo di dimostrare che i puntatori del Basic vengono alterati anche DURANTE l'elaborazione dello stesso programma. Si noti infatti la subroutine 380-470: questa, quando viene richiamata, visualizza su schermo i valori dei puntatori che sono attivi in quel particolare momento dell'elaborazione.
Si noti, inoltre, che per il calcolo si è evitato il ricorso a variabili, proprio per rendere più comprensibile il listato stesso. Esaminiamolo, anzi, in dettaglio:

Riga 160
Prima che una qualsiasi variabile venga dichiarata, si utilizza la subroutine 380. Il risultato dimostra che, almeno in questa prima fase, i puntatori di inizio e fine variabili coincidono fra loro, proprio perché non ne sono state dichiarate.

Riga 170
Vengono dichiarate due sole variabili intere (AA% BB%) e il nuovo rinvio alla subrouitine 380 evidenzia l'occupazione di (112=) 14 byte nell'area destinata alle variabili.

Riga 200
Nuovo incremento dei puntatori di fine variabili (6 dichiarate fino a questo momento).

Riga 220
Alterazione dei valori di due variabili precedentemente dichiarate in riga 170. Si noti che i puntatori rimangono inalterati dimostrando che, una volta che una variabile numerica viene dichiarata, successive modifiche del suo

contenuto *non alterano* il numero di byte destinati all'occupazione da parte delle variabili interessate.

Riga 260-350
Utilizzando le stesse variabili che individuano i puntatori, e ricorrendo ad un ciclo For...Next con Step di 7 (multiplo di occupazione dei byte), vengono visualizzati i contenuti delle locazioni di memoria riservate alle variabili.
Ne approfittiamo per ricordare che il Chr$(14), nelle prime righe del programma, serve per passare al set maiuscolo - minuscolo in modo da render più facilmente individuabili i nomi delle variabili; in caso contrario, infatti, verrebbero evidenziati caratteri semigrafici di difficile interpretazione.
La prima delle due lettere che compaiono (in reverse) rappresentano il nome della variabile. In parentesi sono raffigurati i loro valori del codice interno Commodore. L'ultimo dato (in reverse) del primo rigo di schermo rappresenta l'indirizzo del primo byte contenente il nome della variabile.
Al rigo successivo vengono visualizzati i contenuti dei sette byte interessati dalla variabile (indicata nel rigo di schermo precedente). Si noti con attenzione il modo di allocare il nome delle variabili. Si noti inoltre il fatto che, nel caso di variabili intere o stringa, alcuni byte (gli ultimi) sono sempre posti al valore nullo.
Anche in questo caso il lettore, data la versatilità del programma proposto, può divertirsi a modificare la riga 170 inserendo altre variabili intere, decimali, stringa dai nomi più bizzarri: il segmento di programma 370/470 visualizzerà in ciascun caso ciò che succede all'interno del calcolatore quando vengono dichiarate le tre tipologie di variabili.

Proviamo ora, avendo in memoria ancora il programma 2, a dichiarare le seguenti tre variabili dal nome piuttosto simile:

170 AA%=100:AA=100:AA$="100"

Una volta dato il consueto RUN, ci accorgiamo che, alla fine, nel primo caso il nome viene visualizzato con due lettere A maiuscole, nel secondo con due minuscole mentre nel terzo caso la prima ~ maiuscola e la seconda minuscola. Questo modo di alterare la prima, la seconda o entrambe le lettere delle variabili, consente all'interprete Basic di individuare senza equivoci le diverse variabili nel corso di una qualsiasi elaborazione.

Il terzo listato

```
100 REM LISTATO N.3
110 :
120 REM SECONDA FASE: MODIFICA BYTE
130 REM RELATIVI A VARIABILI INTERE
140 :
150 AA%=100:PRINTCHR$(14)
160 PRINTCHR$(147)"PUNTATORI VARIABILE":GOSUB 380
170 PRINT:PRINT:PRINT"1) MODIFICA AA%"
180 PRINT"2) MODIFICA PUNTATORI":PRINT"*) RITORNOAL MENU"
190 GET A$:IF A$="l" THENGOSUB 240:GOTO 170
200 IF A$="2" THEN GOSUB 280:GOTO 170
210 G0T0 190
220 :
230 REM MODIFICA VALORE DI AA%
240 PRINT:INPUT"(*) AA%=";AA$:IF ÀA$="*" THEN RETURN
250 AA=VAL(AA$):IF AA>32767 OR AA<—32768 THEN 240
260 AA%=AA:GOSUB 380:GOTO 240
270 REM MODIFICA BYTE VAR. AA%
280 INPUT "(*) BYTE";AA$:AA=VAL(AA$)
290 IF AA$="*" THEN RETURN
300 IF AA>I+6 OR AA<I THEN 280
310 A1=AA
320 INPUT "(*) VALORE";AA$: AA=VAL(AA$)
330 IF AA$="*" THEN 280
340 IF AA>255 OR AA<0 THEN 320
350 POKE Al,AA:GOSUB380:GOTO280
360 :
370 REM VISUALIZZAZIONE BYTE VAR. INTERA AA%
380 I=PEEK(45)+PEEK(46)*256:X1=PEEK(I):X2=PEEK(I+1)
390 PRINTCHR$(18)I;X1;CHRS(X1)
```

```
400 PRINTCHRS(18)I+1;X2;CHR$(X2)
410 FOR J=I+2 TO I+6:PRINT[RVS]"J"[RVOFF]"PEEK(J):NEXT
420 PRINT:PRINT"AA%="AA%;"   BB%="BB%;"   ÀB%="AB%;"   BA%="BA%
430 RETURN
440 END
```

Finora abbiamo assistito... passivamente al modo in cui il calcolatore gestisce la memoria RAM nel caso debba definire variabili di qualunque tipo. Vediamo ora che succede se noi, intenzionalmente, proviamo ad alterare il loro contenuto. A tale scopo digitiamo il programma N.3. E' ovvio che il lettore, con la propria fantasia, può apportare tutte le sofisticazioni che ritiene opportune.

Righe 160-210

Menu di scelta. Viene chiesta una delle due scelte possibili:
1) Alterazione del valore di AA% (dichiarata in riga 150)
2) Alterazione dei singoli byte costituenti la variabile.
Nel caso si scelga l'opzione 1, verrà richiesto un nuovo valore da attribuire alla variabile AA% (righe 240/260). E' ovvio che vengono rifiutati valori che escono dall'intervallo -32768 + 32767 (riga 260). Subito dopo il valore digitato viene depositato in AA% e la subroutine 380/430 viene incaricata di visualizzare non solo il nome della prima variabile dichiarata (che. guarda caso, è proprio AA%), ma anche il contenuto dei singoli byte relativi ad essa, oltre alloro indirizzo.
Digitando il carattere di asterisco (*) si ritorna al menu principale. Se ora si sceglie l'opzione 2 (modifica puntatori) si avrà la possibilità di alterare il contenuto dei sette byte, uno alla volta. Modificandoli a caso, ma con criterio, si possono fare le seguenti deduzioni:
a/ Alterando il contenuto degli ultimi tre byte, il valore di AA% non viene in alcun modo modificato. Ciò dimostra che il Basic, nel caso di variabili intere, "guarda" esclusivamente il terzo ed il quarto byte.
b/ Alterando il contenuto di questi due byte si ottengono (riga 420) valori diversi di AA%
c/ Alterando i primi due byte (relativi cioè al nome della variabile) ci accorgiamo che il computer non riconosce più la prima variabile che incontra col nome AA%, ma col nome che le abbiamo attribuito! Grazie alla riga 420 vengono visualizzati i valori relativi a quattro variabili intere (AA% AB% BA% BB%):
servono per verificare l'effettivo cambio di nome causato dalle Poke. Provate dunque (avendo davanti a voi la tabella del codice ASCII e addizionando 128 al valore del carattere tabellato) a modificare in tal senso i due byte relativi al nome, dapprima trasformando AA% in BB% (cioè 193-193 in 194-194) e poi in altre lettere qualunque.

```
100 REM LISTATO N.4
110 :
120 REM TERZA FASE : MODIFICA BYTE RELATIVI
130 REM A VARIABILI IN VIRGOLA MOBILE
140 :
150 AA=100: PRINT CHR$(14)
160 PRINT"[CLEAR]PUNTATORI VARIABILE": GOSUB 380
170 PRINT"[D0WN]1) MODIFICA AA"
180 PRINT"2) MODIFICA PUNTATORI":PRINT
190 GET A$: IF A$="1" THEN GOSUB 290:GOTO 170
200 IF A$="2" THEN GOSUB 280: GOTO 170
210 GOTO 190
220 :
230 REM MODIFICA VALORE DI AA
240 INPUT "[DOWN]AA=";AA$: IF AA$="*" THEN RETURN
250 AA=VAL(AA$)
260 GOSUB 380: GOTO 240
270 REM MODIFICA BYTE VAR. AA
280 INPUT "[DOWN]BYTE";AA$:X=VAL(AA$)
290 IF AA$="*" THEN RETURN
300 IF X >I+6 OR X <I THEN 280
310 A1=X
320 INPUT "VALORE";AA$:X=VAL(AA$)
330 IF AA$="*" THEN 280
340 IF X>255 OR X<0 THEN 320
350 POXE A1,X: GOSUB 380: GOTO 280
360 :
370 REM VISUAL.BYTE VAR.VIRGOLA MOBILE AA
380 I=PEEK(45)+PEEK(46)*256:X1=PEEK(I): X2=PEEK(I+1)
```

```
390 PRINT "[RVS]" I;X1 CHR$(X1)
400 PRINT "[RVS]" I+1 X2 CHR$(X2)
410 FOR J=I+2 TO I+6:PRINT "[RVS]" J "[RVOFF]" PEEK(J):NEXT
420 PRINT"AA="AA;"   BB="BB;"   AB="AB;"   BA="BA
430 RETURN
44O END
```

Il listato N.4 modifica i byte relativi a variabili decimali ed è sostanzialmente identico a quello N.3. Le differenze consistono nel trattamento di valori numerici diversi.

```
100 REM LISTATO N.5
110 :
120 REM QUARTA FASE: ESAME ALLOCAZIONE
130 REM MATRICI VARIABILI INTERE E NON
l4O PRINT CHR$(147)
150 X1=X1:X2=X2:X3=X3:X4=X4:X5=X5:I=I
160 :
170 DIM AA%(12,34),CA(12),CX%(32)
180 :
190 PRINT CHR$(14)
200 X1=PEEK(47) + PEEK(48)*256: X2=PEEK(49)+PEEK(50)*256
210 PRINT"[RVS]ULTIMO BYTE 0CCUPATO[RVOFF]" X2
220 PRINT"[RVS]"Xl PEEK(X1) CHR$(PEEK(X1)) "[RVOFF] NOME"
230 PRINT"[RVS]"X1+1 PEEK(X1+1) CHRS(PEEK(X1+1)) "[RVOFF] VETTORE"
240 PRINT"[RVS]"X1+2 PEEK(X1+2) "[RVOFF] BYTE +"
250 PRINT "[RVS]"X1+3 PEEK(X1+3) "[RVOFF] *256 OCCUPATI =";
260 X4=PEEK(Xl+2)+PEEK(Xl+3)*256:PRINTX4
270 X5=X1+PEEK(X1+2)+PEEK(X1+3)*256
280 PRINT"(OCCUPAZIONE FINO A" X5—l ")"
290 PRINT"[RVS]" X1+4 "[RVOFF]" PEEK(X1+4)" DIMENSIONI=";
300 X3=X1+4: PRINT "[RVS]" PEEK(X3)
310 FOR I=1 TO PEEK(X3)*2: PRINT"[RVS]"X3+I+1;PEEK(X3+I):NEXT
320 IF X2—X1=X4 THEN 370
330 X1=X5: REM A. DE SIMONE DIDATTICA '84
340 PRINT "[RVS]PREMI UN TASTO"
350 IF PEEK(197)=64 THEN 350
360 GOTO 220
370 LIST 170
```

Il listato N.5, che non viene descritto in maniera approfondita, esamina l'allocazione dei vettori e segue le regole prima descritte. Il lettore può verificarlo alterando la riga 170 inserendo più vettori o dimensionandoli diversamente. L'automatismo del programma consente di esaminare, byte dopo byte, ciascun valore dichiarato. Diremo soltanto che, anche in questo caso, i primi vettori che si incontrano sono proprio quelli dichiarati seguendo l'ordine "logico" del programma.

```
100 REM LISTATO N.5
110 :
120 REM STUDIO ALLOCAZIONE STRINGHE
130 :
140 AA$="STRINGA"
150 X1=Xl:X2=X2:I=I:PRINT "[CLEAR]"
160 Xl=PEEK(45)+PEEK(46)*256
170 X2=PEEK(47)+PEEK(48)*256
180 PRINT Xl "[RVS]" CHR$(PEEK(X1))"[RVOFF] NOME"
190 PRINT X1+1 "[RVS]"CHR$(PEEK(Xl+l))"[RVOFF] STRINGA"
200 PRINT X1+2 "LUNG. SIRINGA =[RVS]" PEEK(X1+2)
210 PRINT X1+3 "LA.INDIR.STRINGÀ=[RVS]" PEEK(X1+3)
220 PRINT X1+4 "HA.INDIR.STRINGA=[RVS]" PEEK(X1+4)
230 PRINT"[DOWN][RVS](" PEEK(X1+3) "[LEFT] +" PEEK(X1+4)"[LEFT] * 256 =";
240 X2-PEEK(X1+3)+PEEK(X1+4)*256:PRINTX2"[LEFT])[DOWN]"
250 FOR I=0 TO PEEK(X1+2)—1
260 PRINT "[RVS]"X2+I"[RVOFF]" CHR$(PEEK(X2+I)):NEXT
270 LIST 140
```

In quest'ultimo listato (N.6) viene esaminato il luogo in cui i caratteri delle singole stringhe dichiarate vengono depositati. Il lettore può provare ad alterare il contenuto della variabile AA$ (riga 140) oppure a modificarlo ricorrendo a concatenazioni di più stringhe. Uno studio particolareggiato sull'allocazione dei vettori stringa non viene indicato dato che, a questo punto, il lettore, seguendo la falsariga dei programmi pubblicati in queste pagine, può farlo da solo.

Un breve riepilogo

• Nel C/64 la prima locazione di memoria a disposizione è la 2049; nel seguito dell'inserto verrà indicata con LI (Locazione Inizio).
• L'ultima locazione occupata da un programma Basic varia, come è intuitivo, a seconda della lunghezza del programma stesso, e verrà indicata con LF (Locazione Fine).
• Il cosiddetto "indirizzo" di partenza del programma Basic viene calcolato esaminando il contenuto di due locazioni di memoria (puntatori): la 43 e la 44.
• Per conoscere, dunque, l'indirizzo di partenza del programma presente in memoria, sarà sufficiente eseguire il calcolo:

*LI=PEEK(43) +PEEK(44)*256*

In genere il risultato porterà ai valori prima visti (2049 nel C/64).
• Altre due locazioni (la 45 e la 46) contengono, in "codice", l'indirizzo dell'ultima locazione interessata dal programma Basic presente in memoria in quel momento:

*LF=PEEK(45) +PEEK(46)*256*

LF, come già detto, cambia a seconda della lunghezza del programma presente.
• Non appena si accende l'apparecchio Li coincide con LF.
• Col termine "Puntatore" si intende il valore numerico della
locazione interessata in un'operazione. Ad esempio le locazioni 43 e 44 "puntano" all'inizio del programma Basic mentre le 45 e 46 puntano alla sua fine.

L'esigenza dell'overlay

A volte capita, anche se è molto raro, che alcuni programmi non possono essere ospitati in un computer perché sono più lunghi della memoria disponibile. Se infatti si cerca di digitarli una linea alla volta, queste vengono accettate fino a che è possibile: tentando di proseguire si ottiene null'altro che un messaggio di "Out of memory error" che impedisce il proseguimento della digitazione.
Analogamente programmi anche molto brevi e di conseguenza caricabili o digitabili senza difficoltà alcuna, contengono alcune istruzioni del tipo DIM che, per essere eseguite, richiedono una certa quantità di memoria.
Sembrerebbe che, nei casi appena visti, sia impossibile utilizzare i programmi stessi avendo a disposizione una quantità modesta di RAM.
Molto spesso, però, un certo numero di linee di programma vengono usate per "inizializzare" il programma stesso e non occorrono più nel resto dell'elaborazione. Si può, allora, caricare tale parte di programma, farla girare, ed in seguito caricare ed utilizzare la seconda parte.
Se però si agisce come è stato descritto, al momento di impartire il secondo RUN (in seguito al caricamento della seconda parte), si azzerano tutti i valori delle variabili inizializzate precedentemente.
Per non perdere il contenuto delle variabili la Commodore specifica che se si inserisce, all'interno di un listato, una riga del tipo...

XXX Load"seconda parte",8

...il programma con tale nome viene caricato e "lanciato" senza alterare in nessun modo le variabili elaborate dalla prima parte del programma stesso. Possiamo iniziare i nostri esperimenti digitando i seguenti due programmi che differiscono tra loro in minima parte:

100 print"questo è"
110 print"il programma
120 print"alfa"
130 geta$:ifa$= ""then 130
140 ifa$= "b"then 160
150 print:gotol00

```
160 load"beta",8
170 print 2*3-67
180 print "frase"
```

Il listato qui sopra riportato deve essere memorizzato su disco con il nome “alfa”. Dopo averlo registrato, sostituite (righe 120 e 160) il nome “Alfa” con “Beta”, e viceversa, e registratelo nuovamente con il nome “beta”:

```
100 print"questo e'"
110 print"il programma"
120 print"beta"
130 geta$:ifa$= ""then 130
140 ifa$="a"then160
150 print:goto100
160 load"alfa",8
170 print 2*3-67
180 print "frase"
```

Si noti che la lunghezza dei due programmi è rigorosamente identica; ve ne potete accorgere richiedendo Print Fre (0) che deve fornire due valori assolutamente eguali.
Se, ora, fate girare uno qualsiasi dei due programmi, noterete due cose molto importanti:

• premendo un tasto qualsiasi, apparirà il messaggio che indica quale dei due programmi è presente in memoria; premendo “A” (oppure “B”) verrà invece caricato l’altro listato e immediatamente fatto partire.

• le istruzioni presenti nelle righe 170 e 180 (successive, in altre parole, alla riga contenente Load) non vengono mai eseguite:
non appena un programma è caricato, questo parte dalla SUA prima riga.
A parte questo. non si verificano inconvenienti di sorta proprio perché i due listati, volutamente banali, occupano la stessa area di memoria. Provate, però, ad effettuare il seguente esperimento:

- spegnete e accendete il computer
- digitate il seguente listato:

```
100 rem differenze
110:
120 print:print"1 - eseguo differenze"
130 print"2- carico somme"
140 geta$:ifa$= ""then 140
150 ifa$= "1"thengosub 180
160 ifa$="'2"then240
170 print:goto120
180 print:print"(0= ritorna)"
190 input"minuendo";x
200 ifx=0thenprint:return
210 input"sottraendo";y
220 print"differenza="x-y
230 print:goto180
240 load"somme",8
```

- registratelo con il nome “Differenze”.
- cancellatelo, digitate il seguente e registratelo con il nome
somme

```
100 rem somme
110:
120 print:print"1- eseguo somme"
130 print"2- carico differenze"
140 geta$:ifa$= ""then 140
150 ifa$="1"thengosub180
160 ifa$="2"then240
170 print:goto120
180 print:print"(0= ritorna)"
190 input"primo addendo";x
200 ifx=0thenprint:return
```

```
210 input"secon.addendo";y
220 print"somma= "x+y
230 print:goto180
240 load"differenze",8
250 rem questo programma, come si può notare, è più lungo del listato
260 rem chiamato "differenze" a causa, se non altro, di queste righe rem
```

• caricate il programma "differenze" e date il Run
• fate in modo che, premendo il tasto "2", venga caricato il programma "somme".
• premete il tasto Run/Stop e chiedete il List... Sorpresa! Che fine hanno fatto le altre righe Rem e che cosa rappresenta quella riga "strana" prima della fine?
Ma le sorprese non sono finite... Provate a digitare un semplice:
300 rem
...lo schermo si sconvolge e non è possibile recuperare il programma nemmeno con i tasti Run/Stop e Restore. E' necessario spegnere e riaccendere il computer.
Il richiamo di altri programmi, durante l'elaborazione di un programma, è possibile ma è indispensabile che il programma richiamato sia più breve (o al massimo eguale) di quello che lo "chiama"; vedremo tra breve il perché.
Allo scopo di capire in quali casi la procedura può essere applicata e in quali, al contrario, può dar luogo ad inconvenienti di vario tipo, consigliamo di seguire *alla lettera* le fasi qui di seguito indicate.
Al termine della lettura sarete sorpresi della semplicità con cui è possibile aumentare le potenzialità di un personal computer Commodore.

Primo esperimento

Fase 1) Accendete il computer oppure, se è acceso, spegnetelo e riaccendetelo in modo da esser sicuri della sua corretta inizializzazione.

```
100 rem prova n.1/a
110:
120 print chr$ (18) "inizio prova 1/a"
130 a=1:b=2:c=3:d=4
140 e=1.2:f=2.3
150 g=3.4: h=5.6
160 print a;b;c;d;
170 print e;f;g;h
180 print chr$(18)"fine prova 1/a"
190 :
200 rem queste tre righe
210 rem servono solo per
220 rem allungare il brodo
230 :
240 load "prova n.1/b",8
```

Fase 2) Digitate il programma n.7 (dal nome "Prova N.l/a") così come è pubblicato, tranne la riga 240. Controllate che funzioni correttamente, digitate la riga 240 e registratelo su di un disco oppure su di un nastro (che numerate con 1). E' ovvio che la sintassi riportata è valida per chi utilizza l'unità a dischi Chi si serve del registratore a cassette trascriverà semplicemente:

```
240 Load"Prova n.1/b"
```

Tale avvertenza (eliminazione di ",8") vale per tutti i brevi listati presentati nel caso si desideri ricorrere al registratore.

Fase 3) Spegnete e riaccendete il computer. Digitate il programma n.8, provatelo (ottenendo una serie di zeri), e registratelo col nome "Prova n.l/b" nel modo consueto sullo stesso disco oppure su un altro nastro (N.2): Utilizzando due nastri si semplificheranno le varie operazioni che stiamo per descrivere.

```
100 rem prova n.1/b
110 print
120 print "inizio prova 1/b"
130 pnnt a;b;c;d;
140 print e,f;g;h
150 print "fine prova 1/b"
```

Fase 4) Caricate, da nastro o disco, il programma “Prova n.1/a” e digitate RUN. Ciò che accade è piuttosto semplice: Dapprima verrà elaborato il programma presente in memoria; in seguito (riga 240) verrà caricato il listato “Prova n.l/b”. La successiva visualizzazione delle stesse variabili di prima dimostra che, nonostante sia stato caricato un nuovo programma, le variabili dichiarate precedentemente non vengono in alcun modo alterate. Se infatti, all’apparizione del consueto Ready chiediamo il listato, appare null’altro che quello n.8, vale a dire quello caricato e lanciato grazie alla riga 240. Sembrerebbe che la Commodore abbia proprio ragione. Ma... è davvero tutto in ordine?

Fase 5) A questo punto, (vale a dire avendo in memoria il programma “Prova n.l/b” caricato dal programma “Prova n.l/a”), aggiungiamo una linea qualunque come la:

160 rem commodore comput.club

Se ora chiediamo il listato, esso apparirà più lungo di una sola riga e comunque più breve di quello n.7.

Fase 6) Registriamo il programma così formato, con lo stesso nome di prima in modo che possa esser richiamato nuovamente dal programma “Prova N.l/a”.
Il modo di effettuare questa operazione dovrebbe esser noto ma lo riassumiamo brevemente:

Open 15,8,15,“S:Prova N1/a”:cIose1
Save”Prova n.1/b”,8

nel caso si possegga un drive 1541, e:
Save”Prova n.1/b”

utilizzando il nastro 2 (vale a dire cancellando il precedente programma ivi registrato) nel caso si usi il registratore.

Fase 7) Spegnete, riaccendete e caricate da nastro o disco il “vecchio” programma (Prova n. 1/a).
Impartite il RUN. Le istruzioni del primo programma, come è facile verificare, vengono correttamente eseguite. Non appena, però, viene caricato e lanciato il secondo programma, ci accorgiamo che il risultato è diverso da quello visto nella fase 4. Ciò è accaduto, come vedremo tra breve, perché il secondo programma è più lungo del primo e provoca gli stessi inconvenienti dei due programmi “Somme” e “Differenze
Come è possibile che ciò sia avvenuto? Chiediamo il listato (List): esso è proprio identico a quello che ci aspettiamo con in più la sola riga aggiunta nella fase 5: è comunque ben più breve del primo!
Ebbene il secondo programma *sembra* più breve del primo ma risulta *effettivamente* più lungo. Vediamo di spiegarcene il perché.
Torniamo ad esaminare la fase 4 e confrontiamola con quanto segue: quando il programma “Prova n. 1/a”, giunto alla riga 240, carica il programma “Prova n.1/b”, questo viene allocato a partire da LI. I puntatori di LF rimangono però *inalterati.* Ciò è dovuto al fatto che le variabili precedentemente dichiarate iniziano a partire da LF.
Se un programma, ad un certo punto di un’elaborazione, “chiama” un altro programma, questo, dopo il caricamento, viene considerato lungo quanto il precedente anche se è composto da una sola riga! Pertanto l’interprete Basic “vede”, dopo il caricamento di un programma, un listato che termina in LF e se, a questo punto, aggiungiamo una qualsiasi riga, questa viene allocata come di consueto, ed i puntatori di fine Basic, di conseguenza, alterati come se fosse stata aggiunta ad un programma più lungo.
Quanto descritto è proprio l’errore che (intenzionalmente) vi abbiamo fatto commettere aggiungendo la riga 160 nella fase *5.* Questa è stata infatti aggiunta *dopo* che il programma lungo” era stato caricato automaticamente da “Prova n.1/a”. Ciò dimostra che, effettivamente, nel caso di ricorso a tecniche di overlay, è indispensabile che il primo programma da caricare in memoria sia il più lungo di quello (o quelli) che in segito saranno richiamati in cascata e, soprattutto, che non vengano modificati i programmi durante tali delicatissime fasi.
La spiegazione dell’inconveniente è piuttosto semplice da comprendere: dopo l’area Basic inizia l’area delle variabili. Se “invadiamo” una zona di quest’ultima con un programma, la “coda” di questo ne cancella alcune ed esattamente quelle che per prime erano state dichiarate nel corso dell’elaborazione del primo programma.

Verifica del primo esperimento

1) Per verificare quanto è stato affermato, digitare il programma “Prova n.2/a” privo della linea 280; dopo aver verificato che funziona correttamente, digitare la riga 280 e registrano col nome “Prova n.2/a”.

100 rem prova n.2/a
110:
120 print “inizio prova 2/a”
130 a=1:b=2:c=3:d=4

```
140 e=1.2:f=2.3
150 g=3.4: h=5.6
160 print a;b;c;d;
170 print e,f;g;h
180 print "Inizio memoria =";
190 print peek(43)+peek(44)*256
200 print "fine memoria =";
210 print peek(45)+peek(46)*256
220 print "fine prova n.2/a"
230:
240 rem queste tre righe
250 rem servono solo per
260 rem allungare il listato
270 :
280 load "prova n.2/b",8
```

2) per sicurezza spegnete, riaccendete e digitate il programma “Prova n.2/b”.

```
100 rem prova n.2/b
110 :
120 print "inizio prova 2/b"
130 print a;b;c;d;
140 print e;f;g;h
150 print peek(43)+peek(44)*256
160 print peek(45)+peek(46)*256
170 print "fine prova 2/b"
```

Una volta controllatone il corretto funzionamento, trascriviamo su di un foglio di carta i valori forniti dalle righe 150 e 160. Registratelo dunque col nome “Prova n.2/b” in modo che in seguito possa esser richiamato da programma.
3) Carichiamo “Prova n.2/a” e “lanciamolo”. Al termine dell’elaborazione (dopo cioè che viene caricato e lanciato “Prova n.2/b”, riga 280), ci accorgiamo che il secondo programma, benché più breve del primo, possiede gli stessi puntatori di fine Basic. In ogni caso i puntatori indicano valori maggiori di quelli precedentemente trascritti sul foglio di carta.
4) Col programma “Prova n.2/b” (che è ora allocato in memoria perché richiamato da “Prova n.2/a”), aggiungiamo la riga:

180 Rem Commodore Computer Club

oppure una qualsiasi altra riga.
5) Digitiamo RUN e Return. I valori delle variabili (A, B, *C,* D, E, F) sono ovviamente tutti nulli non solo perché è stato dato il RUN, ma anche perché è stata aggiunta una riga. Ricordiamo, per inciso, che l’inserimento, la cancellazione o la semplice modifica di una riga del programma azzera sempre e comunque qualsiasi variabile memorizzata.
Possiamo dunque notare che i puntatori di fine Basic (elaborazione di riga 160) vengono incrementati.

Conclusioni sul primo esperimento

• In un programma *è* possibile, mantenendo inalterate le variabili numeriche, dare l’ordine di caricare un altro programma, purché quest’ultimo sia più breve o al massimo di eguale lunghezza di quello “chiamante”.
• Il programma chiamato non deve in nessun caso essere manipolato, pena la perdita della caratteristica di “brevità”.
• Nel caso si desideri modificare il programma chiamato, procedere come segue:
- Spegnere e riaccendere il computer.
- Caricare il programma che si desidera modificare.
- Modificarlo e registrarlo nuovamente con lo stesso nome di prima avendo l’accortezza di verificare che le modifiche apportate non l’abbiano reso più lungo del programma che, in seguito, lo chiamerà.
• Per conoscere la quantità di byte occupata da un programma, digitare *non 11 semplice* Print Fre(0) ma:
Clr: Restore: Print Fre(0) (Retum).
Se il valore che risulta è negativo, digitate:

Print 65536 + Fre(O)

La gestione delle variabili stringa

I computer Commodore allocano le variabili stringa, a seconda dei casi, in due zone della memoria. Una di queste si trova *al di là* dell'indirizzo di fine Basic. Alcune variabili, invece, vengono individuate, dall'interprete, all'interno del programma Basic stesso.
Tale metodo di memorizzazione dei dati (tipico delle sole variabili stringa), che tra breve verrà in parte descritto, ha lo scopo di ottimizzare l'occupazione della memoria RAM. Vediamo come:
Se in un programma Basic figura una linea del tipo:

100 a$= "abcde":print a$

l'interprete non ha motivo di depositare i caratteri alfanumerici "abcde" in una zona di memoria "esterna" a quella occupata dal programma Basic. Poiché, infatti, tali caratteri si trovano di già in una zona RAM, a quale scopo depositarli da qualche altra parte, creando un inutile doppione? In altre parole quando il computer incontra un'istruzione del tipo Print A$, andrà a rintracciare l'area della memoria alla quale sono associati i vari caratteri. Nel caso del microprogramma appena visto, questa si trova esattamente ad otto byte di distanza dall'inizio del programma stesso, vale a dire (ma dovreste ormai saperlo): 2 byte di link, 2 di numerazione Basic, 2 del nome della varialbile, 1 per il simbolo (=) ed uno, infine, per il simbolo degli apici (").
Ben diverso è il caso della linea Basic:

100 a$="abc"+"defQ print a$

In questo caso (come in tutti i casi di manipolazione di stringhe) il computer *deve* allocare la stringa A$ (ottenuta come elaborazione di due gruppi di caratteri) in qualche parte della memoria, *diversa* da quella relativa ai programmi Basic. Una cosa è infatti disporre dell' indirizzo di partenza in cui si trovano *tutti in fila* i caratteri relativi alla stringa, altra cosa è invece disporre dell'indirizzo in cui è situata un'operazione di concatenazione di stringhe. Per sincerarcene, digitate e fate girare i due micro-programmi seguenti:

100 printfre(0)
110 a$= "abcdefghij":print a$: printfre(0)

e questo...

100 printfre(0)

110 a$= " abcde"+"'fghij": print a$: printfre(0)

Quest'ultimo sembra essere più lungo del precedente di appena tre byte, cioè i caratteri apici, più, apici ("+"). Infatti è proprio così e ce lo dimostra il valore di FRE(0) clic appare prima della visualizzazione di "abcdcfghij". Il secondo valore che appare, indica, invece, il numero di byte a disposizione *dopo* che la variabile A$ è stata dichiarata.
Mentre nel primo programma la variabile A$ è all'interno del programma stesso, nel caso successivo A$ costituisce il risultato di una manipolazione e viene pertanto situata automaticamente in altra zona occupando, di conseguenza una quantità di memoria maggiore, rispetto al precedente microprogramma, del numero di caratteri costituenti la stringa somma.

Conclusioni

1) Se in un programma viene definita una variabile stringa, essa non occupa altra zona se non quella all'interno del programma Basic stesso e viene individuata da puntatori che, nonostante siano situati, come tutti i puntatori, *al di fuori* dell'area destinata al Basic puntano a quelle locazioni di memoria.
2) Se la variabile stringa è il frutto, invece, di una manipolazione (Right$ Left$ ecc.) oppure di una somma (A$+B$), essa viene allocata, come i suoi puntatori, in una zona situata al di fuori dell'area riservata ai programmi Basic.
Una descrizione dettagliata di come il Basic gestisce le stringhe ed i suoi puntatori esula dallo scopo del presente inserto.

(Continuazione e fine al prossimo numero)

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

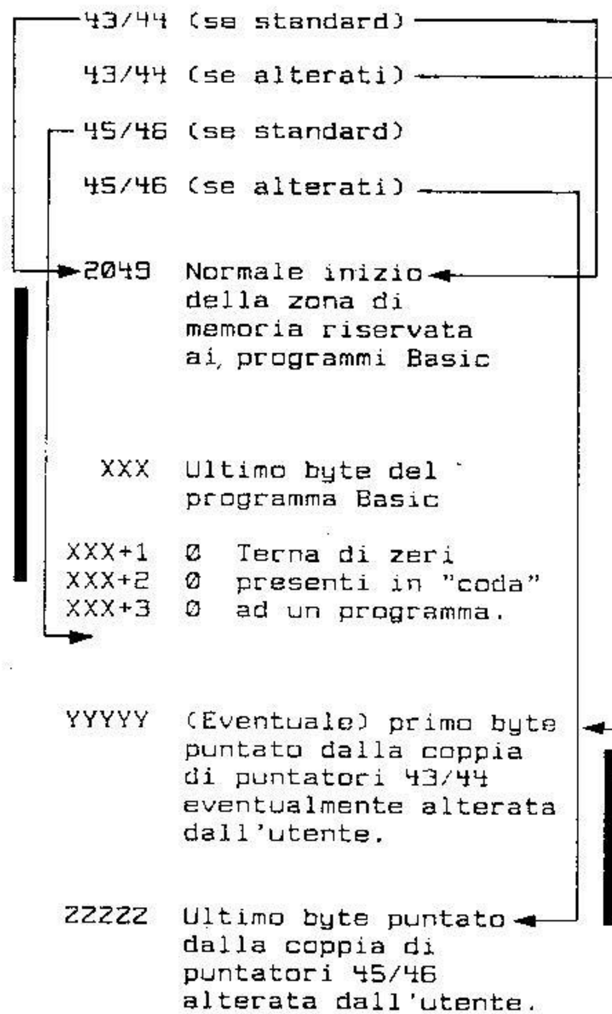
```
43/44   1/8   Puntatori di inizio
              (1+8*256=2049)

45/46   27/8  Puntatori di fine
              (27+8*256=2075)

2049     9      Primo link
2050     8      di linea
                9+8*256=2057

2051   232      Numerazione
2052     3      prima linea
                232+3*256=1000

2053   143      Rem
2054    32      spazio
2055    65      Carattere "a"
2056     0      Fine linea

2057    17      Secondo link
2058     8      di linea
                17+8*256=2065

2059   242      Numerazione
2060     3      seconda linea
                242+3*256=1010

2061   143      Rem
2062    32      Spazio
2063    66      Carattere "b"
2064     0      Fine linea

2065    25      Terzo link
2066     8      di linea
                25+8*256=2073

2067   252      Numerazione
2068     3      terza linea
                252+3*256=1020

2069   143      Rem
2070    32      Spazio
2071    67      Carattere "c"

2072     0      Fine
2073     0      area
2074     0      del Basic
```

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

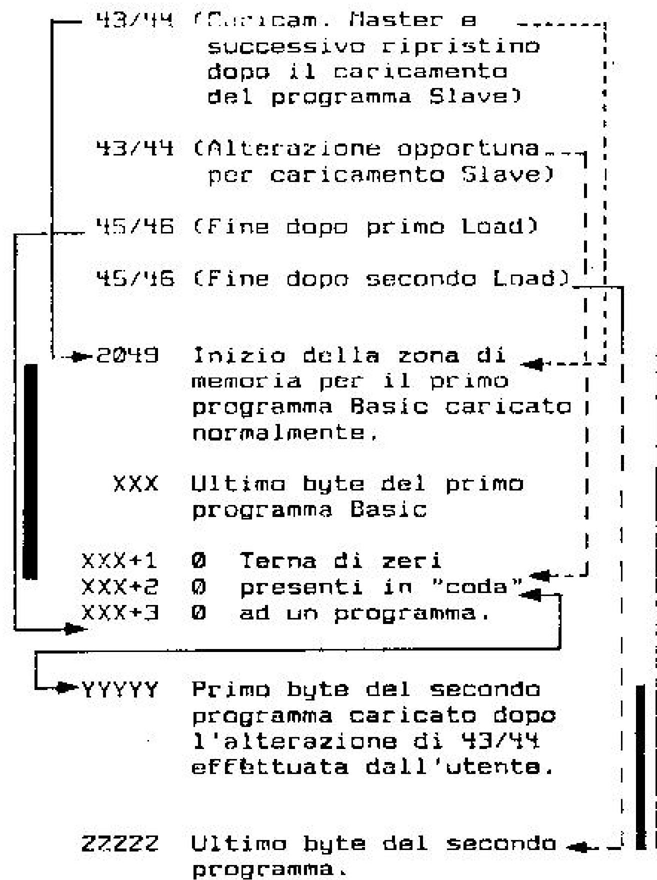
...nel caso sia digitato su un C/64.

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.ˉ 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. r
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 nul1o. 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.