Κεφάλαιο 4

Εργαστηριακή Άσκηση 4

Όπου θα παρουσιάσουμε τα conditionals και τις εντολές επανάληψης.

4.1 Conditionals

Η λήψη αποφάσεων αποτελεί ένα θεμελιώδες στοιχείο του υπολογισμού. Όλα τα μη τετριμμένα προγράμματα, σε οποιαδήποτε γλώσσα προγραμματισμού και να έχουν υλοποιηθεί, παίρνουν αποφάσεις. Σε αυτή την εργαστηριακή άσκηση, μεταξύ άλλων, θα δούμε πώς μπορούμε να χρησιμοποιούμε κάποιες ειδικές συναρτήσεις λήψης αποφάσεων, οι οποίες καλούνται *conditionals*, και επιλέγουν την τιμή που θα επιστρέψουν μεταξύ κάποιου συνόλου εναλλακτικών εκφράσεων βασιζόμενες στην τιμή μίας ή περισσοτέρων *predicate expressions*. Να θυμίσουμε σε αυτό το σημείο ότι μία predicate expression (δηλαδή, ένα κατηγόρημα), είναι μία έκφραση της οποίας η τιμή ερμηνεύεται ως *αληθής ή ψευδής*.

4.1.1 IF conditional

Το πιο απλό και συχνότερα χρησιμοποιούμενο conditional είναι το `IF` conditional. Η γενική σύνταξη ενός `if` τύπου έχει ως εξής:

`(if` *test_form then_form else_form*`)` Εάν η επιστρεφόμενη τιμή του τύπου *test_form* είναι `nonNIL`, αποτιμάται ο τύπος *then_form*, αλλιώς, αν είναι `NIL`, αποτιμάται ο τύπος *else_form*.

Ας δούμε ένα πολύ απλό παράδειγμα:

```
;;; Day-or-date-fun: Checks if its argument is a day or a date
(defun day-or-date-fun (day-or-date)
  ;; Here starts the body of the function
  (if (symbolp day-or-date) 'day 'date)); symbolp predicate
```

Ανοίξτε τον Editor του Lispworks πηγαίνοντας: Tools > Editor, στο menu bar, ή πατώντας το κουμπί που βρίσκεται δίπλα σε αυτό του Listener στο toolbar. Μπορείτε, εάν θέλετε, να μικρύνετε τα παράθυρα του Listener και του Editor, έτσι ώστε να μπορείτε να τα βλέπετε ταυτόχρονα. Πατήστε Alt+X· στο κάτω μέρος της οθόνης (echo area) θα εμφανιστεί το μήνυμα ''Extended Command:'' όπου μπορείτε να εισάγετε εντολές του Editor. Μπορείτε να εισάγετε όλο το όνομα της

εντολής και να πατήσετε το carriage return, ή μέρος του ονόματος και να πατήσετε το tab. Εάν ακολουθήσετε τον δεύτερο τρόπο, ο Editor θα επεκτείνει την εντολή υποθέτοντας τί σκοπεύετε να πληκτρολογήσετε και αν ξαναπατήσετε το tab θα σας εμφανίσει μία λίστα πιθανών επιλογών. Εσείς εισάγετε την εντολή ''New Buffer'' και μόλις εμφανιστεί το παράθυρο του νέου buffer πατήστε πάλι Alt+X και δώστε την εντολή ''Lisp Mode''. Όλη αυτή η διαδικασία έγινε για να κάνουμε τον τρέχον buffer του Editor να δουλεύει σε Lisp Mode, έτσι ώστε να χρωματίζει τις λέξεις κλειδιά και τις παρενθέσεις. Στον Πίνακα 4.1 παρουσιάζονται οι συχνότερα χρησιμοποιούμενες συντομεύσεις εντολών του Lispworks Editor. Καλό θα ήταν

Ctrl+G	Abort current command
Ctrl+Break	Break the current process
Alt+X	Type command explicitly
Ctrl+H	Access help
Ctrl+X followed by Ctrl+S	Save current buffer to file
Ctrl+X followed by S	Save all buffers to their relevant files
Alt+Ctrl+L	Select the previous buffer displayed
Ctrl+X Ctrl+F *file_name*	Load file, specified by *file_name*, for editing
Ctrl+A	Move the cursor to the beginning of the line
Ctrl+E	Move the cursor to the end of the line
Ctrl+V	Scroll one screen forward
Alt+V	Scroll one screen backward
Alt+Shift+<	Move to the beginning of the buffer
Alt+Shift+>	Move to the end of the buffer
Ctrl+K	Kill text from the cursor to the end of the line
Ctrl+K Ctrl+K	Kills a whole line if typed at the beginning of it
Ctrl+Shift+_	Undo the previous command
Ctrl+Space	Sets a mark for copying or cutting text. You can skip this command by just selecting with the mouse the text to be copied or cutted
Ctrl+W	Cuts the text between the mark set by Ctrl+Space and the current position of the cursor
Alt+W	Copies the text between the mark set by Ctrl+Space and the current position of the cursor
Ctrl+Y	Pastes the cut or copied text
Ctrl+Shift+C	Compiles the current form
Ctrl+Shift+R	Compiles a selected region
Ctrl+Shift+B	Compiles and evaluates all forms in the current buffer

Πίνακας 4.1: Συντομεύσεις των εντολών του Editor.

να μελετήσετε το User Guide του Editor το οποίο μπορείτε να βρείτε πηγαίνοντας Help > Manuals..., στο menu bar, ή σε pdf μορφή πηγαίνοντας Start\All Programms\LispWorks 5.0 Personal\Printable Documentation\i - Editor User

Guide. Πληκτρολογήστε τώρα τη συνάρτηση `day-or-date-fun` στον Editor, μαζί με τα σχόλια. Έχουμε εισαγάγει τρεις διαφορετικούς τρόπους σχολιασμού χρησιμοποιώντας τα ‘`;`’, ‘`;;`’, ‘`;;;`’.

`;` Το semicolon χρησιμοποιείται για σχόλια που ακολουθούν κάποιες εντολές. Ως σχόλιο θεωρείται ό,τι ακολουθεί το ‘`;`’ μέχρι το τέλος της τρέχουσας γραμμής.

`;;` Τα δύο semicolons χρησιμοποιούνται για σχόλια που βρίσκονται εντός του ορισμού των συναρτήσεων και καταλαμβάνουν από μόνα τους μία ολόκληρη γραμμή. Συνηθίζεται να ξεκινούν απ’ το ίδιο σημείο που ξεκινάει ο τύπος τον οποίο περιγράφουν.

`;;;` Τα τρία semicolons χρησιμοποιούνται για σχόλια που βρίσκονται εκτός του ορισμού των συναρτήσεων· κι αυτά, επίσης, καταλαμβάνουν από μόνα τους μία ολόκληρη γραμμή. Συνηθίζεται να ξεκινούν απ’ την αρχή της γραμμής στην οποία βρίσκονται.

Πηγαίνετε File > Save As..., στο menu bar, και αποθηκεύστε το αρχείο ως first_prog.lisp. Πατήστε, εν συνεχεία, Ctrl+Shift+B ή το κουμπί με τον κεραυνό και το κείμενο στο toolbar του Editor (Compile Buffer), για να κάνετε compile τα περιεχόμενα του buffer[1]. Αν όλα έχουν πάει καλά, η μεταγλώττιση θα έχει ολοκληρωθεί χωρίς σφάλματα και τώρα είστε έτοιμοι να μεταβείτε και πάλι στο Listener για να τρέξετε τη συνάρτηση day-or-date-fun που μόλις ορίσατε. Με την μέχρι τώρα εμπειρία που έχετε αποκτήσει δεν πρέπει να είναι δύσκολο να καταλάβετε τί ακριβώς κάνει η συνάρτηση αυτή. Να πραγματοποιήσετε κάποιες δοκιμές για να βεβαιωθείτε ότι όλα είναι όπως τα σκεφτήκατε. Γενικά, είναι πολύ σημαντικό να γνωρίζετε καλά τα διαφορετικά εργαλεία τα οποία σας παρέχει ένα περιβάλλον ανάπτυξης εφαρμογών, όπως το Lispworks για τη Lisp, και να αποκτήσετε μια κάποια ευχέρεια στο χειρισμό τους. Για το λόγο αυτό σας συνιστούμε ανεπιφύλακτα, να αφιερώσετε κάποιο χρονικό διάστημα μελετώντας τα manual που συνοδεύουν το Lispworks και να πειραματιστείτε πάνω σε πραγματικές εφαρμογές εξερευνώντας τις δυνατότητές του. Για παράδειγμα, μία πολύ απλή αλλά συνάμα σημαντική ρύθμιση, είναι αυτή της γραμματοσειράς του Listener και του Editor. Μπορείτε να την αλλάξετε πηγαίνοντας Tools > Preferences > Font > Select Font..., στο menu bar[2].

Τώρα που είδατε πώς μπορούμε να αποθηκεύουμε τις πολύ ενδιαφέρουσες εφαρμογές μας για να μη χρειάζεται να τις πληκτρολογούμε ξανά και ξανά, είναι ώρα να δούμε το `if` conditional σε δράση.

- Να επιστρέψετε στον Editor και να δημιουργήσετε ένα καινούριο buffer. Μπορείτε να το κάνετε είτε πηγαίνοντας File > New, είτε πατώντας το αριστερότερο κουμπί του toolbar (έχει για εικονίδιο μία λευκή σελίδα). Πατήστε Alt+Ctrl+L αρκετές φορές, για να δείτε πώς μπορείτε να μεταφέρεστε απ’ τον έναν ανοικτό buffer στον άλλο. Το ίδιο μπορείτε να κάνετε και με τα βέλη του toolbar του Editor ή ακόμα και να επιλέξετε ένα buffer από μία λίστα με αυτούς που είναι ανοικτοί (το μικρό βελάκι που δείχνει προς τα

[1] Αυτό που βλέπετε στον Editor δεν είναι το ίδιο το αρχείο, αλλά τα περιεχόμενά του, τα οποία αποθηκεύονται προσωρινά σε έναν buffer, γι’ αυτό χρησιμοποιούμε τον όρο *buffer* αντί του όρου *αρχείο*.

[2] Δοκιμάστε π.χ. την Courier New, με κανονικό στυλ και μέγεθος 10· είναι αρκετά ευανάγνωστη.

κάτω, δίπλα από κάθε μεγάλο, γαλάζιο βέλος). Αφού επιστρέψετε στο νέο buffer, να πληκτρολογήσετε τους ακόλουθους τύπους:

```
(defun first-fun (x y)
  (if (> (random 10) 5) (second-fun x y) (third-fun x y)))

(defun second-fun (x y)
  (if (> (random 10) 5) (expt x y) (max x y)))

(defun third-fun (x y)
  (if (and (> x 0)
           (> (random 10) 5))
      (round (- x y))
      (min x y)))
```

Κάντε compile ή evaluate τα περιεχόμενα του τρέχοντος buffer χρησιμοποιώντας είτε τις συντομεύσεις που παρουσιάσαμε προηγουμένως είτε τα αντίστοιχα κουμπιά του toolbar. Τώρα οι συναρτήσεις είναι έτοιμες για να τις χρησιμοποιήσετε. Πηγαίνετε στο Listener, δοκιμάστε μία μία τις συναρτήσεις και παρατηρώντας παράλληλα τον κώδικα προσπαθήστε να αιτιολογήσετε τα αποτελέσματα. Εάν δυσκολεύεστε, καθ' ότι κάποιες primitive συναρτήσεις τις συναντάτε μάλλον για πρώτη φορά, δοκιμάστε να αποτιμήσετε τους επιμέρους τύπους που συνθέτουν τις συναρτήσεις ξεχωριστά· μπορείτε για παράδειγμα να κάνετε copy-paste τον τύπο `(random 10)` στο Listener και να ζητήσετε να σας τον αποτιμήσει για διάφορα ορίσματα έτσι ώστε να καταλάβετε πώς ακριβώς λειτουργεί. Αφού ολοκληρώσετε πολύ προσεκτικά όλα τα παραπάνω, μπορείτε να δοκιμάσετε τις ικανότητές σας με το εξής: Προσπαθήστε να τροποποιήσετε τις συναρτήσεις που ορίσατε, έτσι ώστε όταν καλείτε τη `first-fun`, με οποιαδήποτε ορίσματα, ο έλεγχος να πηγαίνει πάντα στη third-fun και εκεί να καλείται πάντα ο τύπος `(day-or-date-fun 'monday)` που θα πρέπει να τον τοποθετήσετε στη θέση του ενός από τους δύο τύπους της `if` (*Υπόδειξη*: Να τροποποιήσετε κατάλληλα τις πιθανότητες).

- Να αποτιμήσετε τους ακόλουθους τύπους (για κάθε έναν απ' αυτούς σκεφτείτε πρώτα τί νομίζετε ότι θα επιστρέψει):

  1. `(if (oddp 1) 'odd 'even)`
  2. `(if (oddp 2) 'odd 'even)`
  3. `(if t 'test-was-true 'test-was-false)`
  4. `(if f 'test-was-true 'test-was-false)`
  5. `(if (symbolp 'foo) (* 5 5) (+ 5 5))`
  6. `(if (numberp 'foo) (* 5 5) (+ 5 5))`

- Οι ακόλουθες ασκήσεις θα σας βοηθήσουν να εξοικειωθείτε ακόμα περισσότερο με την `if`:

  1. Να ορίσετε και να αποθηκεύσετε σε ένα αρχείο μία συνάρτηση, ονόματι `my-abs`, που να υπολογίζει την απόλυτη τιμή ενός αριθμού x, χωρίς να

χρησιμοποιεί το primitive `abs`. Να μην κάνετε compile το buffer, αντ' αυτού να χρησιμοποιήσετε στο Listener τη συνάρτηση `load`, η οποία ορίζεται ως εξής:

`(load ''`*filename_path*`'')` Φορτώνει στο Listener τα περιεχόμενα του αρχείου που καθορίζεται απ' το *filename_path* (να είστε προσεκτικοί με το *filename_path*, καθώς πρέπει να χρησιμοποιείτε slash '/' για τον διαχωρισμό των φακέλων και όχι backslash '\' που χρησιμοποιούν τα Windows).

2. Να ορίσετε μία συνάρτηση, ονόματι `check-input`, που θα παίρνει ένα όρισμα `x` και θα ελέγχει εάν είναι αριθμός. Εάν είναι, θα καλεί μία συνάρτηση `make-even` με όρισμα το `x`, αλλιώς θα επιστρέφει `NIL`. Η `make-even` θα ελέγχει εάν η προς τα πάνω στρογγυλοποίηση του ορίσματός της (primitive συνάρτηση `round`) είναι περιττός αριθμός (primitive συνάρτηση `oddp`) και εάν είναι, θα του προσθέτει ένα για να τον κάνει άρτιο, αλλιώς θα επιστρέφει τον αριθμό που προέκυψε μετά τη στρογγυλοποίηση (συνιστάται να χρησιμοποιήσετε τη συνάρτηση `let` για να μην είστε υποχρεωμένοι να υπολογίζετε συνεχώς τη στρογγυλοποίηση).
3. Ας υποθέσουμε ότι η Lisp δε διέθετε την primitive λογική συνάρτηση `not` κι ότι εσείς θέλετε απεγνωσμένα να υλοποιήσετε τη λογική συνάρτηση `nand` (not and) για να την εισάγετε σε μια εφαρμογή σας. Θα πρέπει να υλοποιήσετε τη `my-not` χρησιμοποιώντας μόνο μία `if` και χωρίς να καλείτε απ' το σώμα της καμία άλλη συνάρτηση. Αφού την υλοποιήσετε να τη χρησιμοποιήσετε για να ορίσετε τη συνάρτηση `nand`.

4.1.2 COND conditional

Το `COND` είναι το κλασικό conditional της Lisp. Η είσοδός του αποτελείται από οποιονδήποτε αριθμό test και consequent όρων, όπως φαίνεται παρακάτω:

$$
\begin{array}{ll}
\texttt{(COND} & (test_1 \;\; consequent_1) \\
& (test_2 \;\; consequent_2) \\
& (test_3 \;\; consequent_3) \\
& \quad \vdots \qquad\quad \vdots \\
& (test_n \;\; consequent_n)
\end{array}
$$

Η `cond` προσπελαύνει τους όρους ακολουθιακά. Εάν ο όρος $test_i$ αποτιμηθεί σε true, η `cond` αποτιμά τον όρο $consequent_i$, επιστρέφει την τιμή του και δεν προσπελαύνει τους υπόλοιπους όρους. Εάν ο όρος $test_i$ αποτιμηθεί σε false, τότε η `cond` αγνοεί το $consequent_i$ και περνάει στον επόμενο όρο. Επομένως, θα μπορούσαμε να πούμε ότι η `cond`, στην ουσία, αποτιμάει μόνο το consequent μέρος του πρώτου αληθούς test όρου. Προφανώς, όσο συναντάει test όρους που αποτιμούν σε false δεν κάνει τίποτα και απλώς προχωράει στους επόμενους. Εάν, τέλος, δε βρει κανέναν αληθή test όρο, τότε απλώς επιστρέφει `NIL`.

- Πληκτρολογήστε την ακόλουθη συνάρτηση:

```
(defun compare (x y)
  (cond ((equal x y) 'numbers-are-the-same)
        ((< x y) 'first-is-smaller)
        ((> x y) 'first-is-bigger)))
```

Να μελετήσετε προσεκτικά τη σύνταξη της `cond` και να τρέξετε αρκετά παραδείγματα χρησιμοποιώντας τη συνάρτηση `compare` που ορίσατε.

- Τροποποιήστε την `cond` έτσι ώστε να κάνει τα εξής:

  1. Αν οι αριθμοί `x`, `y` είναι ίσοι θα καλείται μία συνάρτηση που θα επιστρέφει το μέσο όρο τους.
  2. Στις περιπτώσεις όπου `x`, `y` είναι άνισοι, θα καλείται μία συνάρτηση που θα επιστρέφει την απόλυτη τιμή της διαφοράς τους.

Γενικά, η `cond` και η `if` είναι παρόμοιες συναρτήσεις. Μπορεί η `cond` να μοιάζει πιο ευέλικτη καθώς δέχεται οποιονδήποτε αριθμό όρων, παρ΄ όλα αυτά το ίδιο ακριβώς πράγμα μπορεί να γίνει και με εμφωλευμένες `if`.

Υπάρχει ένα κόλπο με το οποίο μπορούμε να κάνουμε την `cond` να έχει default τιμή. Αρκεί να εισάγουμε ένα ζεύγος όρων της μορφής `(T consequent)`. Μπορείτε να σκεφτείτε σε ποιό σημείο της `cond` θα πρέπει να τοποθετήσουμε την παραπάνω έκφραση έτσι ώστε ο όρος `consequent` να είναι default;

- Να πληκτρολογήσετε και να δοκιμάσετε για αρκετά παραδείγματα την παρακάτω συνάρτηση:

```
(defun where-is (x)
  (cond ((equal x 'paris) 'france)
        ((equal x 'london) 'england)
        ((equal x 'beijing) 'china)
        (t 'unknown)))
```

Οι παρακάτω ασκήσεις θα σας βοηθήσουν να εξοικειωθείτε περαιτέρω:

1. Υπενθυμίζουμε ότι η γενική μορφή μιας `if` έκφρασης είναι:

   `(if` *`test_form then_form else_form`*`)`

   Μπορείτε να φανταστείτε με ποιό τρόπο μπορούμε να μετατρέψουμε οποιαδήποτε `if` έκφραση σε μία ισοδύναμη `cond` έκφραση (*Υπόδειξη*: Η `cond` θα έχει δύο ορίσματα, δύο, δηλαδή, ζεύγη test και consequent όρων);

2. Να ορίσετε μία ακόμα έκδοση της συνάρτησης απόλυτης τιμής `my-abs`, που είδαμε προηγουμένως, χρησιμοποιώντας τώρα την `cond` αντί της `if`.

3. Να ορίσετε μία συνάρτηση, ονόματι `emphasize`, η οποία θα παίρνει ως όρισμα μία λίστα και αν το πρώτο της στοιχείο είναι η λέξη good θα την ''αλλάζει'' με τη λέξη great, αν είναι η λέξη bad θα την ''αλλάζει'' με τη λέξη awful, ενώ αν δεν είναι καμία απ' τις δύο απλώς θα επιστρέφει τη λίστα χωρίς καμία τροποποίηση.

4. Έστω η ακόλουθη συνάρτηση:

```
(defun compute (op x y)
  (cond ((equal op 'sum-of) (+ x y))
        ((equal op 'product-of) (* x y))
        (t '(unknown operation))))
```

Χωρίς να τους τρέξετε, προσπαθήστε να σκεφτείτε τί επιστρέφουν οι ακόλουθοι τύποι:

i `(compute 'sum-of 3 7)`

ii `(compute 'product-of 2 4)`

iii `(compute 'exp-of 2 4)`

5. Να γράψετε μία συνάρτηση, ονόματι `constrain`, η οποία θα παίρνει τρία ορίσματα x, min, max και θα κάνει τα εξής:

   i Εάν $x < min$, θα επιστρέφει min.

   ii Εάν $x > max$, θα επιστρέφει max.

   iii Εάν $min \leq x \leq max$, θα επιστρέφει το x.

6. Να γράψετε μία συνάρτηση, ονόματι `cycle`, η οποία μετράει κυκλικά απ' το 1 μέχρι το 99. Συγκεκριμένα, όταν καλείται με είσοδο i θα πρέπει να επιστρέφει $i + 1$ εκτός και αν το i είναι ίσο με 99, οπότε θα πρέπει να επανεκκινήσει το μετρητή.

Μπορούμε να συνδυάσουμε τις λογικές συναρτήσεις `and`, `or`, `not` κ.ο.κ. με τις `if` και `cond`, για να κατασκευάσουμε πιο σύνθετους ελέγχους.

- Να μελετήσετε και εν συνεχεία να πληκτρολογήστε και να δοκιμάσετε την ακόλουθη συνάρτηση:

```
(defun how-alike (a b)
  (cond ((equal a b) 'the-same)
        ((and (oddp a) (oddp b)) 'both-odd)
        ((and (not (oddp a)) (not (oddp b))) 'both-even)
        ((and (< a 0) (< b 0)) 'both-negative)
        (t 'not-alike)))
```

- Να κατασκευάσετε μία συνάρτηση, ονόματι `sign` η οποία θα χρησιμοποιεί το `if` ή το `cond` conditional και τις λογικές συναρτήσεις `and` και `or`, για να ελέγχει εάν δύο αριθμοί έχουν το ίδιο πρόσημο. Σε κάθε περίπτωση θα πρέπει να επιστρέφεται το αντίστοιχο μήνυμα.

Ακολουθούν δύο ανακεφαλαιωτικές ασκήσεις:

1. Σε τί φαντάζεστε ότι αποτιμάει η `cond`, εάν δεν της δοθούν καθόλου ορίσματα;

2. Μπορούμε συχνά να γράψουμε την `if` ως ένα συνδυασμό των λογικών συναρτήσεων `and` και `or`, σύμφωνα με τον ακόλουθο τρόπο: Αντικαθιστούμε την έκφραση `(if` *`test true_part false_part`*`)` με την `(or (and` *`test true_part`*`)` *`false_part`*`)`. Εάν δοκιμάσετε όμως την έκφραση `(if (oddp 5) (evenp 7) 'foo)`, θα παρατηρήσετε ότι η μετατροπή αποτυγχάνει να επιστρέψει το σωστό αποτέλεσμα. Γιατί γίνεται αυτό; Σκεφτείτε ένα τρόπο για να διορθώσετε τη μέθοδο αυτή (*Υπόδειξη*: Θα χρειαστεί να χρησιμοποιήσετε τη λογική συνάρτηση `not`).