Laboratorio di Linguaggi di Sistema – a.a. 2005/2006 – Progetto finale III

Il seguente testo è valido sia come progetto base*, ignorando il materiale nei riquadri, sia come* progetto avanzato*, includendo il materiale nei riquadri. Il progetto base è indirizzato alla maggioranza degli studenti; quello avanzato è destinato a chi aspiri al massimo della votazione. La scelta del progetto base non pregiudica le possibilità dello studente di raggiungere un buon risultato, ma in tal caso ci si attenderà un orale eccellente. Le modalità di consegna sono identiche per entrambi i casi. Lo studente indichi chiaramente nella relazione se ha inteso sviluppare il progetto base o quello avanzato.*

Il problema

Si vuole realizzare un *valutatore di espressioni in notazione prefissa*. La notazione prefissa prevede che l'operatore da applicare sia indicato per primo, seguito dal numero di operandi appropriato. Per esempio, "2+3" in notazione prefissa si scrive "+ 2 3", mentre "7*(a+2)" diventa "* 7 + a 2". Si noti che la notazione prefissa non richiede parentesi[1], e che la valutazione procede da sinistra a destra.
Il valutatore di espressioni consiste di un unico programma, di nome `valuta`, che implementa le seguenti sintassi:

1. `valuta` "*espressione*" – riceve un singolo argomento su riga di comando, eventualmente delimitato da virgolette se richiesto dalla Shell in uso, valuta il suo valore come espressione prefissa, e stampa il risultato su *stdout*.
2. `valuta` op_1 op_2 ... op_n – riceve un numero arbitrario di argomenti, che saranno operandi o operatori, e valuta l'intera serie di argomenti come espressione prefissa, stampando il risultato su *stdout*. La Shell in uso potrebbe richiedere l'uso di apici, virgolette o caratteri di escape per preservare alcuni simboli (per esempio, "*").
3. `valuta -q` "*espressione*" o `valuta -q` op_1 op_2 ... op_n – analogo ai due precedenti, ma anziché stampare il risultato su *stdout*, lo restituisce come codice di ritorno del programma, approssimato ad intero.
4. `valuta` – senza argomenti, legge delle righe di testo da *stdin*, valuta ogni riga come un'espressione prefissa, e stampa su *stdout* una riga contenente il solo risultato dell'espressione. Il processo viene ripetuto finché vengono fornite righe su *stdin*.
5. `valuta -f` $file_1$ ... $file_n$ – legge le righe di testo dai file indicati (in ordine), valuta ogni riga di ciascun file come un'espressione prefissa, e stampa su *stdout* i risultati.

Linguaggio delle espressioni

Le espressioni che vogliamo valutare constano dei seguenti elementi:

- costanti numeriche, espresse come numeri interi o in virgola mobile (es: 194000, 212.53).
- variabili, il cui nome segue le solite regole: un carattere alfabetico, seguito da zero o più caratteri alfanumerici (es: `a`, `nett1`, `a2i`); sono esclusi i nomi di variabili identici agli operatori speciali indicati sotto. Il valore di una variabile è dato dalle istruzioni elencate nel seguito; l'uso di una variabile a cui non è assegnato un valore costituisce un errore.
- operatori aritmetici, con il significato che gli stessi simboli hanno in C: `+`, `-`, `*`, `/`, `%`, `&`, `|`, ecc.
- operatori logici e relazionali, con il significato che gli stessi simboli hanno in C: `&&`, `||`, `!`, ==, ecc. (si usi la convenzione: 0=FALSE, qualunque altro valore=TRUE).
- i seguenti "operatori" speciali:
  - **let** *var* $expr_1$ $expr_2$ – valuta $expr_1$, quindi valuta $expr_2$, in cui ogni istanza della variabile *var* viene valutata come il risultato di $expr_1$; il risultato della valutazione di $expr_2$ diventa il risultato di tutta l'espressione **let** (es: `let a 3 + a a` vale 6).
  - **if** $expr_1$ $expr_2$ $expr_3$ – valuta $expr_1$; se il risultato della valutazione è TRUE (ovvero, diverso da 0), valuta e restituisce $expr_2$, altrimenti valuta e restituisce $expr_3$. (es: `if > a b a b` restituisce il massimo fra *a* e *b*).
  - **defun** *var*(*var*1, ..., var_n) $expr_1$ $expr_2$ – definisce una funzione (con zero o più argomenti). Tutte le volte che l'espressione *var*(arg_1, ..., arg_n) compare all'interno di $expr_2$, dove arg_1 ... arg_n sono espressioni, viene valutata $expr_1$, al cui interno var_1 ... var_n sono sostituiti, rispettivamente, da arg_1... arg_n. Il risultato di questa valutazione è il valore di *var*(arg_1, ..., arg_n). Tutta l'espressione **defun** ha come valore il risultato della valutazione di $expr_2$ secondo le regole indicate (es: `defun max(a,b) if > a b a b + 1 max(3,10)` ha come valore 11). Si noti che una **defun** che definisca una funzione con 0 argomenti è del tutto analoga a una **let**.
  - **print** *expr* – stampa il valore di *expr* su *stdout*, fra quadre, e lo restituisce come proprio risultato. Se è stata indicata l'opzione `-q` sulla riga di comando, non stampa nulla e si limita a restituire il risultato (**print** *expr* in questo caso è equivalente a *expr*). Vedi sotto per un esempio.

1 a condizione che sia sempre possibile distinguere lessicalmente fra operatori ed operandi: se fosse permesso avere sia un'operatore sia una variabile chiamate "add", la notazione sarebbe ambigua.

- **for** *var* $expr_1$ $expr_2$ $expr_3$ $expr_4$ – valuta $expr_4$ per ogni valore di *var* compreso fra $expr_1$ e $expr_2$, inclusi, a passi di $expr_3$. Restituisce come proprio risultato l'ultimo valore di $expr_4$ valutato, o 0 se $expr_4$ non è mai stato valutato (es: `for a 1 6 2 + 1 print * 2 a` stampa su *stdout* la sequenza `[2] [6] [10]` e ha come risultato 11).
- **while** $expr_1$ $expr_2$ – valuta $expr_1$; se il risultato della valutazione è TRUE (ovvero, diverso da 0), valuta $expr_2$, poi torna a valutare l'intera espressione **while**; se invece il risultato è FALSE (ovvero, uguale a 0), termina la valutazione restituendo l'ultimo valore valutato di $expr_2$ (o 0 se $expr_2$ non è mai stato valutato) (si veda la sezione Esempi per alcuni esempi di uso di **while**).
- **input** – il valore dell'espressione è letto da *stdin* (su una riga separata); letture successive leggeranno valori distinti (si veda la sezione Esempi per alcuni esempi di uso di **input**).

- espressioni fra parentesi, usate solo per chiarezza: il valore di **(** *expr* **)** è quello dato dalla valutazione di *expr* (es: `( * 2 ( + 5 1 ) )` vale 12, esattamente come `* 2 + 5 1`). Si noti che le parentesi costituiscono token distinti, separati dagli altri con degli spazi.

Tempi e modalità di consegna

L'elaborato deve essere consegnato **improrogabilmente** entro il 26 Settembre 2006 e deve essere costituito da una stampa del codice sorgente, adeguatamente formattato e commentato, da una stampa con esempi di esecuzione, e da una breve relazione scritta (2-3 pagine) che descrive il progetto stesso. La versione a stampa può essere consegnata direttamente al docente (stanza 333 del Dipartimento di Informatica), oppure depositata presso il centralino del Dipartimento di Informatica (casella della posta "GERVASI") **entro la data indicata**. Tutto il materiale deve essere *anche* inviato via email al docente (gervasi@di.unipi.it) contestualmente alla consegna della parte cartacea. Si raccomanda di usare il subject "`Consegna progetto LLS-3`" nella mail con cui si invia il materiale.

Altre informazioni

Il programma deve gestire correttamente le situazioni eccezionali, quali ad esempio mancanza di memoria, impossibilità di accedere a certi file, divisioni per 0, errori di sintassi, ecc. Gli eventuali errori devono essere segnalati all'utente attraverso messaggi su standard error. In nessun caso il programma deve inviare su standard output materiale diverso da quanto indicato nel testo. L'esecuzione deve essere ragionevolmente efficiente.

Esempi

Le seguenti espressioni devono produrre i risultati indicati, indipendentemente dal modo in cui sono passate al programma `valuta` (riga di comando con un solo argomento, con più argomenti, da file, da *stdin*, ecc.).

- `let a 5.5 let b * 3.1 a + a b` → 22.55.
- `* + 1 3 + 2 5` → 28.
- `let x input + x 1` → legge da *stdin* un numero, restituisce il successore.
- `while let x input && >= x 0 < x 10 print 1` → legge da *stdin* un numero; se il numero letto è compreso fra 0 (incluso) e 10 (escluso), stampa "1" e torna a leggere un altro numero, altrimenti termina restituendo 1 (o 0 se il primo numero letto era già fuori dall'intervallo).
- `defun f(x) ( if ( > x 1 ) ( * x f(- x 1) ) ( 1 ) ) f(input)` → legge da *stdin* un numero e restituisce il suo fattoriale. Si noti che le parentesi usate per leggibilità sono fra spazi, mentre quelle che fanno parte della sintassi della **defun** seguono immediatamente il nome della funzione definita/invocata.
- `defun add(x,y) ( + x y ) print add(input,input)` → legge da *stdin* due numeri e ne stampa (e restituisce) la somma.
- `let x input let y input + x y` → legge da *stdin* due numeri e ne restituisce la somma.
- `let x input ( let y input ( if ( > x y ) ( x ) ( y ) ) )` → legge da *stdin* due numeri e restituisce il maggiore fra i due.
- `let x ( + 5 3 )` → genera un errore di sintassi: manca $expr_2$.
- `if > a 1 0 1` → genera un errore di sintassi: la variabile *a* non è definita.
- `let a input ( if ( % a 2 ) ( + a 1 ) ( a ) )` → legge da *stdin* un numero; se è pari, lo restituisce, se è dispari restituisce il pari successivo.
- `let a input ( if ( % a 2 ( + a 1 ) ( a ) )` → genera un errore di sintassi (parentesi non bilanciate).
- `defun fibo(x) ( if ( > x 2 ) ( + fibo(- x 1) fibo(- x 2) ) ( 1 ) ) let t input ( for n 1 t 1 print fibo(n) )` → stampa la tabella dei numeri di Fibonacci, da 1 fino al numero letto da *stdin*; restituisce l'ultimo numero di Fibonacci generato.