Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
http://courses.softlab.ntua.gr/pl2/

Γλώσσες Προγραμματισμού II

Οι ασκήσεις πρέπει να παραδοθούν στους διδάσκοντες σε ηλεκτρονική μορφή μέσω του συνεργατικού συστήματος ηλεκτρονικής μάθησης `moodle.softlab.ntua.gr`. Η προθεσμία παράδοσης θα τηρείται αυστηρά. Έχετε δικαίωμα να καθυστερήσετε το πολύ μία άσκηση.

Άσκηση 5 Συστήματα τύπων — μηχανή στοίβας

Προθεσμία παράδοσης: 21/4/2014

Τις περισσότερες φορές τα συστήματα τύπων συνοδεύουν γλώσσες στις οποίες οι εκφράσεις κατέχουν σημαντικό ρόλο, π.χ. προστακτικές ή συναρτησιακές γλώσσες προγραμματισμού. Στην άσκηση αυτή θα δούμε ότι είναι χρήσιμα και για αρκετά διαφορετικές γλώσσες, όπως π.χ. για γλώσσες χαμηλού επιπέδου που περιγράφουν τη λειτουργία μίας *μηχανής στοίβας* (stack machine).

Έστω η γλώσσα τα προγράμματα p της οποίας ορίζονται από την παρακάτω γραμματική, όπου n είναι φυσικός αριθμός και όλα τα υπόλοιπα (πλην των p, φυσικά) είναι τερματικά σύμβολα.

$$\begin{aligned} p ::= {} & n \mid \textbf{true} \mid \textbf{false} \mid + \mid - \mid * \mid / \mid < \mid = \mid \textbf{and} \mid \textbf{not} \\ \mid {} & \textbf{nop} \mid \textbf{dup} \mid \textbf{pop} \mid \textbf{swap} \mid \textbf{swap2} \mid p_1\ p_2 \mid \textbf{cond}\,[p_1 \mid p_2] \mid \textbf{loop}\,[p] \end{aligned}$$

Λειτουργική σημασιολογία: Ο τρόπος εκτέλεσης των προγραμμάτων σε αυτή τη γλώσσα δίνεται από τους παρακάτω κανόνες. Η κατάσταση της αφηρημένης μηχανής περιγράφεται από από ένα ζεύγος (p, σ), όπου p το προς εκτέλεση πρόγραμμα και σ μία στοίβα που περιέχει ακέραιους αριθμούς ή λογικές τιμές. Η σχέση αποτίμησης συμβολίζεται με $p; \sigma \longrightarrow p'; \sigma'$ που σημαίνει ότι αν το πρόγραμμα p εκτελεστεί με αρχική στοίβα σ, μετά από ένα υπολογιστικό βήμα θα μένει να εκτελεστεί το πρόγραμμα p' και τα περιεχόμενα της στοίβας θα είναι σ'. Ως συντομογραφία, γράφουμε $p; \sigma \longrightarrow \sigma'$ αντί για $p; \sigma \longrightarrow \textbf{nop}; \sigma'$. Με $\sigma \cdot v$ συμβολίζουμε την τοποθέτηση της τιμής v στην κορυφή της στοίβας σ.

$$n; \sigma \longrightarrow \sigma \cdot n \qquad \textbf{true}; \sigma \longrightarrow \sigma \cdot \text{true} \qquad \textbf{false}; \sigma \longrightarrow \sigma \cdot \text{false}$$

$$+; \sigma \cdot n_1 \cdot n_2 \longrightarrow \sigma \cdot (n_1 + n_2) \qquad -; \sigma \cdot n \longrightarrow \sigma \cdot (-n) \qquad *; \sigma \cdot n_1 \cdot n_2 \longrightarrow \sigma \cdot (n_1 \times n_2)$$

$$\frac{n_2 \neq 0 \qquad (q, r) = \text{quotRem}(n_1, n_2)}{/; \sigma \cdot n_1 \cdot n_2 \longrightarrow \sigma \cdot q \cdot r} \qquad \begin{aligned} <; \sigma \cdot n_1 \cdot n_2 &\longrightarrow \sigma \cdot (n_1 < n_2) \\ =; \sigma \cdot n_1 \cdot n_2 &\longrightarrow \sigma \cdot (n_1 = n_2) \end{aligned}$$

$$\textbf{and}; \sigma \cdot b_1 \cdot b_2 \longrightarrow \sigma \cdot (b_1 \wedge b_2) \qquad \textbf{not}; \sigma \cdot b \longrightarrow \sigma \cdot (\neg b) \qquad \textbf{dup}; \sigma \cdot v \longrightarrow \sigma \cdot v \cdot v$$

$$\textbf{pop}; \sigma \cdot v \longrightarrow \sigma \qquad \textbf{swap}; \sigma \cdot v_1 \cdot v_2 \longrightarrow \sigma \cdot v_2 \cdot v_1 \qquad \textbf{swap2}; \sigma \cdot v_1 \cdot v_2 \cdot v_3 \longrightarrow \sigma \cdot v_3 \cdot v_1 \cdot v_2$$

$$\textbf{nop}\ p; \sigma \longrightarrow p; \sigma \qquad \frac{p_1; \sigma \longrightarrow p_1'; \sigma'}{p_1\ p_2; \sigma \longrightarrow p_1'\ p_2; \sigma'} \qquad \textbf{cond}\,[p_1 \mid p_2]; \sigma \cdot \text{true} \longrightarrow p_1; \sigma$$

$$\textbf{loop}\,[p]; \sigma \cdot \text{true} \longrightarrow p\ \textbf{loop}\,[p]; \sigma \qquad \textbf{loop}\,[p]; \sigma \cdot \text{false} \longrightarrow \sigma \qquad \textbf{cond}\,[p_1 \mid p_2]; \sigma \cdot \text{false} \longrightarrow p_2; \sigma$$

Η εκτέλεση ενός προγράμματος αρχίζει με κενή στοίβα ($\emptyset$) και τερματίζεται όταν το πρόγραμμα γίνει ίσο με **nop**. Η εκτέλεση "κολλάει" αν τα τελούμενα στην κορυφή της στοίβας είναι διαφορετικού είδους από αυτά που απαιτεί μία εντολή, ή αν η εντολή απαιτεί τελούμενα αλλά η στοίβα είναι άδεια.

Παράδειγμα: Για το παρακάτω πρόγραμμα p, ισχύει $p; \emptyset \longrightarrow^{+} 42$ σε 75 βήματα (βλ. παράρτημα).

$$p \equiv 1\ 3\ \textbf{true loop}\ [\textbf{dup}\ 2 <\ \textbf{cond}\ [\textbf{false} \mid \textbf{dup}\ 1 -\ +\ \textbf{swap2}\ *\ \textbf{swap true}]]\ \textbf{pop dup}\ 1\ +\ *$$

Ζητείται: Ορίστε ένα σύστημα τύπων για την παραπάνω γλώσσα, τέτοιο ώστε τα προγράμματα που δεν έχουν σφάλματα τύπων να μην είναι δυνατό να κολλήσουν κατά την εκτέλεση (με μοναδική εξαίρεση τη διαίρεση με το μηδέν). Συγκεκριμένα:

1. Περιγράψτε τη μορφή της σχέσης τύπων που θα χρησιμοποιήσετε.
2. Ορίστε τη σχέση τύπων με ένα κατάλληλο σύνολο κανόνων.

Δώστε ιδιαίτερη προσοχή στην ακρίβεια των τυπικών ορισμών σας. Δε χρειάζεται να αποδείξετε την ορθότητα του συστήματός σας, δηλαδή να το συσχετίσετε με τη λειτουργική σημασιολογία, ούτε να υλοποιήσετε ελεγκτή τύπων ή διερμηνέα για τη γλώσσα (εκτός αν θέλετε να ελέγξετε κατά πόσο το σύστημα τύπων που προτείνετε ανταποκρίνεται στις προσδοκίες σας). Αν χρησιμοποιήσετε σχετικές εργασίες τρίτων που βρήκατε π.χ. στο διαδίκτυο, μην ξεχάσετε να αναφέρετε λεπτομερώς τις πηγές σας.

Μερικές υποδείξεις:

1. Δεν υπάρχει μοναδική σωστή λύση για αυτή την άσκηση.
2. Αυτό που μας ενδιαφέρει φυσικά είναι οι τύποι των περιεχομένων της στοίβας.
3. Κάθε εντολή απαιτεί συγκεκριμένους τύπους για τα περιέχομενα n θέσεων στην κορυφή της στοίβας και, όταν εκτελεστεί, τα περιεχόμενα αυτά έχουν αντικατασταθεί από m θέσεις των οποίων οι τύποι είναι γνωστοί. Γενικά, $n \geq 0$ και $m \geq 0$.
4. Ιδιαίτερη προσοχή χρειάζονται οι εντολές **cond** και **loop**. Είναι πιθανό να χρειαστεί να είστε συντηρητικοί στις επιλογές σας, για να πετύχετε το ζητούμενο.

Παράρτημα: Η εκτέλεση του παραδείγματος

Χάριν συντομίας, έχουν παραλειφθεί τα βήματα απλοποίησης της εντολής **nop**. Στις γραμμές με εσοχή γράφονται τα περιεχόμενα της στοίβας.

$\emptyset$
1 3 true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  1
3 true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  3 · 1
true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  true · 3 · 1
loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  3 · 1
dup 2 < cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  3 · 3 · 1
2 < cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 3 · 3 · 1
< cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  false · 3 · 1
cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  3 · 1
dup 1 − + swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  3 · 3 · 1
1 − + swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  1 · 3 · 3 · 1
− + swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  −1 · 3 · 3 · 1
+ swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 3 · 1
swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  3 · 1 · 2
∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  3 · 2
swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 3
true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  true · 2 · 3
loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 3
dup 2 < cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 2 · 3
2 < cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 2 · 2 · 3
< cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  false · 2 · 3
cond [false | dup 1 − + swap2 ∗ swap true] loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 3
dup 1 − + swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 2 · 3
1 − + swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  1 · 2 · 2 · 3
− + swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  −1 · 2 · 2 · 3
+ swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  1 · 2 · 3
swap2 ∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  2 · 3 · 1
∗ swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  6 · 1
swap true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗
  1 · 6
true loop [dup 2 < cond [false | dup 1 − + swap2 ∗ swap true]] pop dup 1 + ∗

true · 1 · 6
loop [**dup** 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**]] **pop dup** 1 + ∗
1 · 6
dup 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**] **loop** [**dup** 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**]] **pop dup** 1 + ∗
1 · 1 · 6
2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**] **loop** [**dup** 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**]] **pop dup** 1 + ∗
2 · 1 · 1 · 6
< **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**] **loop** [**dup** 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**]] **pop dup** 1 + ∗
true · 1 · 6
cond [**false** | **dup** 1 − + **swap2** ∗ **swap true**] **loop** [**dup** 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**]] **pop dup** 1 + ∗
1 · 6
false loop [**dup** 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**]] **pop dup** 1 + ∗
false · 1 · 6
loop [**dup** 2 < **cond** [**false** | **dup** 1 − + **swap2** ∗ **swap true**]] **pop dup** 1 + ∗
1 · 6
pop dup 1 + ∗
6
dup 1 + ∗
6 · 6
1 + ∗
1 · 6 · 6
+ ∗
7 · 6
∗
42
nop