di Raffaello De Masi

Stringhe e conversione di tipi

Come già dicemmo la volta scorsa, parleremo in questa puntata di stringhe e di conversione numerica-alfanumerica. Cosa vuol dire è presto detto se si tien conto di quanto già esiste negli altri linguaggi programmatori, primo tra tutti il Basic con almeno cinque o sei operatori destinati allo scopo (quando non si va nel Basic HP dove se ne contano almeno una ventina). Parliamo di comandi, in Basic, come Left$, STR$, VAL$ e così via, tanto comodi per rendere la vita facile al programmatore. Ma non basta, se teniamo conto che per un linguaggio come il Prolog che ha bisogno di entrare davvero molto all'interno delle strutture degli oggetti (stringhe alfanumeriche e numeri) destinati a manipolare, abbisogna di operatori efficienti per selezionare con il più basso margine d'errore dati all'interno di un database. Prolog fornisce al programmatore una serie di operatori di buon livello destinati a manipolare le stringhe e ad estrarre dati da esse. Inoltre, cosa altrettanto utile, fornisce operatori che fungono da tramite e da tool di conversione tra catene alfanumeriche e numeri, consentendo, appena possibile, una opportuna traduzione da una all'altro e viceversa. Inoltre, ma si tratta di comandi già visti altrove, il nostro linguaggio fornisce operatori destinati a convertire caratteri nei loro corrispondenti codici ASCII e viceversa

Ricapitolando

Qualche puntata fa, quando abbiamo parlato delle stringhe, abbiamo enunciato alcune regole e principi, oltre a dare delle definizioni formali. È il caso di riprenderle.

Una stringa è una sequenza di lettere, numeri, simboli speciali e spazi, definita morfologicamente e con una sua struttura alfanumerica. Inizia generalmente con una lettera minuscola. Se inizia con una maiuscola o se contiene spazi, deve essere «delimitata», definita da vigolette ["] all'inizio e alla fine; viceversa se in qualche modo è individuabile come una sola parola, le virgolette sono superflue.

Esse sono classificate, in Prolog Standard, nel tipo Domain [Dominio]. Della differenza tra stringhe e simboli abbiamo parlato diffusamente in precedenza; agli atti pratici ambedue possono essere usati in maniera intercambiabile. Ad esempio, molti dei predicati descritti nelle puntate precedenti possono essere usati egualmente come stringhe e simboli.

Le istruzioni di base

Prolog fornisce due predicati di base che consentono di investigare la natura di una stringa o di un simbolo. Si tratta di due facility più che di vere e proprie utility e rappresentano ambedue dei determinatori circa la natura e la utilizzabilità di una stringa.

Il primo operatore determina la lunghezza di una stringa-simbolo, il secondo risponde alla domanda: «È possibile usare un certo insieme di caratteri come un valido nome per la stringa stessa?». Questa utility, di cui può sembrare apparente l'inutilità, diviene necessaria nel caso l'utilizzatore, ad esempio debba introdurre un valore che il programma deve usare come nome, o se si sta leggendo un file da disco e si desidera sapere se è stato letto, un nome valido prima di procedere.

Il primo predicato già presente nel linguaggio è [str_len] che informa di quanti caratteri, spazi e simboli speciali è formata o il simbolo su cui si sta lavorando. Il predicato accetta due argomenti; il primo è la stringa vera e propria su cui l'investigazione avrà luogo, il secondo è una variabile numerica (intera) nella quale sarà conservato il risultato della ricerca.

Un esempio dell'uso del predicato sarà:

```
Goal: str_len("Ditegli sempre di sì",Lunghezza)
Lunghezza=20
1 Solution
Goal:
```

Circa l'uso di tale predicato, esso è tanto evidente che c'è ben poco da dire; nella manipolazione di file sequenziali esso trova la sua ragion d'essere, ma anche in applicazioni di tutti i giorni, come word-processing o database senza di esso le cose non sarebbero tanto facili.

L'altra informazione utile descritta in precedenza può essere raccolta attraverso il predicato [isname], stavolta senza l'underscore, chissà perché. Il predicato maneggia una stringa o un simbolo come argomento e restituisce una risposta logica, [True] se l'argomento è un nome accettato e valido, [False] se viene rifiutato. Questo predicato è inutilizzabile per i simboli, sebbene le regole per nominare simboli e stringhe siano le stesse. Ogni valore che sia identificato come simbolo è anch'esso un nome valido.

Di seguito diamo una serie di esempi dell'uso del predicato [isname]:

```
Goal: isname("nome").
True
Goal: isname("3a").
False
Goal: isname("nome_della_stringa").
True
Goal: isname("nome della stringa").
False
```

Nel secondo e nel quarto caso il nome proposto non è utilizzabile perché, nel primo caso inizia con una cifra numerica, nel secondo caso contiene degli spazi (che se annullati dall'underscore, come nel terzo esempio, sono ammessi).

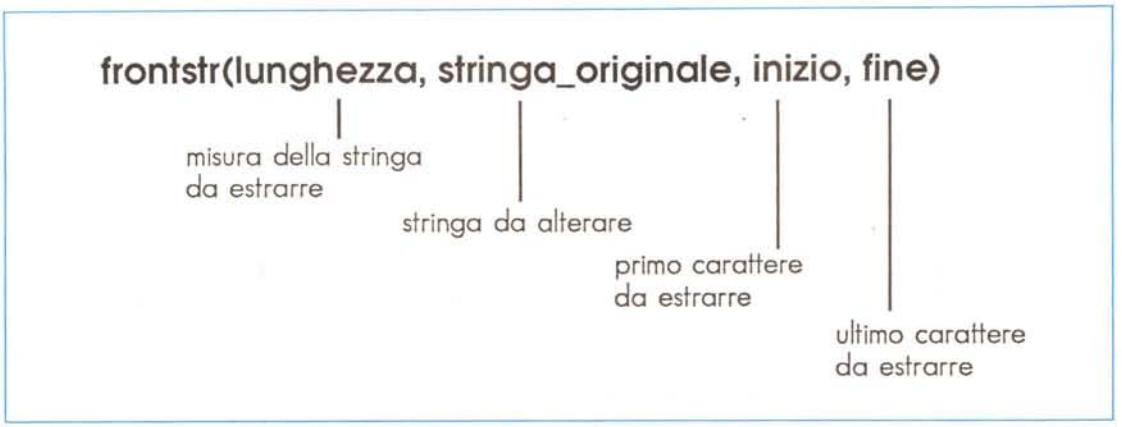

Figura a. Tipologia del predicato [frontstr].

Operatori più complessi

È possibile estrarre da una stringa (o simbolo) caratteri a nostro piacimento utilizzando una serie di comandi dedicati. Vediamone qualcuno.

Il primo carattere di una stringa può essere estratto utilizzando il predicato [frontchar]; questo predicato richiede tre argomenti, rispettivamente una stringa, un carattere e una stringa nell'ordine. Il primo argomento è la stringa su cui si desidera lavorare, il secondo è il carattere che si desidera estrarre, il terzo è la parte di stringa restante.

```
Goal: frontchar("Stringa",A,Resto)
A=S, Resto=tringa
1 Solution
Goal: frontchar("10_e_lode",A,Resto)
A=1, Resto=10_e_lode
1 Solution
Goal: frontchar("Questa è una stringa",A,Re-
sto)
A=Q, Resto=uesta è una stringa
1 Solution
```

È possibile usare in maniera diversa questo predicato, invertendo opportunamente l'ordine degli argomenti; avremo ad esempio:

```
Goal: frontchar(A,,"A,"stringa")
A=Astringa
1 Solution
```

La cosa può tornare utile, ad esempio, per aggiungere lettere all'inizio di una stringa già conformata.

E andiamo avanti, sempre più. È possibile dividere una stringa in due tronconi, con il predicato [frontstr]; esso è equivalente al predicato [frontchar] appena visto tranne che maneggia stringhe pluricarattere. Esso ha quattro argomenti invece di tre, ed è organizzato come in figura a. Il predicato estrae la prima «misura» di carattere dalla stringa originale, mettendo i caratteri estratti in [inizio] e la rimanente parte della stringa in [fine]. Un esempio può essere rappresentato da:

```
Goal: frontstr(4,"le voci di dentro",A,B).
A=le v, B = oci di dentro
1 Solution
Goal: frontstr(24,"le voci di dentro",A,B).
No Solution
Goal:
```

Alla seconda richiesta non c'è risposta in quanto il numero dei caratteri imposti come primo argomento eccede la lunghezza della stringa stessa.

E ancora, può accadere che una stringa contenga informazioni che desideriamo estrarre indipendentemente dal sapere dall'inizio quanti caratteri occupi una particolare porzione di stringa stessa. Un esempio può essere rappresentato da un inventario dove la prima parte della stringa è rappresentata da un numero d'ordine e la seconda dal tipo di materiale di cui si tratta. In questo caso non si sa bene quanti caratteri saranno occupati, all'inizio, per il numero d'ordine.

Esiste un comando molto particolare, in TurboProlog, non facilmente reperibile altrove, dal nome caratteristico [frontoken]. Esso accetta tre argomenti: il primo è la stringa su cui lavorare, il secondo è il "token" che si desidera estrarre, il terzo è quanto rimane della stringa dopo l'estrazione del token stesso. Un esempio servirà, come al solito a chiarire il tutto.

```
Goal: fronttoken("122scarpe",Numero,Tipo)
Numero = 122, Tipo = scarpe
Goal: fronttoken("87stampante",Numero
,Tipo)
Numero = 87, Tipo = stampante
Goal: fronttoken("12255quaderno",Numero
,Tipo)
Numero = 12255, Tipo = quaderno
```

L'esempio credo sia valso più di ogni spiegazione; ma cosa è un token? Semplice, è un gruppo di uno o più caratteri che obbediscono a una di queste condizioni:

- costituiscono un valido nome per Prolog;
- rappresentano un valido intero o un numero reale in forma di stringa;
- costituiscono un singolo carattere valido ad esclusione di uno spazio.

A questo punto gli esempi sono davvero ben chiari e funzionanti. [frontoken] estrae i numeri dalla stringa in quanto rappresentano validi interi (e soddisfano quindi alla seconda condizione). Purtroppo questo operatore non funziona con i simboli, in quanto essi, come abbiamo visto in precedenza rappresentano validi nomi e quindi non c'è proprio nulla da estrarre da essi.

Due tipi di predicati che modificano le stringhe sono ancora disponibili in TurboProlog (finora avevamo solo estratto). Vediamoli uno per uno:

[upper_lower] converte una stringa da maiuscola a minuscola, come si vede negli esempi seguenti:

```
Goal: upper_lower("STRINGA",S)
s = stringa
1 Solution
Goal: upper_lower("STringa",S)
s = stringa
1 Solution
Goal: upper_lower("strINGA",S)
s = STRINGA
1 Solution
Goal: upper_lower("StRiNga",S)
s = stringa
1 Solution
Goal: upper_lower("sTRINGA",S)
s = Stringa
1 Solution
```

Il primo argomento della stringa, come si può vedere dagli esempi, determina l'inversione della stringa totale, senza che venga tenuto in alcun conto

tutto quello che c'è successivamente. Ovviamente numeri e altri caratteri sono lasciati intatti; ove mai un numero capitasse al primo posto della stringa la stringa stessa viene scorsa fino a trovare la prima lettera disponibile per eseguire il confronto e la trasformazione.

E passiamo alla trasformazione di stringhe in numeri; TurboProlog fornisce tre operatori destinati a lavorare a questo scopo: [str_int] che esegue conversioni tra stringhe e interi, [str_real] che esegue conversioni tra stringhe e numeri reali, [str_char] che esegue conversioni tra stringhe e caratteri. Poiché come sappiamo e abbiamo più volte detto le stringhe sono largamente intercambiabili con i simboli, questi tre predicati rendono possibile traslare dati da qualunque Dominio.

Tutti e tre i predicati lavorano allo stesso modo; ognuno richiede due argomenti: il primo è la stringa da convertire, il secondo è il tipo di variabile in cui va acconciamente eseguita la conversione. Ancora, come al solito, un esempio:

```
Goal: str_int (A,10)
A=10
1 Solution
Goal: str_int ("45",A)
A=45
1 Solution
Goal: str_int ("Paglia",A)
No Solution
Goal
```

La terza parte dell'esempio è di facile comprensione se si tien conto che "Paglia" non viene interpretato come numero.

```
Goal: str_char ("Paglia",A)
No Solution
Goal: str_char ("a",A)
A=a
1 Solution
Goal: str_real ("45.41",A)
A=45.41
1 Solution
Goal: str_real ("45",A)
A=45
1 Solution
```

E stiamo per concludere; l'ultimo predicato che studieremo in questa puntata equivale, più o meno al ASC$ del Basic; esso individua il valore ASCII del carattere analizzato. Esempio:

```
Goal: char_int ("D",A)
A=68
1 Solution
Goal: char_int (A,66)
A=B
1 Solution
```

Il primo argomento è il carattere da convertire, il secondo è un intero compreso tra 0 e 255; numeri più grandi non creano un errore, ma la conta inizia daccapo (si esegue tanto per intenderci un "fuori-255").

E anche stavolta abbiamo finito, a Dio piacendo. È giunto il momento adesso di fare un passo all'esterno e di vedere le tecniche di I/O di cui il Nostro dispone; vedremo le prossime volte come far dialogare il linguaggio con il mondo degli umani, per fargli manipolare dischi e file nel miglior modo e per far chiedere all'utente quello di cui il programma ha bisogno; a risentirci! MC