Κεφάλαιο 1

Εργαστηριακή Άσκηση 1

Όπου ο αναγνώστης θα εξοικειωθεί με το προγραμματιστικό περιβάλλον, θα γίνει παρουσίαση των βασικών τύπων δεδομένων και των primitive διαδικασιών της Lisp και θα επιχειρηθεί μία εισαγωγή στις λίστες.

1.1 Εισαγωγικά

Στην $1^η$ Εργαστηριακή Άσκηση θα έρθετε σε μια πρώτη επαφή με το περιβάλλον *Lispworks Personal Edition* 5.0.1 , το οποίο θα χρησιμοποιήσετε στα πλαίσια του παρόντος εργαστηρίου για την ανάπτυξη των Common Lisp προγραμμάτων σας. Η πρώτη αυτή επαφή θα γίνει καθώς θα ξεδιπλώνεται η συζήτηση σχετικά με τους βασικούς τύπους δεδομένων της Lisp και τις εντολές διαχείρισης λιστών.

- Εκτελέστε:

  Start\All Programs\LispWorks 5.0 Personal\LispWorks

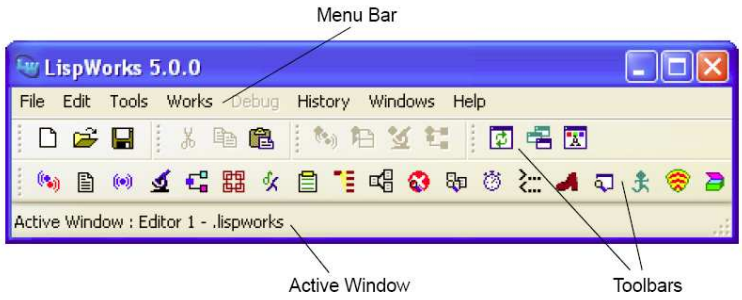
Θα τρέξει το *LispWorks Integrated Development Environment* με ανοικτό το default εργαλείο, τον *Listener*. Εάν δεν μπορείτε να δείτε τον Listener ή εάν τον κλείσετε, μπορείτε να τον ανοίξετε με δύο τρόπους:

1. Πηγαίνοντας: Tools > Listener, στο menu bar.
2. Πατώντας το πρώτο κουμπί από αριστερά (κάτω αριστερά) στο toolbar (Βλέπε Σχήμα 1.1).

Ο Listener είναι ένα εργαλείο που κάνει δυνατή την interactive αποτίμηση των Common Lisp εκφράσεων. Τα αποτελέσματα εκτυπώνονται απ΄ ευθείας στον Listener. Ο Listener είναι ένα πολύ χρήσιμο εργαλείο για την εύκολη εκτέλεση κώδικα μικρού μεγέθους.

- Πληκτρολογήστε για παράδειγμα: `CL-USER 1 > (+ 3.14 2.71)`
  και πατήστε return.

Η Lisp θα απαντήσει με το αποτέλεσμα της πρόσθεσης και στον Listener θα εμφανιστεί νέο prompt.



Σχήμα 1.1: Το podium του Lispworks (Εικόνα απ' το *Lispworks Personal Edition 5.0.0*).

Υπάρχουν αρκετά πράγματα που μπορούμε να διδαχθούμε απ' το παράδειγμα που μόλις είδαμε. Το πρώτο που πρέπει να επισημάνουμε είναι ότι ο Listener εκτυπώνει ένα prompt και περιμένει για είσοδο απ' τον χρήστη. Η είσοδος αυτή είναι μία *s-expression*[1] που μόλις ο χρήστης την πληκτρολογήσει και πατήσει το carriage return η Lisp θα πραγματοποιήσει τα ακόλουθα:

1. Θα *διαβάσει* τη δοθείσα s-expression.
2. Θα διερμηνεύσει τη δοθείσα s-expression ως την εκτυπωμένη αναπαράσταση ενός *τύπου* (*form*) - ένα αντικείμενο της Lisp που πρόκειται να αποτιμηθεί.
3. Θα *αποτιμήσει*/*εκτιμήσει* (*evaluate*) τον τύπο ως κάποιο άλλο (ή ίσως και ως το ίδιο) *value object*.
4. Θα επιλέξει μία εκτυπώσιμη αναπαράσταση για το value object.
5. Θα *εκτυπώσει* την αναπαράσταση που επέλεξε.

Λόγω αυτής της ακολουθίας ενεργειών που πραγματοποιούν, οι Lisp listeners καλούνται επίσης και *read-eval-print* loops. Επομένως, μπορούμε να πούμε ότι στο διάλογό μας με τη Lisp επαναλαμβάνονται συνεχώς τα εξής: Εμείς πληκτρολογούμε την εκτυπώσιμη αναπαράσταση ενός τύπου· η Lisp τον αποτιμάει και τυπώνει ως απάντηση την εκτυπώσιμη αναπαράσταση της τιμής του τύπου.

Το επόμενο που αξίζει να παρατηρήσουμε, είναι ότι όταν κάτι περικλείεται από παρενθέσεις στην Common Lisp καλούμε το αποτέλεσμα *λίστα* (*list*) και αυτά που περικλείονται απ' τις παρενθέσεις δεν είναι άλλα απ' τα *στοιχεία* (*elements*) της λίστας. Επομένως, η `(+ 3.14 2.71)` είναι μία λίστα με τρία στοιχεία, τα `+`, `3.14` και `2.71`. Βλέπουμε τέλος ότι τα στοιχεία μιας λίστας χωρίζονται με κενά.

Γιατί όμως γράφουμε `(+ 3.14 2.71)` και όχι `3.14 + 2.71` όπως στη συνήθη αριθμητική σημειογραφία; Στη Lisp το όνομα της *διαδικασίας* (*procedure*) που επιθυμούμε να εκτελεστεί πρέπει να τοποθετείται πάντα πρώτο για να μπορεί να το εντοπίσει ο διερμηνευτής. Τα υπόλοιπα στοιχεία της λίστας αποτελούν τότε τα *ορίσματα* (*arguments*) της διαδικασίας που αντιστοιχεί στο πρώτο στοιχείο της λίστας. Επομένως, στο παράδειγμά μας η Lisp διαβάζει: "Εκτέλεση της διαδικασίας

[1] Είναι συμβολικές εκφράσεις, όπου το 's' είναι απ' το *symbolic* και χρησιμοποιείται για να τις διαχωρίσει από τις *m-expressions*, όπου το 'm' είναι από το *meta*. Οι m-expressions χρησιμοποιούνταν μόνο απ' τον John McCarthy για την αναπαράσταση των s-expressions.

`+` με ορίσματα `3.14` και `2.71`''. Ο τρόπος αυτός γραφής καλείται *prefix notation*[2] και προάγει την ομοιομορφία, αφού η διαδικασία θα βρίσκεται πάντοτε στην πρώτη θέση, ανεξαρτήτως του πλήθους των ορισμάτων.

- Ποιό περιμένετε να είναι το αποτέλεσμα των ακόλουθων πράξεων:

  1. `(- 3 2)`
  2. `(- 1 99)`
  3. `(* 2 5)`
  4. `(* 5 -6.7)`
  5. `(/ 20 5)`
  6. `(/ 3 2)`
  7. `(/ 36 24)`
  8. `(/ 3 2.0)`
  9. `(+ 3 2 5 7)`

- Να εκτελέσετε τις παραπάνω πράξεις στον Listener του Lispworks για να ελέγξετε εάν σκεφτήκατε σωστά.

Οι πράξεις που μόλις εκτελέσατε ανέδειξαν κάποια ακόμη πράγματα:

1. Όταν μία διαίρεση μεταξύ δύο ακεραίων δεν είναι τέλεια, η Lisp παρουσιάζει το αποτέλεσμα με τη μορφή κλασματος. Μάλιστα, το κλάσμα αυτό είναι όσο το δυνατόν πιο απλοποιημένο (π.χ. 7).

2. Η ύπαρξη ενός δεκαδικού αριθμού σε οποιαδήποτε πράξη αναγκάζει το αποτέλεσμα να επιστραφεί σε δεκαδική μορφή (π.χ. 8).

3. Οι διαδικασίες `+, -, /` και `*` μπορούν να συνοδεύονται από οποιοδήποτε πλήθος ορισμάτων (π.χ. 9). Αν τα ορίσματα είναι πάνω από δύο, να θυμάστε ότι οι συγκεκριμένοι τελεστές εφαρμόζονται από αριστερά προς τα δεξιά (αυτό, ωστόσο, δεν συμβαίνει με όλες τις διαδικασίες). Να το επαληθεύσετε εκτελώντας τις πράξεις `(- 3 2 1)` και `(- 1 2 3)`.

Ας δούμε τώρα κάποιους λίγο πιο ακριβείς ορισμούς των όσων είδαμε μέχρι τώρα.

- Μια *διαδικασία* (*procedure*) είναι ένας βήμα προς βήμα καθορισμός, εκφρασμένος σε κάποια προγραμματιστική γλώσσα, του πως να γίνει κάτι.

- Μια διαδικασία, όπως η `+`, που παρέχεται απ' την ίδια τη Lisp , καλείται *primitive*.

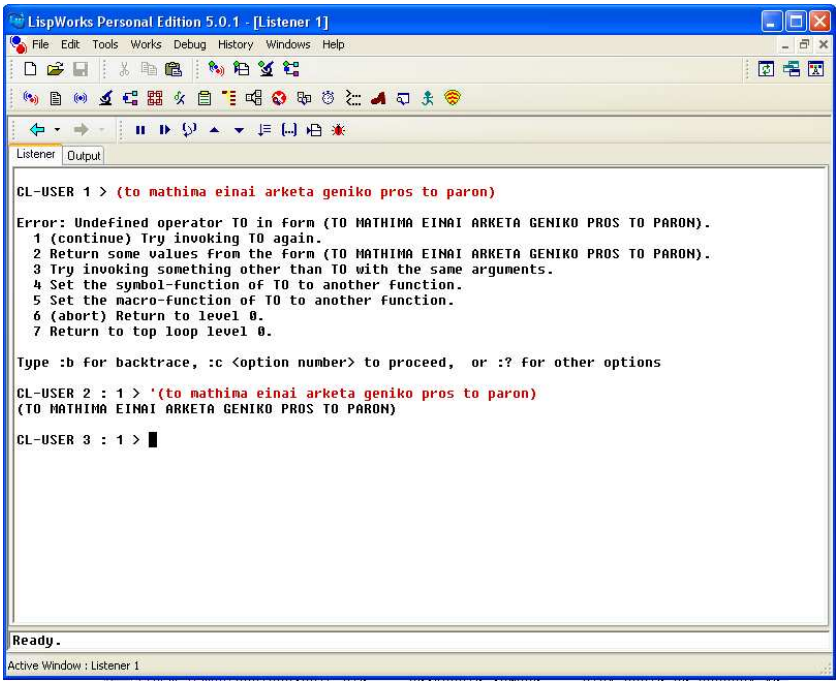
- Μία διαδικασία που ορίζεται απ' τον χρήστη καλείται *user-defined procedure*.

- *Πρόγραμμα* καλείται μία συλλογή από διαδικασίες που δουλεύουν μαζί.

Αξίζει, τέλος, να αναφέρουμε ότι η Lisp *δεν είναι case sensitive*, δηλαδή δε διαχωρίζει πεζούς από κεφαλαίους χαρακτήρες.

- Πληκτρολογήστε:

```
> (to mathima einai arketa geniko pros to paron).
```



Σχήμα 1.2: Παράδειγμα αλληλεπίδρασης με τον Listener του Lispworks (Εικόνα απ' το *Lispworks Personal Edition 5.0.1*).

Η Lisp θα σας απαντήσει με ένα μήνυμα σφάλματος. Πού πιστεύεται ότι οφείλεται αυτό; Αν η απάντηση δεν είναι προφανής, σκεφτείτε τί είπαμε ότι έκανε η Lisp όταν δώσαμε τη λίστα `(+ 3.14 2.71)` και προσπαθήστε να εντοπίσετε ποιά διαφορά της με τη λίστα που δώσαμε τώρα προκαλεί το σφάλμα.

- Πατήστε το κουμπί Abort του Debugger (βρίσκεται ακριβώς πάνω απ' το παράθυρο του Listener, στο τελευταίο toolbar από πάνω προς τα κάτω) ή πληκτρολογήστε `:a`. Εναλλακτικά, μπορείτε να επιλέξετε μία από τις επιλογές που εμφανίζονται μετά το μήνυμα του σφάλματος με το format `:c <option number>`. Για παράδειγμα, κάποιες επιλογές σας επιτρέπουν να επιλέξετε διαφορετική τιμή προς αποτίμηση απ' αυτή που προκάλεσε το σφάλμα, ή ακόμη και να αναθέσετε στο σύμβολο μία τιμή[3]. Τέλος, μπορείτε μετά το σφάλμα να πληκτρολογήσετε `:?` ή `:help` ώστε να εμφανιστούν όλες οι δυνατές επιλογές που σας δίνει ο debugger. Να δοκιμάσετε όλες τις προαναφερθείσες επιλογές, κάνοντας αλλεπάλληλα συντακτικά λάθη, έτσι ώστε να εξοικειωθείτε και να μπορείτε κάθε φορά να χρησιμοποιείτε όποια σας βολεύει περισσότερο.

- Πιέστε Alt+P ή πατήστε το κουμπί με το βέλος που δείχνει προς τα αριστερά (βρίσκεται ακριβώς πάνω απ' το παράθυρο του Listener, δίπλα στο toolbar του debugger). Με τον τρόπο αυτό μπορούμε να επαναφέρουμε πα-

[2] Γνωστός και ως *Cambridge Prefix Notation* από το M.I.T. Cambridge, Massachusetts στο οποίο σχεδιάστηκε η Lisp από τον John McCarthy και το Artificial Intelligence Group.

[3] Θα δούμε στη συνέχεια ότι τα σύμβολα χρησιμεύουν και ως μεταβλητές.

λιές εκφράσεις που πληκτρολογήσαμε. Εδώ, θέλουμε να επαναφέρουμε την s-expression: `(to mathima einai arketa geniko pros to paron)`.

- Προσθέστε πριν την πρώτη παρένθεση της λίστας το σύμβολο ' (*single-quote character*) και πατήστε carriage return.

Στο Σχήμα 1.2 παρουσιάζεται όλη η προαναφερθείσα αλληλεπίδραση με τον Listener της Lisp. Γιατί πιστεύετε ότι έπαψε το λάθος; Τί διαφορετικό ζητήσαμε τώρα απ' τη Lisp; Παρατηρήστε ότι η s-expression που δώσαμε τη δεύτερη φορά αποτιμάει στον εαυτό της και μάλιστα η Lisp τυπώνει με κεφαλαία ό,τι είχαμε δώσει εμείς με πεζά. Τί τύπος δεδομένων είναι η s-expression: `(to mathima einai arketa geniko pros to paron)` και πόσα στοιχεία έχει;

1.2 Τύποι δεδομένων στη Lisp

1.2.1 Γενικά

Ας δούμε συνοπτικά ποιοί είναι οι βασικοί τύποι δεδομένων της Lisp (Βλέπε και Σχήμα 1.3):

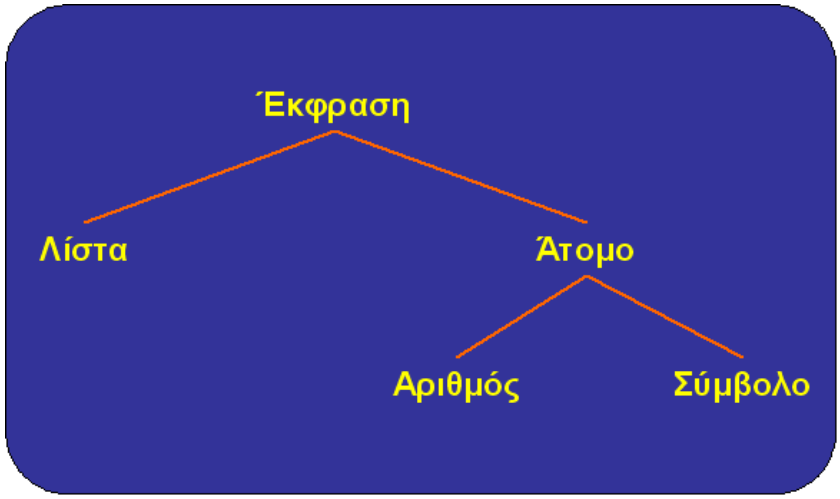
- Αδιαίρετες οντότητες όπως τα `27`, `3.14` και `+`, τα οποία έχουν προφανή σημασία, καθώς επίσης και πράγματα όπως τα `FOO`, `CEID`, `EYFYHS-PROGRAMMATISMOS` καλούνται *άτομα* (*atoms*).
- Άτομα όπως τα `27` και `3.14` καλούνται *αριθμητικά άτομα* (*numeric atoms*), ή πιο λακωνικά, *αριθμοί*.
- Άτομα όπως τα `FOO`, `CEID`, `EYFYHS-PROGRAMMATISMOS`, `FIRST` και `+` καλούνται *συμβολικά άτομα* (*symbolic atoms*), ή πιο λακωνικά, *σύμβολα*.
- Μία λίστα αποτελείται από μία αριστερή παρένθεση, που ακολουθείται από μηδέν ή περισσότερα άτομα ή λίστες, που ακολουθούνται από μία δεξιά παρένθεση.
- Τόσο τα άτομα όσο και οι λίστες καλούνται *συμβολικές εκφράσεις* (*s-expressions*), ή πιο λακωνικά, *εκφράσεις*.

Σημειώστε ότι όταν ενδιαφερόμαστε για την τιμή μιας έκφρασης συνηθίζουμε να αναφερόμαστε στην έκφραση με τον όρο ''*τύπος*'' (*form*). Εάν ο τύπος είναι μία λίστα, τότε το πρώτο στοιχείο της είναι γενικά το όνομα του procedure που θα χρησιμοποιηθεί για να παράγει την τιμή της έκφρασης. Αυτή ακριβώς η διαδικασία, του υπολογισμού της τιμής ενός τύπου, καλείται *αποτίμηση*.

1.2.2 Αριθμοί

Δώστε με τη σειρά τα ακόλουθα στο Listener (να πατάτε το carriage return μετά από κάθε αριθμό):

1. `124`, `-124`, `+124`.
2. `.3`, `0.05`, `0.005`, `0.0005`.
3. `3e-1`, `5e-2`, `.5e-2`, `50e-5`.



Σχήμα 1.3: Οι βασικοί τύποι δεδομένων της Lisp.

4. `3e1`, `3e2`, `3e3`, `5e6`, `5e7`, `9e1000`

5. `1/3`, `2/6`, `14/42`, `4/2`, `9/3`, `20/5`.

Λογικά, μετά και την εκτέλεση των παραπάνω παραδειγμάτων, θα έχετε ήδη παρατηρήσει ότι στη Lisp υπάρχουν 3 είδη αριθμών:

1. Οι *ακέραιοι* (*integers*), π.χ. `124`, `-124`, `+124`.

2. Οι *αριθμοί κινητής υποδιαστολής* (*floating point numbers*), π.χ. `.3`, `0.05`, `3e-1`.

3. Οι *λόγοι/κλάσματα* (*ratios*), π.χ. `1/3`, `2/6`.

Σε ό,τι αφορά στους αριθμούς κινητής υποδιαστολής, αυτό που πρέπει να προσέξουμε είναι ότι μπορούμε να τους γράψουμε σε *scientific notation*, όπως είναι για παράδειγμα ο αριθμός `3e-1`, που μεταφράζεται ως "*τρία επί δέκα εις τη μείον ένα*" (3×10^{-1}) και που η Lisp απλώς αναγνωρίζει ως `0.3`. Φυσικά, όπου το μέγεθος της αναπαράστασης το επιβάλλει, η Lisp θα επιστρέφει *scientific notation* ακόμα και αν εμείς είχαμε δώσει τον αριθμό κινητής υποδιαστολής στη συνήθη σημειογραφία, όπως π.χ. γίνεται για τον αριθμό `0.0005`. Επίσης, αντί για το e, μπορούμε να χρησιμοποιήσουμε και τα s, f, d, l για short-float, single-float, double-float και long-float, αντίστοιχα. Όλα αυτά αποτελούν υποκατηγορίες των αριθμών κινητής υποδιαστολής και διαφέρουν μεταξύ τους ως προς το πλήθος των bits που χρησιμοποιούνται για την αναπαράσταση. Εμείς θα χρησιμοποιούμε από εδώ και στο εξής μόνο το e εκτός και αν αναφέρεται διαφορετικά.

- Πληκτρολογήστε τους αριθμούς `1.0e-05`, `1.0s-05`, `1.0f-05`, `1.0d-05`, `1.0l-05`. Μπορείτε να δείτε γιατί στο Lispworks αρκεί να χρησιμοποιούμε το e; [4]

[4] Το ίδιο ισχύει και για τις περισσότερες υλοποιήσεις της Lisp.

Σε ό,τι αφορά στους λόγους, αξίζει να αναφέρουμε ότι η Lisp διαιρεί πάντα αριθμητή και παρονομαστή ενός λόγου με το *Μέγιστο Κοινό Διαιρέτη* τους, απλοποιώντας έτσι τους λόγους κάθε φορά που αυτό είναι εφικτό, π.χ. μετατρέπει το `14/42` σε `1/3`. Επίσης, κάθε λόγο που αντιστοιχεί σε τέλεια διαίρεση τον μετατρέπει στο αντίστοιχο πηλίκο, π.χ. το `4/2` μετατρέπεται σε `2`.

Ασκήσεις πάνω στους αριθμούς:

1. Δοκιμάστε τους αριθμούς $-.6$, `.43e5` και `0.0000521347`. Παρατηρήστε τί απαντάει κάθε φορά η Lisp.

2. Δοκιμάστε τους αριθμούς `3.` και `3.0`. Ερμηνεύονται ως ακέραιοι ή ως αριθμοί κινητής υποδιαστολής από την Lisp;

3. Μπορεί η Lisp να ξεχωρίσει τον αριθμό `0.0` από τον αριθμό `0`; Πειραματιστείτε επίσης με τους αριθμούς -0, `.0` και -0.0.

4. Πληκτρολογήστε τους αριθμούς `54`, `325`, `782` και `0.345879` αφήνοντας αρκετά κενά μεταξύ τους και μόνον αφού τους πληκτρολογήσετε όλους, πατήστε carriage return. Τί παρατηρείτε;

5. Πληκτρολογήστε τους χαρακτήρες `123;45` σε μία γραμμή. Το ';' είναι ο *χαρακτήρας σχολιασμού* (*comment character*). Η Lisp αγνοεί αυτόν καθώς και ό,τι ακολουθεί μετά απ' αυτόν μέχρι το τέλος της γραμμής. Δοκιμάστε, εν συνεχεία, να πληκτρολογήσετε το `;45` και αφού πατήσετε carriage return δοκιμάστε να πληκτρολογήσετε το `123` στην επόμενη γραμμή.

6. Πειραματιστείτε με τα primitives `ABS` και `SQRT`, που υπολογίζουν την απόλυτη τιμή και τη ρίζα, αντίστοιχα, ενός αριθμού. Να τις καλέστε τόσο με πεζά όσο και με κεφαλαία γράμματα για να δείτε εάν υπάρχει κάποια διαφορά. Χρησιμοποιούνται όπως τα `+, -, /, *` μόνο που δε δέχονται πάνω από ένα όρισμα.

1.2.3 Σύμβολα

Τα σύμβολα είναι ένας ακόμη τύπος δεδομένων της Lisp. Πρακτικά, μπορούν να περιέχουν οποιοδήποτε συνδυασμό γραμμάτων του λατινικού αλφαβήτου και αριθμών, καθώς επίσης και κάποιων ειδικών χαρακτήρων όπως οι παύλες ('-'). Δίπλα σε κάθε ένα απ' τα ακόλουθα, σημειώστε ένα 'N' αν είναι αριθμός, ή ένα 'S' αν είναι σύμβολο:

```
AARDVARK            ___
87                  ___
PLUMBING            ___
1-2-3-GO            ___
1492                ___
3.141592265358979   ___
22/7                ___
zerop               ___
ZERO                ___
0                   ___
-12                 ___
dekaepta            ___
```

Μπορείτε αφού συμπληρώσετε τα παραπάνω να ελέγξετε την ορθότητα των απαντήσεών σας με χρήση των κατηγορημάτων `numberp` και `symbolp`, που ελέγχουν αν το όρισμά τους είναι αριθμός ή σύμβολο, αντίστοιχα, και απαντούν με `T` για αλήθεια/ναι ή `NIL` για ψεύδος/όχι. Σε επόμενο εργαστήριο θα δούμε αναλυτικά τί είναι και πώς χρησιμοποιούνται τα κατηγορήματα. Εσείς μπορείτε προς το παρόν να τα χρησιμοποιείτε όπως τα primitives που έχουμε δει έως τώρα.

Ένα σύμβολο μπορεί να αναπαριστά κάτι για το οποίο θέλουμε να αποθηκεύσουμε πληροφορία. Για παράδειγμα, το σύμβολο `kostas` μπορεί να αναπαριστά κάποιο φυσικό πρόσωπο. Επιπρόσθετα, τα σύμβολα μπορούν να χρησιμοποιηθούν ως μεταβλητές. Έτσι, ένα σύμβολο που χρησιμοποιείται ως μεταβλητή μπορεί να έχει κάποια τιμή, περίπτωση κατά την οποία θα λέμε ότι είναι *δεσμευμένο* (*bound*) σε αυτή την τιμή, ή μπορεί να μην έχει κάποια τιμή, οπότε και θα λέμε ότι είναι *μη-δεσμευμένο* (*unbound*). Το πώς δεσμεύουμε τα σύμβολα σε τιμές θα το δούμε σε μετέπειτα εργαστήριο· προς το παρόν θα δούμε κάποια σύμβολα τα οποία έχουν ήδη κάποια τιμή από τη Lisp. Μερικά από αυτά είναι τα `pi`, `*read-base*`, `*print-base*` και `*package*`. Πληκτρολογήστε το `pi`, στον Listener και πατήστε carriage return· θα καταλάβετε αμέσως τί αναπαριστά. Πληκτρολογήστε το τώρα ως `'pi`. Η διαφορά οφείλεται στο ότι στην πρώτη περίπτωση δώσαμε το σύμβολο ως τύπο που η Lisp κατάλαβε ότι πρέπει να αποτιμήσει, ενώ αντίθετα στη δεύτερη δηλώσαμε ρητά ότι εισάγουμε δεδομένα τα οποία δεν επιθυμούμε να αποτιμηθούν (λόγω του `'`).

- Να δοκιμάσετε στο Listener ορισμένα παραδείγματα στα οποία θα δηλώνετε ρητά ότι ενδιαφέρεστε για το ίδιο το σύμβολο και όχι για την τιμή που πιθανώς να έχει.

- Εισάγετε ένα τυχαίο σύμβολο (ίσως ένα που να σημαίνει κάτι στο φυσικό κόσμο) και πατήστε carriage return. Καταλαβαίνετε γιατί αποτυγχάνει η Lisp; Εάν όχι, μια ματιά στο μήνυμα του σφάλματος είναι η απάντηση που ζητάτε.

Τώρα σε ό,τι αφορά στα `*read-base*` και `*print-base*`, αυτά λένε στη Lisp ποιά βάση θα χρησιμοποιούμε όταν εισάγουμε αριθμούς και ποιά βάση θέλουμε να χρησιμοποιεί η Lisp όταν τους τυπώνει, αντίστοιχα. Ζητήστε απ' τη Lisp να σας πει την τιμή τους. Το `*package*` θα συζητηθεί σε επόμενο εργαστήριο.

Τα πιο σημαντικά σύμβολα στη Lisp τα έχουμε ήδη αναφέρει και δεν είναι άλλα απ' τα `T` και `NIL`. Λόγω της τεράστιας σημασίας τους, είναι ήδη δεσμευμένα στον εαυτό τους, δηλαδή η τιμή τους είναι τα ίδια τα σύμβολα.

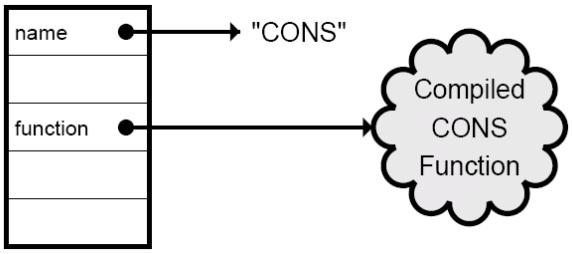
- Να επαληθεύσετε στην πράξη ότι τα `T`, `NIL` είναι δεσμευμένα στον εαυτό τους.

- Το primitive `eql` παίρνει δύο ορίσματα και ελέγχει αν αυτά είναι ίσα (δηλαδή, ταυτόσημα). Χρησιμοποιείστε το για να επαληθεύσετε ότι η Lisp δεν είναι case sensitive. Ελέγξτε και τί επιστρέφει όταν τα ορίσματά του δεν είναι ίσα.

- Κάθε σύμβολο έχει και ένα όνομα που δεν είναι άλλο από το *αλφαριθμητικό* (*string*) που χρησιμοποιείται για την εκτυπώσιμη μορφή του συμβόλου. Χρησιμοποιήστε το primitive `symbol-name` για να μάθετε τα ονόματα ορισμένων συμβόλων της επιλογής σας (καλείται με ένα μόνο όρισμα).

Το ίδιο το σύμβολο είναι κάτι το τελείως διαφορετικό απ' το όνομά του. Αξίζει να αναφέρουμε ότι ένα σύμβολο είναι ένα block που αποτελείται από πέντε δείκτες.

Ένας απ' αυτούς τους δείκτες, δείχνει στη θέση μνήμης που διατηρεί το όνομα του συμβόλου, ενώ κάποιος άλλος δείχνει στη διαδικασία/συνάρτηση που αντιπροσωπεύει το σύμβολο, εάν φυσικά αντιπροσωπεύει κάποια (Βλέπε Σχήμα 1.4). Το



Σχήμα 1.4: Η εσωτερική δομή ενός συμβόλου.

όνομα ενός συμβόλου μπορεί να είναι όσο μεγάλο θέλουμε. Οι αποδεκτοί χαρακτήρες τους οποίους μπορούμε να εισάγουμε στο όνομα ενός συμβόλου είναι τα γράμματα του λατινικού αλφαβήτου, τα 10 ψηφία του ινδο-αραβικού αριθμητικού συστήματος[5] και οι χαρακτήρες `!`, `$`, `%`, `&`, `*`, `+`, `-`, `.`, `/`, `<`, `=`, `>`, `?`, `@`, `[`, `]`, `^`, `_`, `{`, `}` και `~`. Αν θέλουμε να συμπεριλάβουμε κάποιον άλλο χαρακτήρα στο όνομα ενός συμβόλου ή εάν θέλουμε ένα πεζό γράμμα να διατηρηθεί πεζό, θα πρέπει να βάλουμε πριν απ' αυτά το *χαρακτήρα διαφυγής* (*escape character*) \ ή να τα τοποθετήσουμε εντός ενός ζεύγους από αγκύλες διαφυγής ||.

- Να δημιουργήσετε αρκετά σύμβολα με περίεργα ονόματα, για να εξοικειωθείτε. Έπειτα, να εισάγετε ένα σύμβολο στο οποίο θα εναλλάσσονται κεφαλαίοι και πεζοί χαρακτήρες.

1.2.4 Λίστες

Η *λίστα* είναι ο πιο σημαντικός τύπος δεδομένων της Lisp (αρκεί να σκεφτούμε ότι από τις λίστες πήρε η γλώσσα το όνομά της). Ας κάνουμε λίγο πιο συμπαγή τον ορισμό που δώσαμε στην αρχή της παραγράφου μιλώντας τώρα για *λίστα s-expression* (*list s-expression*):

> *Μία αριστερή παρένθεση ακολουθούμενη από μηδέν ή περισσότερες s-expressions ακολουθούμενη από μία δεξιά παρένθεση είναι μία λίστα s-expression.*

Με βάση τον ορισμό που μόλις είδατε, πείτε εάν τα ακόλουθα είναι λίστες[6]:

1. `LIST`
2. `(NOT A LIST)`
3. `((LIST) (LIST) LIST)`
4. `()`

[5] Πρωτοεμφανίστηκαν στην Ινδία τον 6ο αιώνα μ.Χ. και αργότερα υιοθετήθηκαν απ' τους Άραβες τον 8ο αιώνα μ.Χ.

[6] Θα χρησιμοποιούμε τον όρο *λίστα* για να αναφερόμαστε σε λίστες s-expressions.

5. `((abc`
6. `)(`

Γίνεται προφανές, ότι αφού η λίστα είναι μία s-expression, μια λίστα μπορεί να είναι στοιχείο μιας λίστας. Για παράδειγμα, η s-expression `(1 (2 3.3) 4)` είναι μία λίστα με τρία στοιχεία, δύο αριθμούς και μία λίστα. Με την σειρά του το όρισμα `(2 3.3)` είναι, όπως είπαμε, μία λίστα και έχει δύο στοιχεία τα `2`, `3.3`.

Ερώτηση: Είναι, πιστεύετε, οι λίστες `()` και `(())`το ίδιο;

Απάντηση: Όχι, πρόκειται για δύο διαφορετικές λίστες, καθώς η πρώτη είναι η κενή λίστα, ενώ η δεύτερη είναι μία λίστα με ένα στοιχείο το οποίο τυχαίνει να είναι η κενή λίστα.

Αν εισάγουμε μία λίστα στο Listener της Lisp, θα τη διαβάσει, θα κατασκευάσει το *αντικείμενο-λίστα* (*list object*) που την αναπαριστά και θα προσπαθήσει να αποτιμήσει αυτό το αντικείμενο. Μπορούμε να αποφύγουμε την αποτίμηση του αντικειμένου δηλώνοντας στη Lisp πως ό,τι εισαγάγαμε είναι δεδομένα και το κάνουμε αυτό τοποθετόντας πριν από την s-expression που θέλουμε να μην αποτιμηθεί, το σύμβολο `'` (απόστροφος - δίπλα στο Enter) [7]. Για να δείτε πώς δουλεύει, εισάγετε στο Listener την έκφραση `'(1 (2 3.3) 4)`.

- Δοκιμάστε αρκετά δικά σας quoted παραδείγματα λιστών. Τοποθετήστε καθόλου ή και πολλές αποστρόφους για να δείτε τί συμβαίνει. Να διαβάζετε το μήνυμα σφάλματος όποτε αυτό υπάρχει και να προσπαθείτε να το ερμηνεύσετε. Πληκτρολογώντας `:a` μετά από κάποιο σφάλμα μπορείτε να κάνετε abort στον debugger έτσι ώστε να μη συσσωρεύονται πολλές εκκρεμούσες εκτελέσεις (γιατί μετά από κάθε σφάλμα ο debugger περιμένει εντολή απ' τον χρήστη για το πώς να συνεχίσει).

Όταν δίνουμε μία quoted λίστα, ο Listener συνεχίζει να εκτελεί κατά τα γνωστά το read-eval-print loop του, δηλαδή κατασκευάζει ένα quoted αντικείμενο, αλλά *η τιμή ενός quoted αντικειμένου είναι το ίδιο το αντικείμενο*, και ως εκ τούτου τυπώνει τη λίστα που δώσαμε απλώς επιλέγοντας τη δική του αναπαράσταση (π.χ. όλα τα γράμματα τυπώνονται με κεφαλαία και η κενή λίστα `'()` τυπώνεται ως `NIL`).

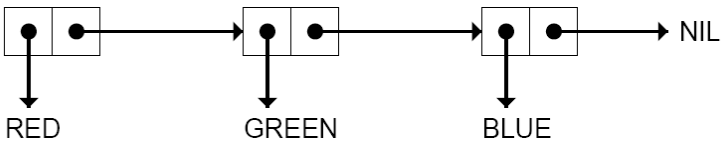
- Δοκιμάστε τις quoted λίστες `'()` και `'(())` στο Listener.

Αν ανοίξουμε μία παρένθεση, γράψουμε ορισμένα στοιχεία και πατήσουμε το carriage return, ο Listener απλώς θα μας πάει σε μια νέα γραμμή χωρίς να διαβάσει αυτό που γράψαμε έως τώρα. Σε τέτοιες περιπτώσεις σημαίνει ότι έχουμε ξεχάσει να κλείσουμε μία ανοικτή παρένθεση. Μόνον αφού κλείσουμε όλες τις ανοικτές παρενθέσεις και πατήσουμε το carriage return, θα διαβάσει η Lisp αυτό που γράψαμε. Σε ορισμένες περιπτώσεις μπορεί να είναι δύσκολο να δούμε αν έχουμε κλείσει όλες τις ανοικτές παρενθέσεις. Κάθε φορά που πληκτρολογούμε μία δεξιά παρένθεση, το περιβάλλον Lispworks μιας διευκολύνει χρωματίζοντας με πράσινο χρώμα την αντίστοιχη αριστερή παρένθεση που μόλις κλείσαμε. Ένας άλλος τρόπος για να μην χανόμαστε είναι να κρατάμε στο μυαλό μας έναν counter που θα ξεκινάει απ' το μηδέν, θα αυξάνει κατά ένα για κάθε αριστερή παρένθεση και θα μειώνεται κατά ένα για κάθε δεξιά. Έτσι, όταν ο counter γίνει και πάλι μηδέν θα ξέρουμε ότι η λίστα μας έχει κλείσει.

[7] Θα λέμε ότι η s-expression είναι quoted.

- Για την λίστα `(1 ( ) (2 (3) 4) 5 ( ( (6) ) 7) )` να κάνετε τα εξής:
  1. Να εξετάσετε τις παρενθέσεις με τη μέθοδο του counter.
  2. Πληκτρολογήστε την (quoted, δηλαδή με απόστροφο) χωρίς την τελευταία της παρένθεση στο Listener και πατήστε το carriage return (δώστε προσοχή και στο πώς χρωματίζονται οι παρενθέσεις καθώς τις κλείνετε). Τί παρατηρείτε;
  3. Εξετάστε τί συμβαίνει αν την εισάγεται χωρίς απόστροφο.

Εάν έχουμε πληκτρολογήσει μία λίστα που καταλαμβάνει αρκετές γραμμές και για κάποιο λόγο θέλουμε να τη σβήσουμε, μπορούμε να σβήσουμε τα πάντα με το Backspace και να επιστρέψουμε στην αρχική γραμμή ή εναλλακτικά μπορούμε να πατήσουμε το κουμπί Break, που βρίσκεται στο toolbar του debugger και εν συνεχεία να δώσουμε `:a` (abort), ή όποια άλλη επιλογή επιθυμούμε. Δοκιμάστε το στο Listener για να το συνηθίσετε.



Σχήμα 1.5: Η εσωτερική αναπαράσταση της λίστας `(red green blue)`.

Πριν κλείσουμε με αυτή την εργαστηριακή άσκηση, θα ήταν καλό να πούμε δύο λόγια για τον τρόπο με τον όποιο αναπαρίστανται εσωτερικά οι λίστες στη Lisp. Ως μηχανικοί, θα μπορείτε να φανταστείτε ότι η εσωτερική αναπαράσταση μιας λίστας δεν περιλαμβάνει ούτε παρενθέσεις, ούτε μοιάζει καθόλου με την εκτυπώσιμη μορφή της. Αυτό που πραγματικά συμβαίνει στη μνήμη είναι ότι οι λίστες οργανώνονται σε αλυσίδες από *cons cells*, τα οποία μπορούμε να σχεδιάζουμε ως κουτιά. Τα cons cells συνδέονται με *δείκτες* (*pointers*), που σχεδιάζονται ως βέλη. Κάθε cons cell έχει δύο δείκτες, έναν προς τη θέση μνήμης του στοιχείου της λίστας στο οποίο αντιστοιχεί και έναν προς το επόμενο cons cell στην αλυσίδα (Βλέπε Σχήμα 1.5). Επομένως, γι΄ αυτό λέμε ότι μια λίστα μπορεί να περιέχει αριθμούς, σύμβολα και λίστες, γιατί πολύ απλά κάθε cons cell της μπορεί να έχει ένα δείκτη προς μια θέση μνήμης που περιέχει έναν αριθμό, ένα σύμβολο ή ακόμα και προς ένα cons cell κάποιας άλλης λίστας[8].

Η Lisp υποστηρίζει και άλλους τύπους δεδομένων, όπως οι χαρακτήρες, οι συμβολοσειρές, οι πίνακες (arrays) και οι δομές (structures), αλλά περισσότερα γι' αυτούς θα δούμε σε μετέπειτα εργαστηριακές ασκήσεις.

[8] Γενικά, οι λίστες που περιέχουν άλλες λίστες καλούνται *εμφωλευμένες* (*nested lists*), ενώ οι λίστες που δεν είναι *εμφωλευμένες* καλούνται *επίπεδες* (*flat lists*).