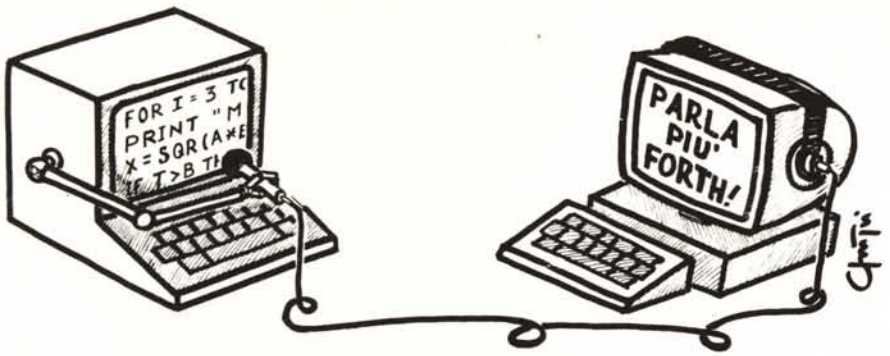

Parla più FORTH

di Raffaello De Masi

Nona parte

Word di input/output

Ci avviamo, ormai, verso la conclusione della nostra (lunga?) trattazione del Forth. Chi ci ha seguito ha avuto modo di macinare vecchie conoscenze di altri linguaggi viste in un'altra ottica, o, ancora, nuovi mezzi di cui non aveva idea. Ma, finora, il lavoro dell'utente era al massimo quello di inserire, dall'inizio, i valori da elaborare e manipolare nello stack, e di avviare il programma. Il programmatore o l'utilizzatore si servivano del computer con lo stesso rapporto che intercorre tra un autore ed un attore. Il primo scrive un copione (il programma) che il calcolatore (l'attore) esegue pedissequamente senza intervenire, in alcun modo, a modificarne il corso; inoltre, lo stesso autore (programmatore), una volta avviata la rappresentazione (programma), non ha più la possibilità di intervenire per modificarne il corso.

Generalmente, è questo il caso dei programmi Forth, favoriti da quel mezzo marinaio che è lo stack, capace di accumulare fin dall'inizio i dati da manipolare. Esiste, però, la possibilità che, pervenuto ad un certo punto, il programma abbia necessità di acquisire nuovi dati non inseribili, fin dall'inizio, nello stack, o, ancora, debba assumere delle decisioni non risolubili con semplici sequenze decisionali. È, ad esempio, il caso dei videogame (in cui trova ampia applicazione il Forth, almeno negli Stati Uniti) dove, evidentemente, esiste una inderogabile necessità di ricevere, in tempo reale, continue indicazioni dall'esterno (manopole, tasti e comunque elementi di controllo). Inoltre, quale mezzo di segnalazione di un messaggio, sul video o su una stampante, conosciamo solo la word [.], sia essa applicata a numeri o frasi.

Questa puntata insegna le word necessarie ad interagire col programma, word che accettano informazioni dalla tastiera (o da strumenti di misura, quali trasduttori, sensori, termocoppie, ecc) e word che ne inviano a strumenti di segnalazione o comunque di output (stampanti, videoterminali, memorie di massa, ecc.). Le informazioni, in Forth come in pressoché tutti gli altri linguaggi, sono trasmesse nello speciale codice ad 8 bit detto ASCII, e molte operazioni di cui parleremo coinvolgono tali caratteri. In questa occasione, poiché si parla anche di tipologia di output, accenneremo anche a come sono definiti, in Forth, i formati (PRINT USING del Basic), raccomandando il rimando ai manuali di istruzione singoli, in quanto gli standard di tali operazioni sono molti e, talora, notevolmente diversificati.

Operazioni sui caratteri

La prima e principale operazione di I/O è quella che coinvolge un solo carattere, sia esso letto dalla tastiera o trasmesso alla stampante o al video, La word KEY (ricordate l'INKEY$) attende che l'indirizzo di input corrente (generalmente la tastiera od uno strumento di misura) fornisca un carattere, il cui codice ASCII viene posto in TOS. Al contrario, la word EMIT (ricordate le prime puntate, ne abbiamo già parlato) estrae un valore ASCII dallo stack e lo trasmette all'indirizzo di output (stampante o video).

Poiché la word KEY determina una attesa incondizionata fino alla fornitura di un carattere, battendo KEY, ed immediatamente RETURN, non si produrrà il solito messaggio di OK né sarà mostrato, poi, il valore del tasto quando questo sarà battuto. Vale a dire che battendo:

```
KEY (seguito da RETURN)
```

e

```
R     (seguito da RETURN)
```

otterremo il messaggio OK ma nient'altro. Occorrerà battere [.] per ottenere:

```
82 OK
```

ed infatti, nel set di caratteri ASCII, 82 equivale alla R maiuscola.

Ciò non è, evidentemente, molto pratico, a meno di non avere a portata di mano una tavola ASCII di rapida consultazione. È invece molto più semplice definire la word:

```
: GET       (in analogia a quanto)
            (presente in Basic)
  KEY
  EMIT
            (fine definizione)
;
```

Il fatto che la word KEY, da sola, non sia stampata, può essere utilizzato come parola d'ordine o codice di accesso ad un sistema. In controllo macchina, come ad esempio in un sistema d'allarme, l'inserimento di un carattere errato può portare allo scatto, ad esempio, di un allarme. Ad esempio:

```
: GGET      (controlla che il tasto premuto)
            (sia pari ad un codice)
            (in particolare A)
BEGIN
KEY 65=
UNTIL
            (fine definizione)
;
```

ha un compito molto ovvio, che non val la pena di descrivere.

Tale loop ha il difetto di bloccare la situazione finché non si batte A. Può essere più utile che il valore ottenuto con KEY sia confrontato con uno già presente nello stack (e messo, ad esempio, da un altro utente andato via appena prima). Ciò è utile, ad esempio, per inserire parole d'accesso ad un sistema, da mutare giorno per giorno. Avremo, così:

```
: GGGET     (controlla che il tasto premuto)
            (abbia codice eguale a quello)
            (del carattere presente nello)
            (stack)
BEGIN
DUP         (duplica il TOS)
KEY=        (legge un carattere da tastiera)
            (e ne verifica l'eguaglianza)
            (con quello in TOS)
UNTIL
DROP        (ad esito positivo, cancella lo)
            (stack)
.           "accesso consentito"
            (fine definizione)
;
```

Esistono inoltre in FORTH tre word utilizzate come sequenze operative. La prima, BELL (campana) invia un carattere BEL (ASCII 7) all'indirizzo di output, attivando un cicalino, campanella, o, comun-

que, avvisatore acustico, così come in altri linguaggi.

La seconda word, BL, lascia un carattere vuoto (BLANK, ASCII 32) in TOS. Esso opera in modo analogo alla pressione della barra spaziatrice in risposta a KEY. SPACE, la terza word, invia all'indirizzo di output un ASCII 32 *senza* inviare tale valore al TOS. SPACE, come CR (ve lo ricordate il carriage return?) è molto utile per separare i valori in uscita.

Operazioni sulle stringhe

Analogamente a quanto eseguito precedentemente con i caratteri, il programma può contenere istruzioni destinate ad aspettare una stringa dalla tastiera. Poiché il set di caratteri ASCII comprende tutto il set di caratteri disponibili, ivi compresi quelli di controllo, è possibile così ricevere da tastiera qualunque serie di caratteri, mediante due word appositamente dedicate, EXPECT e TYPE. Ambedue le word utilizzano due parametri di stack: un indirizzo di partenza della stringa, ed un numeratore di caratteri.

Spieghiamoci meglio. La sequenza:

```
200 25 EXPECT
```

del tipo ind n EXPECT

attende dalla tastiera una frase di 25 caratteri e la pone (con sequenze d'indirizzo crescenti) in memoria partendo dall'indirizzo 200. In effetti ciò è vero solo in parte, in quanto all'indirizzo 200 è posta la lunghezza (in decimale) della stringa (25) e la vera word inizia all'indirizzo 201. Intendiamoci; non è affatto necessario che la stringa sia davvero lunga 25 caratteri; essa *non può però essere più lunga*. Poiché, quindi, un byte può contenere il valore massimo di 255, è questa la lunghezza massima di una frase che può essere inserita senza particolari finezze di programmazione (tranne poi consentirne un allungamento con particolari tecniche di dimensionamento, necessarie in ogni linguaggio per stringhe ben più brevi).

Battendo alla tastiera una stringa di 25 caratteri, possono verificarsi due casi: che questa sia più corta di 25 e pertanto sarà necessario premere il RETURN o che questa sia eguale o più lunga degli n caratteri precedentemente indicati. Accade, così, nel primo caso, che la memoria contenga, nel byte 200 il valore 25 e nei successivi (con indirizzi crescenti) la stringa carattere per carattere, completando infine i posti non occupati con due caratteri NULL (ASCII 0). Viceversa, battendo dalla tastiera la stringa più lunga di 25 caratteri, il computer interromperà la lettura della tastiera dopo la pressione, appunto, del 25° tasto, non accettando, pertanto, altri caratteri e, diversamente da altri linguaggi, non aspettando la pressione del tasto RETURN.

La word TYPE effettua la operazione opposta di ESPECT. Essa trasmette all'indirizzo di output una stringa presente in memoria all'indirizzo ind, della lunghezza specificata in n. Essa si presenta strutturata in maniera analoga ad ESPECT, vale a dire:

```
200     25     TYPE
(ind    n      TYPE)
```

restituirà, sullo schermo o sulla stampante, la stringa immagazzinata precedentemente.

Sistemi di formattazione

Formattazione di stringhe.

Anche qui esistono diverse implementazioni di Forth, a seconda del sistema usato, della versione e del produttore della scheda o programma impiegato. Daremo qui solo le tipologie principali, presenti senz'altro in tutte le implementazioni, rimandando ai singoli manuali per le tipologie specifiche.

In aggiunta a CR, SPACE, BL e le altre word già nominate (non dimentichiamo la potenza del FORTH, che permette di creare nuove word partendo da quelle precedenti) il linguaggio ha altre due Word utili per formattare stringhe di testo. SPACES (che può essere ampiamente simulato con una struttura DO - LOOP e la word SPACE, ad esempio) non è altro che il multiplo di SPACE, che usa quale parametro numerico il TOS. Avremo che la sequenza:

```
." nome " 6 spaces ." indirizzo " 6 spaces ." telefono "
```

darà l'intestazione, formattata, di una rubrica telefonica.

TRAILING, un'altra word, sempre presente, elimina i caratteri bianchi di chiusura di una stringa. Esiste talora la word + TRAILING che, volendo, elimina anche caratteri BLANK compresi nella stringa stessa. In ogni caso ambedue le word agiscono solo sulla tipologia di output, senza mai intervenire sulla memoria.

Formattazione di numeri

Qui, ancora una volta, il FORTH dimostra il suo carattere scientifico evidenziando molte più word destinate a formattare le cifre (dall'incolonnamento a sinistra ed a destra all'inserimento di caratteri speciali in esso, quali segni e linee di separazione, punti decimali, segni di valuta, ecc). Poiché però anche qui si ha ampia variabilità di forme non entreremo nella discussione delle singole word, limitandoci ad un elenco di quelle più diffuse, utilizzate, e generalmente presenti su sistemi anche piccoli (implementazioni su Spectrum, QL, Commodore, tanto per intenderci); eccellente ad esempio, un FORTH, addirittura in virgola fluttuante, della CP Sfw, oppure l'ABERSOFT presentato da Bergami qualche mese fa.

WORD	Significato	Uso
.R	stampa il numero n, allineato a destra, secondo l'ampiezza imposta	n ampiezza ---
D.R.	idem, ma con numero doppio	d ampiezza ---
U.R.	idem, ma con numero non segnato	u ampiezza ---

Altre word consentono la trasformazione di stringhe in numeri e viceversa (VAL (A$) o VAL$ (A) del Basic), ma qui non ci dilungheremo perché la corona si sfila e non basterebbero 10 pagine per spiegarne l'uso.

Nuove operazioni su disco

Qualche puntata fa, abbiamo imparato a trasferire programmi da e sul disco, in unità di 1024 byte chiamate blocchi. Un blocco può essere formato da un testo (ad esempio una sequenza di definizione di una word, nel qual caso parliamo, più esattamente, di screen), ma può essere rappresentato anche da sequenze di dati diversi, come quelli presenti, ad esempio, in un data file. Avevamo a suo tempo imparato come conservare le definizioni; impareremo, adesso, come conservare informazioni o dati non rappresentanti un programma od una word.

Allorché si ha bisogno di utilizzare una certa area di memoria per inserirvi un testo o una serie di numeri, è necessario, inizialmente, allocare un buffer destinato ad accoglierli. La word BUFFER preleva un numero dal TOS e riserva un block buffer (1024 byte) di memoria per questo blocco. Se il precedente contenuto del buffer è stato affetto dalla word UPDATE, il sistema lo deposita prima su disco. Per conoscere, comunque, dove il block buffer è allocato in memoria, il sistema lascia in TOS l'indirizzo iniziale di memoria.

Ad esempio:

```
64 BUFFER U.
```

(ricordiamo che U. visualizza il valore di un numero senza segno) darà:

```
42022 OK
```

vale a dire, allocherà un block buffer per il blocco 64 in memoria, e mostrerà il suo indirizzo di partenza (in decimale; in esadecimale sarà A426). Poiché un blocco, come già detto è lungo 1024 byte, l'indirizzo finale sarà 42022 + 1024 = 43046.

Si noti, in ogni caso, come l'indirizzo di

memoria NON corrisponda al semplice prodotto del numero di blocco per 1024; l'indirizzo iniziale è scelto direttamente dal sistema tenendo conto delle aree che volta per volta ha a disposizione ed il sistema di allocazione non rispetta criteri di proporzionalità alcuna.

Esiste, ancora, la word BLOCK, simile a BUFFER, e che funziona analogamente, eccetto che BLOCK lavora su blocchi che sono stati già allocati e sono presenti sia in memoria che su dischetto.

Una volta che un blocco è stato salvato su dischetto, è possibile reinserirlo in memoria sempre con la word BLOCK. Ad esempio:

```
64 BLOCK
```

legge il contenuto del blocco 24 in memoria, se già non c'è, e lascia l'indirizzo in TOS.

Aggiungendo, evidentemente, un offset all'indirizzo di partenza, è possibile accedere ad ogni specifico dato presente in memoria.

Ad esempio, per inserire il contenuto del byte 100 del blocco 64 nello stack, sarà sufficiente eseguire:

```
64 BLOCK 2 + 100 + C@
```

dove C@ effettua una ricerca del byte meno significativo. Si assume che il primo byte di dati sul blocco sia 0 così che 100 è effettivamente il 101^ byte dati. Allo stesso modo, per cambiare il contenuto del byte 100 al valore 4 eseguiremo:

```
4 64 BLOCK 2 + 100 + C!
```

Evidentemente, se lavoriamo su numeri doppi, occorrerà raddoppiare lo scarto, aggiungendo all'indirizzo di partenza +2. Le sequenze precedenti diverranno:

```
64 BLOCK 2 + 100 2 * + @
4 64 BLOCK 2 + 100 2 * + !
```

(si noti l'uso di @ e di ! al posto di C@ e C!).

Per svincolarsi dalla cosa, e cioè avvisare il sistema che un blocco è stato cambiato, occorre far seguire, come al solito, l'ultimo cambio con UPDATE.

Infine, per chiudere l'argomento, è possibile duplicare un blocco. Ogni block buffer in memoria è preceduto da due byte. Il primo byte contiene il numero di blocco in cui il buffer è allocato. Il secondo contiene un indicatore di sequenza di update; questo indicatore è eguale ad 1 se sul blocco è stata eseguita una operazione di aggiornamento, altrimenti contiene 0. Cambiando il contenuto del primo byte, l'identificatore di blocco, è possibile duplicare un blocco, quando, ad esempio, ne serva una copia di riserva. La procedura di duplicazione è estremamente semplice: si altererà, semplicemente, l'identificatore di blocco, si setterà il flag di updating, e si scriverà il buffer su disco. Per esempio, se volessimo duplicare il solito blocco 64 nel 65 si eseguirà:

```
65 64 BLOCK 2 - C!
```

e poi

```
UPDATE SAVE-BUFFER
```

tecnica che può essere utilizzata convenientemente per eseguire copie di screen. Ciò può essere utile, ad esempio, quando una particolare definizione non funziona bene, e se ne vuole una copia per poterci lavorare sopra, pur mantenendo intatta la versione originale. MC