di Raffaello De Masi

Finestre e gestione dello schermo con Prolog

In questa puntata vedremo come manipolare alcuni tool non proprio alfanumerici, come il layout di schermo e le finestre; sebbene non si tratti di tool necessari in senso stretto, essi si prestano in maniera egregia a rendere più interattivo l'ambiente, ad avere un migliore controllo delle fasi di I/O e, in definitiva, a rendere più piacevole il lavoro all'utente finale. Vedremo per prima cosa, come controllare lo schermo; in altri termini come maneggiare il contenuto e l'aspetto dei dati mostrati sullo schermo, come eseguire un editing efficace, come modificare e maneggiare nella maniera più veloce il cursore, il tutto, il che non guasta, in relazione a finestre anche multiple, presenti sullo schermo e costruite dallo stesso utente. Successivamente vedremo, come sarà possibile assegnare a programmi TurboProlog un aspetto professional con la possibilità di renderli, come dicevamo in precedenza, facili da leggere e da usare. Addirittura sarà possibile penetrare nel mondo della grafica e del suono per rendere ancora più user-friend i programmi costruiti

In generale, in Turbo Prolog come in tutti gli altri linguaggi, in default le informazioni, sullo schermo, non sono mostrate in maniera molto sofisticata; ogni stringa, numero o altro occupano una riga fino al CR, e i caratteri sono mostrati bianco su nero (o con altri colori, a seconda di quelli scelti nella finestra di Setup presente al lancio).

Talvolta, comunque, può essere utile o addirittura necessario avere particolari forme di evidenziazione di informazioni sullo schermo; ad esempio è necessario inviare un messaggio da evidenziare in un riquadro di un certo colore, o magari è opportuno far rilevare, attraverso forme particolari di messaggio, che un particolare evento si è verificato (è il caso tipico degli avvisi d'errore). Ad esempio, per una macchina destinata al controllo di strumentazione, può essere necessario evidenziare sullo schermo che la stessa è indaffarata, per tot secondi, a leggere i trasduttori delle periferiche.

In questo caso è opportuno far ricorso a certe particolari prerogative di Turbo Prolog servendosi di predicati articolari per rendere più flessibile e efficace l'apparenza di informazioni che appaiono sullo schermo. I predicati a disposizione del programmatore per controllare queste possibilità sono descritti nella figura a. Descriveremo uno per uno questi predicati, con le loro specifiche d'uso.

Il predicato [attribute]

Si tratta dell'attributo ad azione più generale del gruppo; come si legge dalla didascalia della figura a, appunto, esso serve a definire lo schema dei colori di default per lo schermo. Esso maneggia un intero, che può essere anche, ovviamente, custodito in una variabile. Ma cosa significa questo intero?

La figura b evidenzia appunto il significato dell'argomento del valore associato a questo predicato. La prima parte corrisponde all'uso di uno schermo monocromatico, la seconda nel caso in cui [attribute] sia destinato a un display a colori.

Nel primo caso il significato è semplice; bianco, bianco su nero, nero su bianco (non è ovviamente possibile bianco su bianco); l'uso di [attribute] con monitor a colori è invece un tantino più complesso. Per eseguire la combinazione di colori desiderata, prima occorre stabilire i colori da usare, per lo sfondo e per la scritta; poi si cercano questi nello schemino allegato; la somma rappresenterà l'argomento, appunto, del predicato; nel caso si desideri avere un carattere lampeggiante sullo schermo, è sufficiente aggiungere 128 al risultato. È tutto!

Per avere un esempio d'uso del predicato, battiamo, in editor:

```
Goal: attribute(Attr).
Attr = 44
1 Solution
Goal:
```

col risultato di avere lo schermo con caratteri rossi su sfondo verde.

L'uso inverso del predicato (quello di forzare un determinato colore) è esemplificato di seguito:

```
Goal: attribute(72)
True
1 Solution
Goal:
```

Cosa succede? Immaginiamo di aver lavorato finora con il valore di 18 (blu su verde); passerà, a seguito del nuovo comando, a rosso su grigio; il sistema migliore d'uso di questo attributo è quello di creare un loop che evidenzi tutte le possibili combinazioni, abbinato a una serie di messaggi che evidenziano la combinazione e a una chiamata in Input, per consentire la visualizzazione di una stringa su sfondo, dei colori scelti. Ciò può essere utile per scegliere

Figura a
Predicati di controllo dello schermo.

attribute	definisce il colore di default per lo schermo
scr_attr	cambia gli attributi per un carattere
scr_char	scrive o legge un carattere
display	scrive una stringa
edit	chiama l'Editor
editmsg	esegue un editing di messaggio in fondo allo schermo
cursor	posiziona o cerca il cursore
cursorform	cambia di forma il cursore

il default di visualizzazione (passando all'editor e settando il default nella finestra di setup), alterando permanentemente l'ambiente di lancio.

Accanto a questo tipo di attributo, piuttosto generale, ce n'è un altro, [scr_attr] (Screen_character_attribute), che consente di definire o cambiare una combinazione di colori specifica di un particolare carattere. Questo predicato maneggia tre argomenti, tutti numeri interi; i primi due rappresentano l'individuazione (riga e colonna) della posizione del carattere (qualcosa di simile al LOCATE di BASIC), il terzo argomento è del tutto analogo a quello del caso precedente. Un esempio del tipo:

	nero	blu	verde	cyan	rosso	magenta	marrone	bianco
nero	0	16	32	48	64	80	96	112
grigio	8	24	40	56	72	88	104	120
blu	1	17	33	49	65	81	23	113
celeste	9	34	41	57	73	89	105	121
verde	2	18	34	50	66	82	98	114
verde legg.	10	26	42	58	54	90	106	122
cyan	3	19	35	51	67	83	99	115
cyan legg.	11	27	43	59	75	91	107	123
rosso	4	20	36	52	68	84	100	116
rosso legg.	12	28	44	60	76	92	108	124
magenta	5	21	37	43	69	85	91	117
magenta legg.	13	29	45	61	77	93	109	125
marrone	6	22	38	54	70	86	102	118
giallo	14	30	46	62	78	94	110	126
bianco	7	23	39	55	71	87	103	119
bianco brill.	,15	31	47	63	79	95	111	127

Figura b

```
Goal: scr_attr(5,8,72)
True
1 Solution
Goal:
```

trasformerà il carattere presente alla 5.a riga e all'8.a colonna in giallo su sfondo rosso. Quando si specificano, quindi, gli indirizzi di riga e colonna di una posizione di un carattere sullo schermo, ci si riferisce sempre alla posizione in alto a sinistra della finestra in cui appare. Lo spigolo in alto a sinistra è indirizzato (0,0). Ma in ogni caso per schermi differenti il numero di righe e colonne sarà diverso; ancora peggio avviene quando l'utilizzatore cambia la grandezza della finestra corrente durante il programma; ma di questo parleremo tra breve. Se si forniscono valori che capitano all'esterno della window viene generalmente restituito un messaggio d'errore.

Un predicato analogo a quello precedente è [scr_char] che permette di leggere quale carattere è presente in una particolare posizione (si tratta, in pratica, dell'inverso di quello precedente). Come questo, maneggia tre argomenti, che funzionano allo stesso modo che nel primo caso. La funzione, come prevedibile, restituisce un intero che rappresenta il valore ASCII del carattere presente alla locazione individuata dai primi due numeri.

Generalizzando, tutto dipende dal terzo membro del predicato; se esso è un valore definito, (o è un vero e proprio carattere ASCII, compreso tra singole virgolette ['']) il carattere stesso viene stampato (la locazione di schermo subisce una forzatura, nel secondo caso la locazione viene letta e le relative informazioni restituite); proprio l'uno il contrario dell'altro, e l'ago della bilancia è dato dalla definizione del terzo termine.

C'è solo da precisare l'obbligatorietà dell'uso della singola virgoletta ['], l'uso della doppia virgoletta qui non è ammesso, in quanto, in questo caso, quanto compreso tra esse viene visto come una stringa, non come un solo carattere, e la risposta è un messaggio d'errore.

L'uso del predicato [display]

Lavorare con un carattere alla volta può essere nella maggior parte dei casi seccante, e talora inutile; talvolta è necessario, o semplicemente più utile, disporre la lettura di una parte della riga. Anche a questo hanno pensato gli implementatori del linguaggio, che hanno inserito diversi predicati destinati a lavorare con caratteri in gruppo.

Quanto diremo di seguito rappresenta, quindi, una serie di tecniche destinate a leggere e identificare gruppi di lettere. Come nel caso precedente, tutti i predicati lavorano in default con la finestra attiva, ma esiste la possibilità di ridigere le operazioni su altre finestre anche lasciando attiva quella corrente. Ci arriveremo nel corso della discussione.

Cominciando dall'inizio, proprio dal facile facile, per scrivere nella locazione in alto a sinistra dello schermo (per essere precisi nella posizione dove si trova il cursore) è sufficiente usare il comando [display]. Dei quattro predicati destinati a questo genere di operazioni, questo è il più facile da usare, essendo privo, tra l'altro, di informazioni relative al punto dove la stringa va posizionata. [display] usa solo un valore, la stringa da scrivere, che, ovviamente, può essere rappresentata anche da una variabile. L'esempio più semplice è:

```
Goal: clearwindow.
True
Goal:display("Hello!").
```

Succedono tre cose diverse; primo, il testo del messaggio è riportato alla prima linea della finestra, secondo il cursore non passa alla linea appresso, ma resta in alto a sinistra, in corrispondenza della lettera (H), terzo proprio per la posizione assunta dal cursore è possibile lavorare direttamente sul testo.

Attenzione, però, lavorare significa muoversi; leggere lettere o stringhe non è possibile. Sotto questo punto di vista [display] non serve poi a molto.

L'uso di [display], comunque, non è certamente elastico; ben altro è possibile fare con altri predicati, che permettono di lavorare con campi di dati; essi sono, nell'ordine, field_attr e field_str. Ambedue manipolano quattro argomenti; i primi due, come al solito contengono la combinazione di indirizzo linea-colonna, corrispondente alla posizione di partenza del campo. Il terzo è la lunghezza della stringa, in caratteri.

Fin qui i due predicati manipolano gli stessi tre argomenti. Il quarto li differenzia in maniera abbastanza netta, specializzandoli. Nel caso di field_attr, l'argomento è un intero costruito in accordo alla tabella B, nel caso di field_str esso è una stringa, o una variabile di stringa, che, ovviamente, rappresenta il "materiale" da stampare.

Il vero uso dei due predicati appare chiaro quando sia necessario eseguire un output formattato in maniera efficace.

L'ultimo predicato di questa puntata è window_str, che fornisce informazioni circa la window in un modo particolare. Esso utilizza una stringa come argomento, e mostra informazioni nella finestra corrente.

La stringa fornita come argomento occupa l'intera finestra. Se questo non avviene, per difetto di caratteri, il riempimento viene eseguito con caratteri bianchi, se invece la stringa è troppo lunga essa viene troncata alla bisogna.

E anche stavolta abbiamo concluso con i nostri predicati; la prossima ci interesseremo di editing dello schermo; a risentirci!