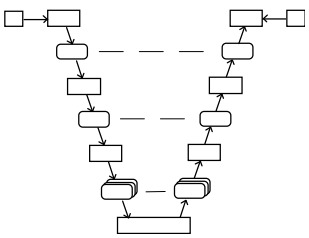
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Μεταγλωττιστές 2021

Θέμα εργασίας

Η γλώσσα Llama



Μεταγλωττιστές

`https://courses.softlab.ntua.gr/compilers/`

Διδάσκοντες: *Νίκος Παπασπύρου*
Κωστής Σαγώνας

Βοηθός: *Βίκυ Κοντουρά*

Αθήνα, Μάρτιος 2021

ΘΕΜΑ:

Να σχεδιαστεί και να υλοποιηθεί από κάθε ομάδα, το πολύ δυο σπουδαστών, ένας μεταγλωττιστής για τη γλώσσα Llama. Γλώσσα υλοποίησης μπορεί να είναι μια από τις C/C++, Java, SML, OCaml, Haskell ή Python, αλλά έχετε υπ' όψη σας ότι κομμάτια του μεταγλωττιστή και εργαλεία που μπορεί να σας φανούν χρήσιμα μπορεί να μην είναι διαθέσιμα σε κάποιες από τις παραπάνω γλώσσες και σε αυτήν την περίπτωση θα πρέπει να τα υλοποιήσετε μόνοι σας. Η επιλογή κάποιας άλλης γλώσσας υλοποίησης μπορεί να γίνει κατόπιν συνεννόησης με τους διδάσκοντες. Επιτρέπεται να χρησιμοποιηθούν και επίσης συνιστάται η χρήση εργαλείων, π.χ. `flex`/`ML-Lex`/`ocamllex`/`Alex`, `bison`/`ML-Yacc`/`ocamlyacc`/`Happy`, `JavaCC`, κ.λπ. Περισσότερες πληροφορίες σχετικές με αυτά τα εργαλεία θα δοθούν στις παραδόσεις.

Παραδοτέα, ημερομηνίες και βαθμολόγηση

Τα τμήματα του μεταγλωττιστή και η κατανομή μονάδων φαίνονται στον παρακάτω πίνακα. Οι ημερομηνίες παράδοσης αναγράφονται στη σελίδα του μαθήματος.

Τμήμα του μεταγλωττιστή	*Μονάδες*	*Bonus*
Λεκτικός αναλυτής	0.5	0.1
Συντακτικός αναλυτής	0.5	0.1
Σημασιολογικός αναλυτής	1.0	0.6
Ενδιάμεσος κώδικας	1.2	0.4
Βελτιστοποίηση	0.6	0.4
Τελικός κώδικας	1.2	0.4
Συνολική εργασία και έκθεση	5.0	2.0

Για τα διάφορα τμήματα της εργασίας πρέπει να παραδίδεται εμπρόθεσμα από κάθε ομάδα ο αντίστοιχος κώδικας σε ηλεκτρονική μορφή, καθώς και σαφείς οδηγίες για την παραγωγή ενός εκτελέσιμου προγράμματος επίδειξης της λειτουργίας του αντίστοιχου τμήματος, από τον κώδικα αυτόν. Καθυστερημένες ασκήσεις θα βαθμολογούνται με μικρότερο βαθμό, αντιστρόφως ανάλογα προς το χρόνο καθυστέρησης. Παρακαλούμε, **μην παραδίδετε τυπωμένες εργασίες**! Η μορφή και τα περιεχόμενα των παραδοτέων, συμπεριλαμβανομένης και της τελικής έκθεσης, πρέπει να συμφωνούν με τις οδηγίες που δίνονται στην ενότητα 4 του παρόντος.

Προαιρετικά τμήματα και μονάδες bonus

Το σύνολο των μονάδων της παρούσας εργασίας είναι 7. Από αυτές, οι 2 μονάδες είναι bonus (που σημαίνει ότι το σύνολο μονάδων του μαθήματος είναι 12) και αντιστοιχούν στα παρακάτω τμήματα της εργασίας, που είναι προαιρετικά:

- (10%) Πραγματικοί αριθμοί (`float`).
- (30%) Εξαγωγή τύπων (type inference).
- (20%) Τύποι που ορίζονται από τον προγραμματιστή.
- (20%) Βελτιστοποίηση ενδιάμεσου κώδικα.
  Οτιδήποτε απαιτεί ανάλυση ροής ελέγχου ή/και ανάλυση ροής δεδομένων.
- (20%) Δέσμευση καταχωρητών και βελτιστοποίηση τελικού κώδικα.

Περιεχόμενα

1 Περιγραφή της γλώσσας Llama

Η γλώσσα Llama είναι μια σχετικά απλή γλώσσα που συνδυάζει τα μοντέλα του συναρτησιακού και του προστακτικού προγραμματισμού. Βασίζεται σε ένα υποσύνολο της OCaml, με την οποία παρουσιάζει πολλές ομοιότητες. Τα κύρια χαρακτηριστικά της εν συντομία είναι τα εξής:

- Ισχυρό σύστημα τύπων με εξαγωγή τύπων (type inference).
- Βασικοί τύποι δεδομένων για ακέραιους αριθμούς, χαρακτήρες, λογικές τιμές και πραγματικούς αριθμούς.
- Τύποι αναφορών (δεικτών) και πινάκων μιας ή πολλών διαστάσεων. Οι θέσεις μνήμης όπου δείχνουν οι αναφορές καθώς και τα στοιχεία των πινάκων είναι μεταβλητές και επιδέχονται αναθέσεων.
- ~~Τύποι δεδομένων που ορίζονται από τον προγραμματιστή, πιθανώς και αναδρομικοί.~~
- Συναρτήσεις υψηλής τάξης, χωρίς όμως μερική εφαρμογή. Πέρασμα παραμέτρων κατ' αξία.
- Βιβλιοθήκη συναρτήσεων.

Περισσότερες λεπτομέρειες της γλώσσας δίνονται στις παραγράφους που ακολουθούν.

1.1 Λεκτικές μονάδες

Οι λεκτικές μονάδες της γλώσσας Llama χωρίζονται στις παρακάτω κατηγορίες:

- Τις *λέξεις κλειδιά*, οι οποίες είναι οι παρακάτω:

```
and       array     begin     bool      char      delete
dim       do        done      downto    else      end
false     ~~float~~     for       if        in        int
let       match     mod       mutable   new       not
of        rec       ref       then      to        true
type      unit      while     with
```

- Τα *ονόματα*, τα οποία αποτελούνται από ένα γράμμα του λατινικού αλφαβήτου, πιθανώς ακολουθούμενο από μια σειρά γραμμάτων, δεκαδικών ψηφίων ή χαρακτήρων υπογράμμισης (underscore). Τα ονόματα δεν πρέπει να συμπίπτουν με τις λέξεις κλειδιά που αναφέρθηκαν παραπάνω. Τα πεζά και τα κεφαλαία γράμματα διαφέρουν και μάλιστα υπάρχουν δύο κατηγορίες ονομάτων:
  - Τα ονόματα σταθερών, μεταβλητών, συναρτήσεων και τύπων, τα οποία αρχίζουν με πεζό γράμμα.
  - Τα ονόματα κατασκευαστών, τα οποία αρχίζουν με κεφαλαίο γράμμα.
- Οι *ακέραιες σταθερές* χωρίς πρόσημο, που αποτελούνται από ένα ή περισσότερα δεκαδικά ψηφία. Παραδείγματα ακέραιων σταθερών είναι τα ακόλουθα:

```
0     42     1284     00200
```

- ~~Οι *πραγματικές σταθερές* χωρίς πρόσημο, που αποτελούνται από ένα ακέραιο μέρος, ένα κλασματικό μέρος και ένα προαιρετικό εκθετικό μέρος. Το ακέραιο μέρος αποτελείται από ένα ή περισσότερα δεκαδικά ψηφία. Το κλασματικό μέρος αποτελείται από το χαρακτήρα `.` της υποδιαστολής, ακολουθούμενο από ένα ή περισσότερα δεκαδικά ψηφία. Τέλος, το εκθετικό μέρος αποτελείται από το πεζό ή κεφαλαίο γράμμα `E`, ένα προαιρετικό πρόσημο `+` ή `-` και ένα ή περισσότερα δεκαδικά ψηφία. Παραδείγματα πραγματικών σταθερών είναι τα ακόλουθα:~~

Πίνακας 1: Ακολουθίες διαφυγής (escape sequences).

Ακολουθία διαφυγής	Περιγραφή
\n	ο χαρακτήρας αλλαγής γραμμής (line feed)
\t	ο χαρακτήρας στηλοθέτησης (tab)
\r	ο χαρακτήρας επιστροφής στην αρχή της γραμμής (carriage return)
\0	ο χαρακτήρας με ASCII κωδικό 0
\\	ο χαρακτήρας \ (backslash)
\'	ο χαρακτήρας ' (απλό εισαγωγικό)
\"	ο χαρακτήρας " (διπλό εισαγωγικό)
\x$nn$	ο χαρακτήρας με ASCII κωδικό nn στο δεκαεξαδικό σύστημα

```
42.0      4.2e1      0.420e+2      42000.0e-3
```

- Οι *σταθεροί χαρακτήρες*, που αποτελούνται από ένα χαρακτήρα μέσα σε απλά εισαγωγικά. Ο χαρακτήρας αυτός μπορεί να είναι οποιοσδήποτε κοινός χαρακτήρας ή *ακολουθία διαφυγής* (escape sequence). Κοινοί χαρακτήρες είναι όλοι οι εκτυπώσιμοι χαρακτήρες πλην των απλών και διπλών εισαγωγικών και του χαρακτήρα \ (backslash). Οι ακολουθίες διαφυγής ξεκινούν με το χαρακτήρα \ (backslash) και περιγράφονται στον πίνακα 1. Παραδείγματα σταθερών χαρακτήρων είναι οι ακόλουθες:

```
'a'      '1'      '\n'      '\"      '\x1d'
```

- Οι *σταθερές συμβολοσειρές*, που αποτελούνται από μια ακολουθία κοινών χαρακτήρων ή ακολουθιών διαφυγής μέσα σε διπλά εισαγωγικά. Οι συμβολοσειρές δεν μπορούν να εκτείνονται σε περισσότερες από μια γραμμές προγράμματος. Παραδείγματα σταθερών συμβολοσειρών είναι οι ακόλουθες:

```
"abc"           "Route66"           "Helloworld!\n"
     "Name:\t\"DouglasAdams\"\nValue:\t42\n"
```

- Τους *συμβολικούς τελεστές*, οι οποίοι είναι οι παρακάτω:

```
->      =       |       +       -       *       /       +.
-.      *.      /.      **      !       ;       &&      ||
<>      <       >       <=      >=      ==      !=      :=
```

- Τους *διαχωριστές*, οι οποίοι είναι οι παρακάτω:

```
(       )       [       ]       ,       :
```

Εκτός από τις λεκτικές μονάδες που προαναφέρθηκαν, ένα πρόγραμμα Llama μπορεί επίσης να περιέχει τα παρακάτω, τα οποία διαχωρίζουν λεκτικές μονάδες και αγνοούνται:

- *Κενούς χαρακτήρες*, δηλαδή ακολουθίες αποτελούμενες από κενά διαστήματα (space), χαρακτήρες στηλοθέτησης (tab), χαρακτήρες αλλαγής γραμμής (line feed) ή χαρακτήρες επιστροφής στην αρχή της γραμμής (carriage return).

- *Σχόλια μιας γραμμής*, τα οποία αρχίζουν με την ακολουθία χαρακτήρων `--` και τερματίζονται με το τέλος της τρέχουσας γραμμής.

- *Σχόλια πολλών γραμμών*, τα οποία αρχίζουν με την ακολουθία χαρακτήρων `(*` και τερματίζονται με την ακολουθία χαρακτήρων `*)`. Τα σχόλια αυτής της μορφής επιτρέπεται να είναι φωλιασμένα.

1.2 Τύποι δεδομένων

Η Llama υποστηρίζει πέντε βασικούς τύπους δεδομένων:

- `unit`: τετριμμένος τύπος δεδομένων, αντίστοιχος του `void` της C,
- `int`: ακέραιοι αριθμοί,
- `char`: χαρακτήρες,
- `bool`: λογικές τιμές, και
- ~~`float`: πραγματικοί αριθμοί.~~

Εκτός από τους βασικούς τύπους, η Llama υποστηρίζει επίσης τύπους αναφορών, τύπους πινάκων, τύπους συναρτήσεων και τύπους ορισμένους από τον προγραμματιστή.

Οι τύποι αναφορών συμβολίζονται με t `ref`, όπου ο t πρέπει να είναι έγκυρος τύπος, όχι όμως τύπος πίνακα.

Οι τύποι πινάκων συμβολίζονται με `array [*, ... *] of` t, όπου ο t επίσης πρέπει να είναι έγκυρος τύπος, όχι όμως τύπος πίνακα. Το πλήθος των `*` που εμφανίζονται μέσα στις αγκύλες παριστάνει το πλήθος των διαστάσεων του πίνακα. Αν πρόκειται για μονοδιάστατο πίνακα, ο συμβολισμός μπορεί να απλοποιηθεί σε `array of` t.

Οι τύποι συναρτήσεων συμβολίζονται με t_1 `->` t_2, όπου t_1 και t_2 πρέπει να είναι έγκυροι τύποι και ο t_2 δεν μπορεί να είναι τύπος πίνακα. Ένας τύπος συνάρτησης μπορεί πάντα να γραφεί στη μορφή:

$$t_1 \texttt{ -> } (t_2 \texttt{ -> } (\ldots \texttt{ -> } (t_n \texttt{ -> } t)\ldots))$$

όπου ο τύπος αποτελέσματος t δεν είναι τύπος συνάρτησης. Αυτή μπορεί να συντομευθεί σε:[1]

$$t_1 \texttt{ -> } t_2 \texttt{ -> } \ldots \texttt{ -> } t_n \texttt{ -> } t$$

Μια συνάρτηση αυτού του τύπου λέμε ότι δέχεται n παραμέτρους, με τύπους $t_1, \ldots t_n$, και επιστρέφει αποτέλεσμα τύπου t.

Οι τύποι που ορίζονται από τον προγραμματιστή συμβολίζονται με ονόματα και δηλώνονται με χρήση λέξης κλειδιού `type`.

1.3 Δομή του προγράμματος

Ένα πρόγραμμα Llama αποτελείται από μια (πιθανώς κενή) ακολουθία ορισμών. Οι ορισμοί ανήκουν σε δύο κατηγορίες:

- Ορισμοί σταθερών, μεταβλητών και συναρτήσεων, που αρχίζουν με `let` ή με `let rec`.
- Ορισμοί τύπων, με τη λέξη κλειδί `type`.

Σε κάθε ορισμό μπορεί να ορίζεται συγχρόνως ένα ή περισσότερα ονόματα, με χρήση της λέξης κλειδιού `and`.

Η εμβέλεια κάθε ορισμού που γίνεται με απλό `let` εκτείνεται από το τέλος του ορισμού μέχρι το τέλος του προγράμματος. Η εμβέλεια των ορισμών που γίνονται με `let rec` ή με `type` εκτείνεται από την αρχή του ορισμού μέχρι το τέλος του προγράμματος. Κατ' αυτόν τον τρόπο επιτρέπονται αναδρομικοί ορισμοί. Η εμβέλεια ενός ορισμού δεν περιέχει τμήματα του προγράμματος όπου το ίδιο όνομα επανορίζεται (αυτό επιτρέπεται να γίνει ακόμα και σε επόμενο ορισμό στο ίδιο επίπεδο).

[1] Θεωρούμε ότι ο τελεστής συνάρτησης `->` είναι δεξιά προσεταιριστικός και έχει μικρότερη προτεραιότητα από το `ref` και το `array ... of`. Παρενθέσεις μπορούν να χρησιμοποιηθούν για την ομαδοποίηση τύπων έτσι ώστε να ξεπεραστούν οι κανόνες προτεραιότητας, π.χ. `(int->int)->int`.

1.3.1 Ορισμοί σταθερών

Μερικά παραδείγματα ορισμού σταθερών:

```
let x = 3
let y = x+1   -- y is 4
```

Σε κάθε δήλωση μπορεί να δηλώνεται ο τύπος της σταθεράς, π.χ.

```
let x : int = 3
let y : int = x+1   -- y is 4
```

Παράδειγμα σύγχρονου ορισμού με μεταγενέστερο επανορισμό των ίδιων σταθερών:

```
let a = 3
and b = 7
let a = b   -- a is now 7
and b = a   -- b is now 3
```

Η εκτέλεση του ορισμού μιας σταθεράς προκαλεί τον υπολογισμό της τιμής της, δηλαδή της έκφρασης που ακολουθεί το =.

1.3.2 Ορισμοί μεταβλητών

Οι μεταβλητές ορίζονται με χρήση της λέξης κλειδιού `mutable`. Παραδείγματα:

```
let mutable x : int
```

Το όνομα `x` έχει στη συνέχεια τύπο `int ref` και είναι μια αναφορά που δείχνει σε μία νέα θέση μνήμης, όπου μπορεί να αποθηκευθεί ένας ακέραιος. Τα αρχικά περιεχόμενα της μεταβλητής είναι απροσδιόριστα.

Ιδιαίτερο είδος μεταβλητών είναι οι πίνακες. Ο ορισμός τους γίνεται με αναγραφή των διαστάσεών τους, που μπορούν να είναι οποιεσδήποτε εκφράσεις, π.χ.

```
let mutable a [5, 3] : float
```

ορίζει έναν διδιάστατο πίνακα 5×3 πραγματικών αριθμών. Ο τύπος του ονόματος `a` είναι `array [*,*] of float`. Τα αρχικά περιεχόμενα του πίνακα είναι απροσδιόριστα.

Ο τύπος της μεταβλητής (ή του στοιχείου του πίνακα) σε έναν ορισμό μπορεί να παραλείπεται και σε αυτή την περίπτωση υπολογίζεται αυτόματα από το μεταγλωττιστή της Llama, βάσει του τρόπου με τον οποίο η μεταβλητή χρησιμοποιείται.

1.3.3 Ορισμοί συναρτήσεων

Στους ορισμούς συναρτήσεων, το όνομα της συνάρτησης ακολουθείται από ένα ή περισσότερα ονόματα τυπικών παραμέτρων, η εμβέλεια των οποίων είναι το σώμα της συνάρτησης. Παραδείγματα:

```
let inc x = x + 1
let odd x = x mod 2 <> 0
```

Σε έναν ορισμό συνάρτησης μπορούν να καθορίζονται οι τύποι των παραμέτρων ή/και του αποτελέσματος:

```
let inc (x : int) : int  = x + 1
let odd (x : int) : bool = x mod 2 <> 0
```

Επίσης επιτρέπονται ορισμοί αναδρομικών συναρτήσεων με χρήση του `let rec`:

```
let rec fact n = if n = 0 then 1 else n * fact (n-1)
```

ή και αμοιβαία αναδρομικών συναρτήσεων:

```
let rec even n = if n = 0 then true else odd (n-1)
    and odd  n = if n = 0 then false else even (n-1)
```

Επιτρέπεται επίσης ο ορισμός συναρτήσεων υψηλής τάξης, που δέχονται ως παραμέτρους άλλες συναρτήσεις:

```
let twice f x = f (f x)
let y = twice inc 5       -- y is 7
```

Απαγορεύεται όμως ο ορισμός συναρτήσεων που επιστρέφουν ως αποτέλεσμα συνάρτηση.

1.3.4 Ορισμοί τύπων

Οι ορισμοί τύπων γίνονται με χρήση της λέξης κλειδιού `type`. Για κάθε νέο τύπο ορίζεται ένα σύνολο κατασκευαστών (constructors) που κατασκευάζουν στοιχεία αυτού του τύπου. Για παράδειγμα:

```
type color = Red | Green | Blue
```

ορίζει έναν τύπο με όνομα `color`, τα μόνα στοιχεία του οποίου είναι οι κατασκευαστές `Red`, `Green` και `Blue` (πρβλ. τους απαριθμητούς τύπους της Pascal ή τα `enum` της C).

Οι κατασκευαστές μπορούν να δέχονται οσεσδήποτε παραμέτρους οποιωνδήποτε τύπων, για παράδειγμα:

```
type number = Integer of int | Real of float | Complex of float float
```

Στον ορισμό αυτό, οι κατασκευαστές `Integer` και `Real` δέχονται μία παράμετρο (τύπου `int` και `float` αντίστοιχα) ενώ ο κατασκευαστής `Complex` δέχεται δύο παραμέτρους (τύπου `float`).

Τέλος, οι τύποι που ορίζονται από τον προγραμματιστή μπορούν να είναι αναδρομικοί (αυτοαναφορικοί). Για παράδειγμα, ο τύπος:

```
type list = Nil | Cons of int list
```

ορίζει λίστες με πληροφορία ακέραιους αριθμούς. Μπορούν επίσης να είναι και αμοιβαία αναδρομικοί, όπως για παράδειγμα ο τύπος:

```
type tree   = Leaf | Node of int forest
 and forest = Empty | NonEmpty of tree forest
```

που ορίζει (γενικευμένα) δέντρα με πληροφορία ακέραιους αριθμούς.

1.4 Εκφράσεις

Οι εκφράσεις είναι το βασικό δομικό υλικό των προγραμμάτων της Llama. Κάθε έκφραση διαθέτει ένα μοναδικό τύπο και μπορεί να αποτιμηθεί δίνοντας ως αποτέλεσμα μια τιμή αυτού του τύπου.

1.4.1 Σταθερές εκφράσεις

Στις σταθερές εκφράσεις της γλώσσας Llama συγκαταλέγονται οι ακόλουθες:

- Οι ακέραιες σταθερές χωρίς πρόσημο, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `int` και η τιμή τους είναι ίση με το μη αρνητικό ακέραιο αριθμό που παριστάνουν.
- Οι λέξεις κλειδιά `true` και `false`, με τύπο `bool` και προφανείς τιμές.
- Οι πραγματικές σταθερές χωρίς πρόσημο, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `float` και η τιμή τους είναι ίση με τον μη αρνητικό πραγματικό αριθμό που παριστάνουν.
- Οι σταθεροί χαρακτήρες, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `char` και η τιμή τους είναι ίση με το χαρακτήρα που παριστάνουν.

- Οι σταθερές συμβολοσειρές, όπως περιγράφονται στην ενότητα 1.1. Έχουν τύπο `arrayofchar`. Κάθε τέτοια σταθερά είναι ένας πίνακας χαρακτήρων, όπου βρίσκονται αποθηκευμένοι με τη σειρά οι χαρακτήρες της συμβολοσειράς. Στο τέλος του πίνακα αποθηκεύεται αυτόματα ο χαρακτήρας `'\0'`, σύμφωνα με τη σύμβαση που ακολουθεί η γλώσσα C για τις συμβολοσειρές.

- Η σταθερά `()` είναι το μοναδικό στοιχείο του τύπου `unit`.

1.4.2 Ονόματα

Ονόματα που έχουν προηγουμένως οριστεί, όπως π.χ. σταθερών, συναρτήσεων, παραμέτρων, κατασκευαστών, είναι εκφράσεις. Ο τύπος τους και η τιμή τους έπεται του ορισμού.

1.4.3 Αριθμητικοί τελεστές

Οι αριθμητικοί τελεστές χωρίζονται σε δύο κατηγορίες: αυτούς που εφαρμόζονται σε ακέραια τελουμένα και αυτούς που εφαρμόζονται σε πραγματικά τελούμενα.

- Οι τελεστές με ένα τελούμενο + και − υλοποιούν τους τελεστές προσήμου για ακέραιους αριθμούς. Το τελούμενο πρέπει να είναι τύπου `int` και το αποτέλεσμα είναι του ίδιου τύπου.

- ~~Ομοίως, οι τελεστές με ένα τελούμενο `+.` και `-.` υλοποιούν τους τελεστές προσήμου για πραγματικούς αριθμούς. Το τελούμενο πρέπει να είναι τύπου `float` και το αποτέλεσμα είναι του ίδιου τύπου.~~

- Οι τελεστές με δύο τελούμενα `+`, `-`, `*`, `/` και `mod` υλοποιούν τις ακέραιες αριθμητικές πράξεις. Τα τελούμενα πρέπει να είναι τύπου `int` και το αποτέλεσμα είναι του ίδιου τύπου.

- ~~Οι τελεστές με δύο τελούμενα `+.`, `-.`, `*.` και `/.` υλοποιούν τις πραγματικές αριθμητικές πράξεις. Τα τελούμενα πρέπει να είναι τύπου `float` και το αποτέλεσμα είναι του ίδιου τύπου.~~

- ~~Ο τελεστής με δύο τελούμενα `**` υλοποιεί την ύψωση σε δύναμη. Τα τελούμενα πρέπει να είναι τύπου `float` και το αποτέλεσμα είναι του ίδιου τύπου.~~

1.4.4 Τελεστές συγκρίσεων

Το μόνο ιδιαίτερο στοιχείο των τελεστών σύγκρισης της Llama είναι διάκριση ανάμεσα σε δύο είδη ισότητας: τη δομική ισότητα και τη φυσική ισότητα. Η διαφορά τους γίνεται εμφανής μόνο για τελούμενα τύπων που ορίζονται από τον προγραμματιστή και θα εξηγηθεί παρακάτω. Για τελούμενα των υπόλοιπων τύπων δεδομένων, οι δύο έννοιες ισότητας ταυτίζονται.

- Οι τελεστές = και <> υλοποιούν αντίστοιχα τη δομική ισότητα και ανισότητα. Το αποτέλεσμα είναι τύπου `bool`. Τα τελούμενα πρέπει να είναι του ίδιου τύπου, ο οποίος δεν μπορεί να είναι τύπος πίνακα ή συνάρτησης.

- Οι τελεστές <, >, <= και >= υλοποιούν τις σχέσεις ανισότητας μεταξύ αριθμών ή χαρακτήρων. Το αποτέλεσμα είναι τύπου `bool`. Τα τελούμενα πρέπει να είναι του ίδιου τύπου, ο οποίος πρέπει να είναι ένας από τους `int`, `float` ή `char`.

- Οι τελεστές == και != υλοποιούν αντίστοιχα τη φυσική ισότητα και ανισότητα. Το αποτέλεσμα είναι τύπου `bool`. Τα τελούμενα πρέπει να είναι του ίδιου τύπου, ο οποίος δεν μπορεί να είναι τύπος πίνακα ή συνάρτησης.

Δυο στοιχεία ενός τύπου t ορισμένου από τον προγραμματιστή είναι φυσικά ίσα όταν πρόκειται ακριβώς για το ίδιο στοιχείο, δηλαδή έχουν κατασκευαστεί κατά την εκτέλεση του προγράμματος με την ίδια κλήση ενός κατασκευαστή c. Αντίθετα, δύο στοιχεία του t είναι δομικά ίσα αν έχουν κατασκευαστεί με (πιθανώς διαφορετικές) κλήσεις του ίδιου κατασκευαστή c εφαρμοσμένου πάνω σε τιμές που είναι

κατ' αντιστοιχία δομικά ίσες. Επομένως, δύο στοιχεία που είναι δομικά ίσα μπορεί να είναι φυσικά άνισα. Αν όμως δύο στοιχεία είναι φυσικά ίσα, τότε είναι οπωσδήποτε και δομικά ίσα. Για παράδειγμα, με τον τύπο `number` που ορίστηκε στην ενότητα 1.3.4:

```
let n1 = Integer 1
and n2 = Integer 2
and n3 = Integer 1

let test = (n1 = n1)    -- true
let test = (n1 == n1)   -- true
let test = (n1 = n2)    -- false
let test = (n1 == n2)   -- false
let test = (n1 = n3)    -- true
let test = (n1 == n3)   -- false
```

1.4.5 Λογικοί τελεστές

Εκτός από τους τελεστές σύγκρισης, οι παρακάτω τελεστές παράγουν λογικές εκφράσεις.

- Ο τελεστής `not` υλοποιεί τη λογική άρνηση. Το τελούμενό του πρέπει να είναι τύπου `bool`, και τον ίδιο τύπο έχει και το αποτέλεσμα.

- Οι τελεστές `&&` και `||` υλοποιούν αντίστοιχα τις πράξεις της λογικής σύζευξης και διάζευξης. Τα τελούμενα πρέπει να είναι τύπου `bool` και τον ίδιο τύπο έχει και το αποτέλεσμα. Η αποτίμηση εκφράσεων που χρησιμοποιούν αυτούς τους τελεστές γίνεται με *βραχυκύκλωση* (short-circuit). Δηλαδή, αν το αποτέλεσμα της έκφρασης είναι γνωστό από την αποτίμηση και μόνο του πρώτου τελούμενου, το δεύτερο τελούμενο δεν αποτιμάται καθόλου.

1.4.6 Αναφορές και ανάθεση

Οι παρακάτω τελεστές αφορούν σε τελούμενα τύπου αναφοράς.

- Ο τελεστής `:=` αναθέτει την τιμή του δεύτερου τελουμένου στη μεταβλητή όπου δείχνει το πρώτο τελούμενο. Το πρώτο τελούμενο πρέπει να είναι τύπου t `ref`, ενώ το δεύτερο τελούμενο πρέπει να είναι τύπου t. Το αποτέλεσμα είναι τύπου `unit`.

- Αν e είναι μια έκφραση τύπου t `ref`, τότε `!`e είναι μια έκφραση τύπου t, που αντιστοιχεί στην τιμή που περιέχει η θέση μνήμης όπου δείχνει η αναφορά e.

1.4.7 Στοιχεία και διαστάσεις πινάκων

Οι παρακάτω τελεστές αφορούν σε τελούμενα τύπου πίνακα.

- Αν a είναι ένα όνομα τύπου n-διάστατου πίνακα (`array ... of` t) και $e_1, \ldots e_n$ είναι εκφράσεις τύπου `int`, τότε η έκφραση a`[`$e_1, \ldots e_n$`]` είναι τύπου `ref` t. Η τιμή της είναι μια αναφορά που δείχνει στη θέση μνήμης όπου βρίσκεται αποθηκευμένο το στοιχείο του πίνακα με συντεταγμένες τις τιμές των εκφράσεων $e_1, \ldots e_n$. Η αρίθμηση των συντεταγμένων σε κάθε διάσταση ξεκινά από το 0, όπως στη γλώσσα προγραμματισμού C.

- Αν a είναι ένα όνομα τύπου n-διάστατου πίνακα (`array ... of` t) και i είναι μια ακέραια σταθερά χωρίς πρόσημο με τιμή μεταξύ 1 και n, τότε η έκφραση `dim` i a έχει τύπο `int` και η τιμή της είναι το μέγεθος της i-οστής διάστασης του πίνακα.

1.4.8 Κλήση συναρτήσεων και κατασκευαστών

Αν f είναι το όνομα μιας συνάρτησης (ή ενός κατασκευαστή) με n παραμέτρους τύπων $t_1, \ldots t_n$ και αποτέλεσμα τύπου t, και $e_1, \ldots e_n$ είναι εκφράσεις με τύπους $t_1, \ldots t_n$ αντίστοιχα, τότε η έκφραση $f e_1 \ldots e_n$ έχει τύπο t. Όταν υπολογίζεται, προκαλεί την κλήση της συνάρτησης f, κατά την οποία οι πραγματικές παράμετροι αποτιμώνται από αριστερά προς τα δεξιά.

Επισημαίνεται ότι η κλήση συναρτήσεων και κατασκευαστών δηλώνεται με την απλή παράθεση των πραγματικών παραμέτρων μετά το όνομα της συνάρτησης ή του κατασκευαστή και ότι, αντίθετα με τις περισσότερες γλώσσες συναρτησιακού προγραμματισμού, το αποτέλεσμα μιας κλήσης δεν μπορεί να είναι τύπου συνάρτησης (και αργότερα να εφαρμοστεί σε νέες πραγματικές παραμέτρους).

1.4.9 Αποτίμηση υπό συνθήκη

Αν e είναι μια έκφραση τύπου `bool`, e_1 και e_2 είναι εκφράσεις του ίδιου τύπου t, τότε η έκφραση `if` e `then` e_1 `else` e_2 είναι τύπου t. Η αποτίμησή της γίνεται ξεκινώντας από την αποτίμηση της e. Αν η τιμή αυτής είναι `true`, τότε αποτιμάται η e_1 και η τιμή αυτής είναι το αποτέλεσμα. Διαφορετικά, αποτιμάται η e_2 και η τιμή αυτής είναι το αποτέλεσμα. Σε κάθε περίπτωση, αποτιμάται μόνο ένα από τα τελούμενα e_1 και e_2.

Το σκέλος `else` e_2 μπορεί να παραλειφθεί, στην περίπτωση που η έκφραση e_1 είναι τύπου `unit`. Στην περίπτωση αυτή αν η συνθήκη είναι ψευδής δε γίνεται τίποτα.

1.4.10 Τοπικοί ορισμοί

Με τη χρήση της δομής `let ... in` e μπορούν να οριστούν σταθερές, συναρτήσεις και μεταβλητές η εμβέλεια των οποίων είναι η έκφραση e. Τέτοιοι ορισμοί ονομάζονται τοπικοί, σε αντιδιαστολή με τους γενικούς (καθολικούς) ορισμούς. Οι τοπικοί ορισμοί μπορούν να είναι και αναδρομικοί με χρήση της λέξης κλειδιού `rec`.

1.4.11 Δυναμική διαχείριση μνήμης

Οι τελεστές `new` και `delete` χρησιμοποιούνται για τη δυναμική διαχείριση μνήμης.

- Η έκφραση `new` t, όπου t έγκυρος τύπος αλλά όχι τύπος πίνακα, προκαλεί τη δυναμική παραχώρηση μνήμης. Το αποτέλεσμα είναι τύπου t `ref`. Η αναφορά αυτή τοποθετείται να δείχνει προς μία νέα δυναμική μεταβλητή τύπου t. Τα αρχικά περιεχόμενα της νέας μεταβλητής είναι απροσδιόριστα.

- Η έκφραση `delete` e, όπου e έκφραση τύπου t `ref`, προκαλεί την αποδέσμευση της μνήμης στην οποία δείχνει η τιμή της e. Η μνήμη αυτή πρέπει να έχει προηγουμένως παραχωρηθεί δυναμικά με χρήση του τελεστή `new`. Το αποτέλεσμα είναι τύπου `unit`.

1.4.12 Εντολές για προστακτικό προγραμματισμό

Οι εκφράσεις της Llama που περιγράφηκαν μέχρι τώρα εξυπηρετούν το μοντέλο του συναρτησιακού προγραμματισμού (αν και όχι καθαρού εφόσον υπάρχουν αναφορές και αναθέσεις). Οι ακόλουθες μορφές εκφράσεων αποκαθιστούν το οικειότερο μοντέλο του προστακτικού προγραμματισμού ως ισότιμο με το συναρτησιακό στη γλώσσα Llama.

- Ο τελεστής `;` υπολογίζει τα δύο τελούμενά του και επιστρέφει την τιμή του δεύτερου, αγνοώντας αυτήν του πρώτου. Τα δύο τελούμενα μπορούν να είναι οποιουδήποτε έγκυρου τύπου, όχι κατ' ανάγκην του ίδιου. Το αποτέλεσμα είναι του ίδιου τύπου με αυτόν του δεύτερου τελούμενου.

- Η έκφραση `begin` e `end` είναι ισοδύναμη με την e. Οι λέξεις κλειδιά `begin` και `end` ομαδοποιούν εκφράσεις ακριβώς όπως και οι παρενθέσεις.

- Αν e_1 και e_2 είναι εκφράσεις με τύπο `bool` και `unit` αντίστοιχα, η έκφραση `while` e_1 `do` e_2 `done` είναι τύπου `unit`. Κατά την αποτίμησή της, πρώτα αποτιμάται η συνθήκη e_1. Αν η τιμή της είναι `true` τότε υπολογίζεται η έκφραση e_2 και επαναλαμβάνεται η αποτίμηση όλης της έκφρασης από την αρχή (βρόχος). Διαφορετικά, αν η τιμή της e_1 είναι `false` τότε η αποτίμηση τερματίζεται.

- Αν e_1 και e_2 είναι εκφράσεις με τύπο `int`, i είναι ένα όνομα και e μια έκφραση με τύπο `unit`, τότε η έκφραση `for` i = e_1 `to` e_2 `do` e `done` είναι τύπου `unit`. Η σημασιολογία της είναι η ίδια με την εντολή `for` της Pascal. Ομοίως για την περίπτωση `downto`. Το όνομα i είναι τύπου `int` και η εμβέλειά του είναι η έκφραση e.

1.4.13 Αποσύνθεση τύπων που ορίζονται από τον προγραμματιστή

Έστω e είναι μια έκφραση κάποιου τύπου t που έχει οριστεί από τον προγραμματιστή με έναν ορισμό της μορφής:

`type` t `=` c_1 `of` t_1^1 `...` $t_{m_1}^1$ `|` c_2 `of` t_1^2 `...` $t_{m_2}^2$ `| ... |` c_n `of` t_1^n `...` $t_{m_n}^n$

για κάποιους θετικούς φυσικούς αριθμούς $n, m_1, \ldots m_n$. Έστω επίσης εκφράσεις $e_1, \ldots e_k$ κάποιου κοινού τύπου t' (k θετικός φυσικός αριθμός). Τότε, η έκφραση:

`match` e `with` pat_1 `->` e_1 `|` pat_2 `->` e_2 `| ... |` pat_k `->` e_k `end`

είναι έγκυρη έκφραση τύπου t' υπό την προϋπόθεση τα $pat_1, \ldots pat_k$ να είναι έγκυρα πρότυπα (patterns) για τον τύπο t. Κατά τον υπολογισμό αυτής της έκφρασης, υπολογίζεται πρώτα η έκφραση e. Στη συνέχεια η τιμή της συγκρίνεται κατά σειρά με τα πρότυπα $pat_1, \ldots pat_k$ ώσπου να βρεθεί ένα, έστω το pat_i, που να ταιριάζει. Τότε υπολογίζεται η έκφραση e_i και η τιμή της είναι το αποτέλεσμα της συνολικής έκφρασης. Αν κανένα από τα πρότυπα δεν ταιριάζει στην τιμή της e, τότε προκαλείται σφάλμα εκτέλεσης. Οι έννοιες του έγκυρου προτύπου και του ταιριάσματος ορίζονται παρακάτω.

- Αν n είναι μια έγκυρη ακέραια σταθερά χωρίς πρόσημο, τότε τα n, `+` n και `-` n είναι έγκυρα πρότυπα για τον τύπο `int`. Ταιριάζουν στις ακέραιες τιμές n, n και $-n$ αντίστοιχα.

- Αν f είναι μια έγκυρη πραγματική σταθερά χωρίς πρόσημο, τότε τα f, `+.` f και `-.` f είναι έγκυρα πρότυπα για τον τύπο `float`. Ταιριάζουν στις πραγματικές τιμές f, f και $-f$ αντίστοιχα.

- Αν c είναι μια έγκυρη σταθερά χαρακτήρα, τότε το c είναι έγκυρο πρότυπο για τον τύπο `char`. Ταιριάζει στην τιμή χαρακτήρα c.

- Τα `true` και `false` είναι έγκυρα πρότυπα για τον τύπο `bool`. Ταιριάζουν στις αντίστοιχες λογικές τιμές.

- Αν x είναι ένα όνομα που αρχίζει με πεζό γράμμα, τότε το x είναι έγκυρο πρότυπο για οποιονδήποτε τύπο t. Ταιριάζει σε κάθε τιμή αυτού του τύπου και το ταίριασμά του προκαλεί τον ορισμό μιας νέας σταθεράς με αυτό το όνομα και αυτή την τιμή (σαν να είχε χρησιμοποιηθεί `let`). Η εμβέλεια αυτής της σταθεράς είναι η έκφραση που βρίσκεται στο σκέλος του `match` το οποίο περιέχει αυτό το πρότυπο.

- Αν c_i είναι ένα όνομα κατασκευαστή για κάποιον τύπο t που ορίστηκε όπως παραπάνω, τότε το $c_i\ pat_1\ \ldots\ pat_{m_k}$ είναι έγκυρο πρότυπο για τον τύπο t υπό την προϋπόθεση τα πρότυπα $pat_1, \ldots,$ pat_{m_k} να είναι έγκυρα πρότυπα για τους τύπους $t_1^i, \ldots, t_{m_i}^i$. Ταιριάζει σε τιμές του τύπου t της μορφής $c_i\ v_1\ \ldots\ v_{m_k}$, υπό την προϋπόθεση τα πρότυπα $pat_1, \ldots, pat_{m_k}$ να ταιριάζουν στις τιμές $v_1, \ldots, v_{m_k}$.

- Αν pat είναι έγκυρο πρότυπο για κάποιον τύπο t τότε το (pat) είναι έγκυρο πρότυπο για τον ίδιο τύπο t. Ταιριάζει ακριβώς στις ίδιες τιμές που ταιριάζει και το pat. Οι παρενθέσεις στα πρότυπα χρησιμοποιούνται για ομαδοποίηση.

Πίνακας 2: Προτεραιότητα και προσεταιριστικότητα των τελεστών της Llama.

Τελεστές	Περιγραφή	Αριθμός τελουμένων	Θέση και προσεταιριστικότητα
`new`	Δυναμική δέσμευση μνήμης	1	καμία
`[ ]`	Στοιχείο πίνακα	≥ 2	καμία
`!`	Αποαναφορά	1	prefix
παράθεση	Κλήση συνάρτησης	≥ 2	καμία
`+ - +. -. not delete`	Πρόσημα, λογική άρνηση, δυναμική αποδέσμευση μνήμης	1	prefix
`**`	Ύψωση σε δύναμη	2	infix, δεξιά
`* / *. /. mod`	Πολλαπλασιαστικοί τελεστές	2	infix, αριστερή
`+ - +. -.`	Προσθετικοί τελεστές	2	infix, αριστερή
`= <> > < <= >= == !=`	Τελεστές σύγκρισης	2	infix, καμία
`&&`	Λογική σύζευξη	2	infix, αριστερή
`||`	Λογική διάζευξη	2	infix, αριστερή
`:=`	Ανάθεση	2	infix, καμία
`if then else`	Τελεστής συνθήκης	3	ειδική
`;`	Τελεστής παράθεσης	2	infix, αριστερή
`let in`	Τοπικοί ορισμοί	1	prefix

Για παράδειγμα, η παρακάτω έκφραση προκαλεί την εκτύπωση ενός στοιχείου `num` του τύπου `number` που ορίστηκε στην ενότητα 1.3.4, κατά τρόπο ανάλογο με το είδος αυτού του στοιχείου.

```
match num with
   Integer i      -> print_int i
 | Real f         -> print_float f
 | Complex re 0.0 -> print_float re
 | Complex 0.0 im -> print_string "j"; print_float im
 | Complex re im  -> print_float re;
                     print_string (if im > 0.0 then "+j" else "-j");
                     print_float (abs_float im)
end
```

1.4.14 Προτεραιότητα και προσεταιριστικότητα τελεστών

Στον πίνακα 2 ορίζεται η προτεραιότητα και η προσεταιριστικότητα των τελεστών της Llama. Οι γραμμές που βρίσκονται υψηλότερα στον πίνακα περιέχουν τελεστές μεγαλύτερης προτεραιότητας. Τελεστές που βρίσκονται στην ίδια γραμμή έχουν την ίδια προτεραιότητα.

1.5 Βιβλιοθήκη έτοιμων συναρτήσεων

Η Llama υποστηρίζει ένα σύνολο προκαθορισμένων συναρτήσεων, οι οποίες έχουν υλοποιηθεί σε assembly του 80x86 ως μια *βιβλιοθήκη έτοιμων συναρτήσεων* (run-time library). Είναι ορατές σε κάθε δομική μονάδα, εκτός αν επισκιάζονται από μεταβλητές, παραμέτρους ή άλλες συναρτήσεις με το ίδιο όνομα. Παρακάτω δίνονται οι τύποι τους και εξηγείται η λειτουργία τους.

1.5.1 Είσοδος και έξοδος

```
print_int    : int -> unit
print_bool   : bool -> unit
```

```
print_char   : char -> unit
print_float  : float -> unit
print_string : array of char -> unit
```

Οι συναρτήσεις αυτές χρησιμοποιούνται για την εκτύπωση τιμών που ανήκουν στους βασικούς τύπους της Llama, καθώς και για την εκτύπωση συμβολοσειρών.

```
read_int    : unit -> int
read_bool   : unit -> bool
read_char   : unit -> char
read_float  : unit -> float
read_string : array of char -> unit
```

Αντίστοιχα, οι παραπάνω συναρτήσεις χρησιμοποιούνται για την εισαγωγή τιμών που ανήκουν στους βασικούς τύπους της Llama και για την εισαγωγή συμβολοσειρών. Η συνάρτηση `read_string` χρησιμοποιείται για την ανάγνωση μιας συμβολοσειράς μέχρι τον επόμενο χαρακτήρα αλλαγής γραμμής. Η παράμετρος είναι ο πίνακας χαρακτήρων όπου θα τοποθετηθεί το αποτέλεσμα της ανάγνωσης. Ο χαρακτήρας αλλαγής γραμμής δεν αποθηκεύεται. Σε περίπτωση που το μέγεθος του πίνακα δεν επαρκεί για την αποθήκευση της συμβολοσειράς που διαβάζεται (συμπεριλαμβανομένου του τελικού `'\0'`) τότε αποθηκεύονται οι χαρακτήρες που χωρούν και η ανάγνωση θα συνεχιστεί αργότερα από το σημείο όπου διακόπηκε.

1.5.2 Μαθηματικές συναρτήσεις

```
abs  : int -> int
fabs : float -> float
```

Η απόλυτη τιμή ενός ακέραιου ή πραγματικού αριθμού.

```
sqrt : float -> float
sin  : float -> float
cos  : float -> float
tan  : float -> float
atan : float -> float
exp  : float -> float
ln   : float -> float
pi   : unit -> float
```

Βασικές μαθηματικές συναρτήσεις: τετραγωνική ρίζα, τριγωνομετρικές συναρτήσεις, εκθετική συνάρτηση, φυσικός λογάριθμος, ο αριθμός π.

1.5.3 Συναρτήσεις αύξησης και μείωσης

```
incr : int ref -> unit
decr : int ref -> unit
```

Αυξάνουν κατά ένα και μειώνουν κατά ένα αντίστοιχα το περιεχόμενο της ακέραιας μεταβλητής όπου δείχνει η παράμετρος.

1.5.4 Συναρτήσεις μετατροπής

```
float_of_int : int -> float
int_of_float : float -> int
round        : float -> int
```

Η `int_of_float` επιστρέφει τον πλησιέστερο ακέραιο αριθμό, η απόλυτη τιμή του οποίου είναι μικρότερη από την απόλυτη τιμή της παραμέτρου. Η `round` επιστρέφει τον πλησιέστερο ακέραιο αριθμό. Σε περίπτωση αμφιβολίας, προτιμάται ο αριθμός με τη μεγαλύτερη απόλυτη τιμή.

```
int_of_char : char -> int
char_of_int : int -> char
```

Μετατρέπουν από ένα χαρακτήρα στον αντίστοιχο κωδικό ASCII και αντίστροφα.

1.5.5 Συναρτήσεις διαχείρισης συμβολοσειρών

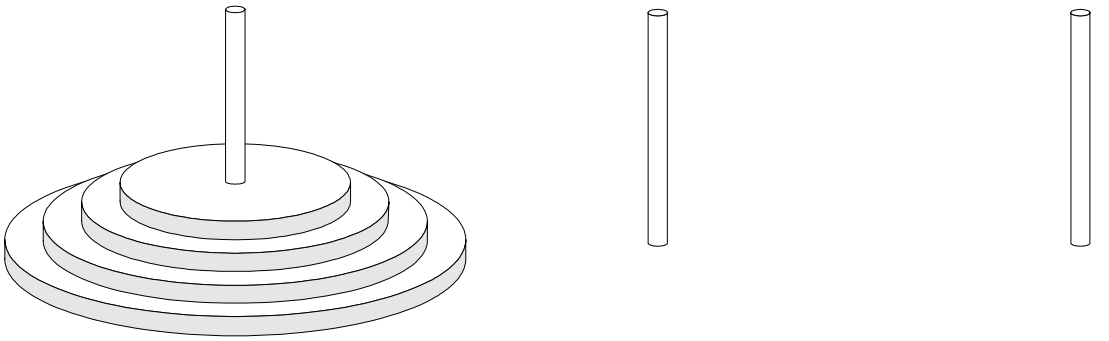
```
strlen : array of char -> int
strcmp : array of char -> array of char -> int
strcpy : array of char -> array of char -> unit
strcat : array of char -> array of char -> unit
```

Οι συναρτήσεις αυτές έχουν ακριβώς την ίδια λειτουργία με τις συνώνυμές τους στη βιβλιοθήκη συναρτήσεων της γλώσσας C.

2 Πλήρης γραμματική της Llama

Η σύνταξη της γλώσσας Llama δίνεται παρακάτω σε μορφή EBNF. Η γραμματική που ακολουθεί είναι *διφορούμενη*, οι αμφισημίες όμως μπορούν να ξεπερασθούν αν λάβει κανείς υπόψη τους κανόνες προτεραιότητας και προσεταιριστικότητας των τελεστών, όπως περιγράφονται στον πίνακα 2. Τα σύμβολα ⟨id⟩, ⟨Id⟩, ⟨int-const⟩, ⟨float-const⟩, ⟨char-const⟩ και ⟨string-literal⟩ είναι τερματικά σύμβολα της γραμματικής.

⟨program⟩ ::= (⟨letdef⟩ | ⟨typedef⟩)*
⟨letdef⟩ ::= "**let**" ["**rec**"] ⟨def⟩ ("**and**" ⟨def⟩)*
⟨def⟩ ::= ⟨id⟩ (⟨par⟩)* ["**:**" ⟨type⟩] "**=**" ⟨expr⟩
| "**mutable**" ⟨id⟩ ["**[**" ⟨expr⟩ ("**,**" ⟨expr⟩)* "**]**"] ["**:**" ⟨type⟩]
⟨typedef⟩ ::= "**type**" ⟨tdef⟩ ("**and**" ⟨tdef⟩)*
⟨tdef⟩ ::= ⟨id⟩ "**=**" ⟨constr⟩ ("**|**" ⟨constr⟩)*
⟨constr⟩ ::= ⟨Id⟩ ["**of**" (⟨type⟩)+]
⟨par⟩ ::= ⟨id⟩ | "**(**" ⟨id⟩ "**:**" ⟨type⟩ "**)**"
⟨type⟩ ::= "**unit**" | "**int**" | "**char**" | "**bool**" | "**float**" | "**(**" ⟨type⟩ "**)**" | ⟨type⟩ "**->**" ⟨type⟩
| ⟨type⟩ "**ref**" | "**array**" ["**[**" "*****" ("**,**" "*****")* "**]**"] "**of**" ⟨type⟩ | ⟨id⟩
⟨expr⟩ ::= ⟨int-const⟩ | ⟨float-const⟩ | ⟨char-const⟩ | ⟨string-literal⟩ | "**true**" | "**false**"
| "**(**" "**)**" | "**(**" ⟨expr⟩ "**)**" | ⟨unop⟩ ⟨expr⟩ | ⟨expr⟩ ⟨binop⟩ ⟨expr⟩
| (⟨id⟩ | ⟨Id⟩) (⟨expr⟩)* | ⟨id⟩ "**[**" ⟨expr⟩ ("**,**" ⟨expr⟩)* "**]**" | "**dim**" [⟨int-const⟩] ⟨id⟩
| "**new**" ⟨type⟩ | "**delete**" ⟨expr⟩ | ⟨letdef⟩ "**in**" ⟨expr⟩ | "**begin**" ⟨expr⟩ "**end**"
| "**if**" ⟨expr⟩ "**then**" ⟨expr⟩ ["**else**" ⟨expr⟩] | "**while**" ⟨expr⟩ "**do**" ⟨expr⟩ "**done**"
| "**for**" ⟨id⟩ "**=**" ⟨expr⟩ ("**to**" | "**downto**") ⟨expr⟩ "**do**" ⟨expr⟩ "**done**"
| "**match**" ⟨expr⟩ "**with**" ⟨clause⟩ ("**|**" ⟨clause⟩)* "**end**"
⟨unop⟩ ::= "**+**" | "**-**" | "**+.**" | "**-.**" | "**!**" | "**not**"
⟨binop⟩ ::= "**+**" | "**-**" | "*****" | "**/**" | "**+.**" | "**-.**" | "***.**" | "**/.**" | "**mod**" | "******" | "**=**" | "**<>**"
| "**<**" | "**>**" | "**<=**" | "**>=**" | "**==**" | "**!=**" | "**&&**" | "**||**" | "**;**" | "**:=**"
⟨clause⟩ ::= ⟨pattern⟩ "**->**" ⟨expr⟩
⟨pattern⟩ ::= ["**+**" | "**-**"] ⟨int-const⟩ | ["**+.**" | "**-.**"] ⟨float-const⟩ | ⟨char-const⟩
| "**true**" | "**false**" | ⟨id⟩ | "**(**" ⟨pattern⟩ "**)**" | ⟨Id⟩ (⟨pattern⟩)*



Σχήμα 1: Οι πύργοι του Hanoi.

3 Παραδείγματα

Στην παράγραφο αυτή δίνονται εννιά παραδείγματα προγραμμάτων στη γλώσσα Llama, η πολυπλοκότητα των οποίων κυμαίνεται σημαντικά. Για κάποια από αυτά τα παραδείγματα (ή για παρόμοια προγράμματα), μπορείτε να βρείτε τον αρχικό, τον ενδιάμεσο κώδικα (χωρίς βελτιστοποίηση), τη μορφή των εγγραφημάτων δραστηριοποίησης των δομικών μονάδων, καθώς και τον τελικό κώδικα σε αντίστοιχα φυλλάδια περιγραφής γλωσσών που δόθηκαν ως θέματα εργασίας στο ίδιο μάθημα σε προηγούμενα έτη, μέσω της ιστοσελίδας του μαθήματος, ή στο Moodle.

3.1 Πες γεια!

Το παρακάτω παράδειγμα είναι το απλούστερο πρόγραμμα στη γλώσσα Llama που παράγει κάποιο αποτέλεσμα ορατό στο χρήστη. Το πρόγραμμα αυτό τυπώνει απλώς ένα μήνυμα.

```
1    let main = print_string "Hello world!\n"
```

3.2 Οι πύργοι του Hanoi

Το πρόγραμμα που ακολουθεί λύνει το πρόβλημα των πύργων του Hanoi. Μια σύντομη περιγραφή του προβλήματος δίνεται παρακάτω.

Υπάρχουν τρεις στύλοι, στον πρώτο από τους οποίους είναι περασμένοι n το πλήθος δακτύλιοι. Οι εξωτερικές διάμετροι των δακτυλίων είναι διαφορετικές και αυτοί είναι περασμένοι από κάτω προς τα πάνω σε φθίνουσα σειρά εξωτερικής διαμέτρου, όπως φαίνεται στο σχήμα 1. Ζητείται να μεταφερθούν οι δακτύλιοι από τον πρώτο στον τρίτο στύλο (χρησιμοποιώντας το δεύτερο ως βοηθητικό χώρο), ακολουθώντας όμως τους εξής κανόνες:

- Κάθε φορά επιτρέπεται να μεταφερθεί ένας μόνο δακτύλιος, από κάποιο στύλο σε κάποιον άλλο στύλο.
- Απαγορεύεται να τοποθετηθεί δακτύλιος με μεγαλύτερη διάμετρο πάνω από δακτύλιο με μικρότερη διάμετρο.

Το πρόγραμμα στη γλώσσα Llama που λύνει αυτό το πρόβλημα δίνεται παρακάτω. Η συνάρτηση `hanoi` είναι αναδρομική.

```
1    let main =
2       let move source target =
3          print_string "Moving from: ";
4          print_string source;
5          print_string " to ";
```

```
6           print_string target;
7           print_string "\n" in
8        let rec hanoi rings source target auxil =
9           if rings > 0 then
10          begin
11             hanoi (rings-1) source auxil target;
12             move source target;
13             hanoi (rings-1) auxil target source
14          end in
15       print_string "Please, give the number of rings: ";
16       let n = read_int () in
17       hanoi n "left" "right" "middle"
```

3.3 Οι πύργοι του Hanoi, ξανά

Το παρακάτω πρόγραμμα είναι παραλλαγή του προηγούμενου, που χρησιμοποιεί ένα νέο τύπο για την αναπαράσταση των στύλων.

```
1    type pile = Left | Middle | Right
2
3    let print_pile pile =
4       match pile with
5         Left   -> print_string "left"
6       | Middle -> print_string "middle"
7       | Right  -> print_string "right"
8       end
9
10   let main =
11      let move source target =
12         print_string "Moving from: ";
13         print_pile source;
14         print_string " to ";
15         print_pile target;
16         print_string "\n" in
17      let rec hanoi rings source target auxil =
18         if rings > 0 then
19         begin
20            hanoi (rings-1) source auxil target;
21            move source target;
22            hanoi (rings-1) auxil target source
23         end in
24      print_string "Please, give the number of rings: ";
25      let n = read_int () in
26      hanoi n Left Right Middle
```

3.4 Πρώτοι αριθμοί

Το παρακάτω παράδειγμα προγράμματος στη γλώσσα Llama είναι ένα πρόγραμμα που υπολογίζει τους πρώτους αριθμούς μεταξύ 1 και n, όπου το n καθορίζεται από το χρήστη. Το πρόγραμμα αυτό χρησιμοποιεί έναν απλό αλγόριθμο για τον υπολογισμό των πρώτων αριθμών. Μια διατύπωση αυτού του αλγορίθμου σε ψευδογλώσσα δίνεται παρακάτω. Λαμβάνεται υπόψη ότι οι αριθμοί 2 και 3 είναι πρώτοι, και στη συνέχεια εξετάζονται μόνο οι αριθμοί της μορφής $6k \pm 1$, όπου k φυσικός αριθμός.

Κύριο πρόγραμμα

τύπωσε τους αριθμούς 2 και 3
για $t := 6$ μέχρι n με βήμα 6 κάνε τα εξής:

αν ο αριθμός $t - 1$ είναι πρώτος τότε τύπωσέ τον
αν ο αριθμός $t + 1$ είναι πρώτος τότε τύπωσέ τον

Αλγόριθμος ελέγχου (είναι ο αριθμός t πρώτος;)

αν $t < 0$ τότε έλεγξε τον αριθμό $-t$
αν $t < 2$ τότε ο t δεν είναι πρώτος
αν $t = 2$ τότε ο t είναι πρώτος
αν ο t διαιρείται με το 2 τότε ο t δεν είναι πρώτος
για $i := 3$ μέχρι $t/2$ με βήμα 2 κάνε τα εξής:
    αν ο t διαιρείται με τον i τότε ο t δεν είναι πρώτος
ο t είναι πρώτος

Το αντίστοιχο πρόγραμμα στη γλώσσα Llama είναι το ακόλουθο. Χρησιμοποιεί αναδρομή αντί βρόχων επανάληψης.

```
1     let rec prime n =
2             if n < 0       then prime (-n)
3        else if n < 2       then false
4        else if n = 2       then true
5        else if n mod 2 = 0 then false
6        else let rec loop i =
7                if i <= n / 2 then
8                   if n mod i = 0 then false
9                                  else loop (i+2)
10               else
11                  true in
12            loop 3
13
14    let main =
15       print_string "Please, give the upper limit: ";
16       let limit = read_int () in
17       print_string "Prime numbers between 0 and ";
18       print_int limit;
19       print_string "\n\n";
20       let mutable counter in
21       counter := 0;
22       if limit >= 2 then (incr counter; print_string "2\n");
23       if limit >= 3 then (incr counter; print_string "3\n");
24       let rec loop number =
25          if number <= limit then
26          begin
27             if prime (number - 1) then
28             begin
29                incr counter;
30                print_int (number - 1);
31                print_string "\n"
32             end;
33             if number <> limit && prime (number + 1) then
34             begin
35                incr counter;
36                print_int (number + 1);
37                print_string "\n"
38             end;
39             loop (number + 6)
40          end in
41       loop 6;
42       print_string "\n";
```

```
43        print_int !counter;
44        print_string " prime number(s) were found.\n"
```

3.5 Αντιστροφή συμβολοσειράς

Το πρόγραμμα που ακολουθεί στη γλώσσα Llama εκτυπώνει το μήνυμα "Hello world!" αντιστρέφοντας τη δοθείσα συμβολοσειρά. Είναι γραμμένο στο προστακτικό μοντέλο προγραμματισμού.

```
1     let main =
2        let reverse s r =
3           let l = strlen s in
4           for i = 0 to l-1 do
5              r[i] := !s[l-i-1]
6           done;
7           r[l] := '\0' in
8
9        let mutable p [20] in
10
11       reverse "\n!dlrow olleH" p;
12       print_string p
```

3.6 Ταξινόμηση με τη μέθοδο της φυσαλίδας

Ο αλγόριθμος της φυσαλίδας (bubble sort) είναι ένας από τους πιο γνωστούς και απλούς αλγορίθμους ταξινόμησης. Το παρακάτω πρόγραμμα σε Llama τον χρησιμοποιεί για να ταξινομήσει έναν πίνακα ακεραίων αριθμών κατ' αύξουσα σειρά. Αν x είναι ο πίνακας που πρέπει να ταξινομηθεί και n είναι το μέγεθός του (θεωρούμε σύμφωνα με τη σύμβαση της Llama ότι τα στοιχεία του είναι τα $x[0], x[1], \ldots x[n-1]$), μια παραλλαγή του αλγορίθμου περιγράφεται με ψευδοκώδικα ως εξής:

Αλγόριθμος της φυσαλίδας (bubble sort)

επανάλαβε το εξής:
    για i από 0 ως $n-2$
        αν $x[i] > x[i+1]$
            αντίστρεψε τα $x[i]$ και $x[i+1]$
όσο μεταβάλλεται η σειρά των στοιχείων του x

Το αντίστοιχο πρόγραμμα σε γλώσσα Llama, επίσης γραμμένο στο προστακτικό μοντέλο προγραμματισμού, είναι το εξής:

```
3     let bsort x =
4        let swap x y =
5           let t = !x in x := !y; y := t in
6        let mutable changed in
7        changed := true;
8        while !changed do
9           changed := false;
10          for i = 0 to dim x - 2 do
11             if !x[i] > !x[i+1] then
12             begin
13                swap x[i] x[i+1];
14                changed := true
15             end
16          done
17       done
18
19    let main =
```

```
20        let print_array msg x =
21           print_string msg;
22           for i = 0 to dim x - 1 do
23              if i > 0 then print_string ", ";
24              print_int !x[i]
25           done;
26           print_string "\n" in
27
28        let mutable seed
29        and mutable x[16] in
30
31        seed := 65;
32        for i = 0 to 15 do
33           seed := (!seed * 137 + 220 + i) mod 101;
34           x[i] := !seed
35        done;
36        print_array "Initial array: " x;
37        bsort x;
38        print_array "Sorted array: " x
```

3.7 Μέση τιμή τυχαίας μεταβλητής

Το πρόγραμμα που ακολουθεί υπολογίζει τη μέση τιμή μιας ακέραιας τυχαίας μεταβλητής, που μεταβάλλεται ομοιόμορφα στο διάστημα από 0 έως $n-1$. Η τιμή του n καθορίζεται από το χρήστη, όπως και το πλήθος k των δειγμάτων που χρησιμοποιούνται για τον υπολογισμό.

```
1     let main =
2        print_string "Give n: ";
3        let n = read_int () in
4        print_string "Give k: ";
5        let k = read_int () in
6
7        let mutable sum
8        and mutable seed in
9
10       sum := 0.0;
11       seed := 65;
12
13       for i = 1 to k do
14          seed := (!seed * 137 + 220 + i) mod n;
15          sum := !sum +. float_of_int !seed
16       done;
17
18       if k > 0 then
19       begin
20          print_string "Mean: ";
21          print_float (!sum /. float_of_int k);
22          print_string "\n"
23       end
```

Από την εκτέλεση του προγράμματος διαπιστώνει κανείς ότι η μέση τιμή που προκύπτει διαφέρει σημαντικά από τη θεωρητική μέση τιμή $(n-1)/2$, που θα έπρεπε να προκύπτει όταν το k γίνει αρκετά μεγάλο. Αυτό οφείλεται στον απλοϊκό αλγόριθμο που έχει χρησιμοποιηθεί για τη γέννηση ψευδοτυχαίων αριθμών, λόγω του οποίου η μεταβλητή απέχει αρκετά από το να είναι τυχαία.

3.8 Πολλαπλασιασμός πινάκων

Το πρόγραμμα που ακολουθεί υπολογίζει το γινόμενο δύο πινάκων, διαστάσεων 3×4 και 4×5, τα στοιχεία των οποίων γεμίζουν με ψευδοτυχαίο τρόπο.

```
let mmult a b c =
   if dim 2 a = dim 1 b && dim 1 c = dim 1 a && dim 2 c = dim 2 b then
   begin
      for i = 0 to dim 1 c - 1 do
         for j = 0 to dim 2 c - 1 do
            c[i, j] := 0;
            for k = 0 to dim 2 a - 1 do
               c[i, j] := !c[i, j] + !a[i, k] * !b[k, j]
            done
         done
      done
   end

let mutable seed

let init = seed := 65

let minit m =
   for i = 0 to dim 1 m - 1 do
      for j = 0 to dim 2 m - 1 do
         seed := (!seed * 137 + 2*i + j) mod 101;
         m[i, j] := !seed
      done
   done

let mprint m =
   for i = 0 to dim 1 m - 1 do
      for j = 0 to dim 2 m - 1 do
         print_int !m[i, j];
         print_string " "
      done;
      print_string "\n"
   done

let main =
   let mutable x[3,4]
   and mutable y[4,5]
   and mutable z[3,5] in

   minit x;
   minit y;

   mprint x;
   print_string "\ntimes\n\n";
   mprint y;
   print_string "\nmakes\n\n";
   mmult x y z;
   mprint z
```

3.9 Δυαδικά δέντρα

Το παράδειγμα που ακολουθεί χρησιμοποιεί τους τύπους της Llama που ορίζονται από τον προγραμματιστή. Ορίζει έναν τύπο δεδομένων που αναπαριστά δυαδικά δέντρα με πληροφορία ακέραιους αριθμούς.

Υποθέτοντας ότι πρόκειται για δέντρα δυαδικής αναζήτησης και ότι δεν μπορούν να περιέχουν την ίδια πληροφορία πολλές φορές, στη συνέχεια ορίζει συναρτήσεις για:

- την εισαγωγή στοιχείων σε ένα δέντρο,
- τη συγχώνευση δύο δέντρων,
- τη διαγραφή στοιχείων από ένα δέντρο,
- την εκτύπωση ενός δέντρου (σε δομημένη pre-order μορφή), και
- το μέτρημα των κόμβων ενός δέντρου.

Έπειτα, με τη βοήθεια μιας γεννήτριας ψευδοτυχαίων αριθμών, το πρόγραμμα κατασκευάζει ένα δέντρο με 10 στοιχεία και το εκτυπώνει. Στη συνέχεια, επιλέγει με τυχαίο τρόπο ένα στοιχείο του δέντρου, το αφαιρεί και τυπώνει ό,τι απέμεινε. Επαναλαμβάνει ωσότου το δέντρο αδειάσει.

```
1   type tree = Nil | Node of int tree tree
2
3   let rec treeInsert t n =
4      match t with
5         Nil          -> Node n Nil Nil
6       | Node m t1 t2 ->      if n < m then Node m (treeInsert t1 n) t2
7                           else if n > m then Node m t1 (treeInsert t2 n)
8                           else t
9      end
10
11  let rec treeMerge t1 t2 =
12     match t1 with
13        Nil            -> t2
14      | Node n t11 t12 -> Node n t11 (treeMerge t12 t2)
15     end
16
17  let rec treeDelete t n =
18     match t with
19        Nil          -> t
20      | Node m t1 t2 -> if n < m then
21                           Node m (treeDelete t1 n) t2
22                        else if n > m then
23                           Node m t1 (treeDelete t2 n)
24                        else
25                           treeMerge t1 t2
26     end
27
28  let rec treePrint t =
29     match t with
30        Nil          -> ()
31      | Node n t1 t2 -> print_int n;
32                        print_string "(";
33                        treePrint t1;
34                        print_string "|";
35                        treePrint t2;
36                        print_string ")"
37     end
38
39  let rec treeCount t =
40     match t with
41        Nil          -> 0
42      | Node n t1 t2 -> 1 + treeCount t1 + treeCount t2
```

```
43          end
44
45      let main =
46          let mutable seed in
47          let next u =
48              seed := (!seed * 4241 + 22) mod 9949;
49              !seed in
50          seed := 65;
51
52          let random max = next () mod max in
53
54          let mutable t in
55          t := Nil;
56
57          for i = 1 to 10 do
58              t := treeInsert !t (random 100)
59          done;
60
61          print_string "Initial tree: ";
62          treePrint !t;
63          print_string "\n";
64
65          let rec choose t =
66              match t with
67                  Node n t1 t2 ->
68                      let c1 = treeCount t1
69                      and c2 = treeCount t2 in
70                      let r = random (1 + c1 + c2) in
71                      if r = 0 then
72                          n
73                      else if r <= c1 then
74                          choose t1
75                      else
76                          choose t2
77              end in
78
79          for i = 1 to treeCount !t do
80              let n = choose !t in
81              print_string "Deleting ";
82              print_int n;
83              print_string ": ";
84              t := treeDelete !t n;
85              treePrint !t;
86              print_string "\n"
87          done
```

4 Οδηγίες για την παράδοση

Ο τελικός μεταγλωττιστής πρέπει να μπορεί να εξάγει κατά βούληση ενδιάμεσο και τελικό κώδικα. Εφόσον δεν έχουν καθορισθεί οι παράμετροι λειτουργίας `-f` ή `-i`, που εξηγούνται παρακάτω, ο μεταγλωττιστής θα δέχεται το πηγαίο πρόγραμμα από ένα αρχείο με οποιαδήποτε κατάληξη (πχ. `*.lla`) που θα δίνεται ως το μοναδικό του όρισμα. Ο ενδιάμεσος κώδικας θα τοποθετείται σε αρχείο με κατάληξη `*.imm` και ο τελικός σε αρχείο με κατάληξη `*.asm`. Τα αρχεία αυτά θα βρίσκονται στον ίδιο κατάλογο και θα έχουν το ίδιο κυρίως όνομα. Π.χ. από το πηγαίο αρχείο `/tmp/hello.lla` θα παράγονται τα `/tmp/hello.imm` και `/tmp/hello.asm`.

Το εκτελέσιμο του τελικού μεταγλωττιστή θα πρέπει να δέχεται τις παρακάτω παραμέτρους:

`-O` σημαία βελτιστοποίησης (προαιρετική).

`-f` πρόγραμμα στο standard input, έξοδος τελικού κώδικα στο standard output.

`-i` πρόγραμμα στο standard input, έξοδος ενδιάμεσου κώδικα στο standard output.

Επίσης, η τιμή που θα επιστρέφεται στο λειτουργικό σύστημα από το μεταγλωττιστή θα πρέπει να είναι μηδενική στην περίπτωση επιτυχούς μεταγλώττισης και μη μηδενική σε αντίθετη περίπτωση.

Για την εύκολη ανάπτυξη του μεταγλωττιστή προτείνεται η χρήση ενός αρχείου `Makefile` με τα εξής (τουλάχιστον) χαρακτηριστικά:

- Με απλό `make`, θα δημιουργεί το εκτελέσιμο του τελικού μεταγλωττιστή.
- Με `makeclean`, θα σβήνει όλα ανεξαιρέτως τα αρχεία που παράγονται αυτόματα (π.χ. αυτά που παράγουν `bison` και `flex`, τα object files και το τελικό εκτελέσιμο).
- Με `make distclean`, θα σβήνει όλα τα παραπάνω και το τελικό εκτελέσιμο.

Ένα παράδειγμα ενός τέτοιου `Makefile`, υποθέτοντας ότι γλώσσα υλοποίησης είναι η C++ και γίνεται χρήση των εργαλείων `flex` και `bison`, είναι το παρακάτω.

```
.PHONY: default clean distclean
CXX=c++
CXXFLAGS=-Wall -std=c++11

default: compiler

lexer.cpp: lexer.l
	flex -s -o lexer.cpp lexer.l

parser.hpp parser.cpp: parser.y
	bison -dv -o parser.cpp parser.y

# Add your dependencies here, e.g.
# parser.o: parser.cpp lexer.hpp symbol.hpp

compiler: lexer.o parser.o symbol.o
	$(CXX) $(CXXFLAGS) -o $@ $^

clean:
	$(RM) lexer.cpp parser.cpp parser.hpp parser.output *.o

distclean: clean
	$(RM) compiler
```