ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Εαρινό 2017

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι

Τελικό Διαγώνισμα (Κανονική Εξέταση)

Το διαγώνισμα αυτό έχει ερωτήσεις 8 βαθμών συνολικά.

1. Γραμματικές (0.2 + 0.2 + 0.6 = 1 βαθμός)

Έστω η παρακάτω γραμματική χωρίς συμφραζόμενα η οποία γεννά τους όρους της προτασιακής λογικής (propositional logic).

```
<Term>  ::= <Term> ∧ <Term>          — σύζευξη (και)
          | <Term> ∨ <Term>          — διάζευξη (ή)
          | <Term> → <Term>          — συνεπαγωγή
          | ¬ <Term>                 — άρνηση
          | ⊥ | <Var> | ( <Term> )   — ψευδής όρος, μεταβλητή, παρενθέσεις
<Var>   ::= P | Q | ... | S
```

α) Δώστε ένα συντακτικό δένδρο για τη συμβολοσειρά `¬(P∧¬Q)`

β) Δείξτε ότι η γραμματική αυτή είναι διφορούμενη.

γ) Τροποποιήστε τη γραμματική έτσι ώστε οι τελεστές να έχουν τη σωστή προτεραιότητα και προσεταιριστικότητα για τις πράξεις που συμβολίζουν. Η συνεπαγωγή πρέπει να έχει τη μικρότερη προτεραιότητα, έπειτα η διάζευξη, έπειτα η σύζευξη και τέλος η άρνηση πρέπει να έχει τη μεγαλύτερη. Η σύζευξη και η διάζευξη πρέπει να είναι αριστερά προσεταιριστικές και η συνεπαγωγή δεξιά προσεταιριστική.

2. Προγραμματισμός σε ML (1 + 1 = 2 βαθμοί)

α) Να γράψετε σε ML μία *κομψή και αποδοτική* συνάρτηση **`listify`** η οποία να δέχεται δύο ορίσματα. Το πρώτο είναι μία λίστα ακεραίων αριθμών ls και το δεύτερο ένας ακέραιος αριθμός x. Το αποτέλεσμά της πρέπει να είναι μία λίστα από λίστες ακεραίων αριθμών, τέτοιες ώστε:

- η πρώτη λίστα, η τρίτη, η πέμπτη, κ.ο.κ., να περιέχουν αριθμούς μικρότερους του x,
- η δεύτερη λίστα, η τέταρτη, η έκτη, κ.ο.κ., να περιέχουν αριθμούς μεγαλύτερους ή ίσους του x,
- η συνένωση όλων των λιστών να είναι ίση με την αρχική λίστα ls, και
- το αποτέλεσμα να περιέχει όσο το δυνατόν λιγότερες λίστες.

Ακολουθούν δύο παραδείγματα:

```
- listify [3,5,1,8,9,2,1,0,1,7,4] 7;
val it = [[3,5,1],[8,9],[2,1,0,1],[7],[4]] : int list list

- listify [3,5,1,8,9,2,1,0,1,7,4] 3;
val it = [[],[3,5],[1],[8,9],[2,1,0,1],[7,4]] : int list list
```

β) Να γράψετε σε ML μία *κομψή και αποδοτική* συνάρτηση **`maxSumSublist`** η οποία να δέχεται

ως όρισμα μια λίστα από ακέραιους αριθμούς και να επιστρέφει ως αποτέλεσμα το μέγιστο άθροισμα οσωνδήποτε διαδοχικών αριθμών της λίστας. Δύο παραδείγματα παρακάτω:

```
- maxSumSublist [-2, 1, -3, 4, -1, 2, 1, -5, 4];
val it = 6 : int

- maxSumSublist [-1, -2, -3, -4, -5];
val it = 0 : int
```

(Στην SML οι αρνητικοί αριθμοί θα γράφονταν σωστά ως ~2 αντί -2, κ.ο.κ., αλλά για το διαγώνισμά μας έχει μικρή σημασία και είναι καλύτερα να συνεννοούμαστε.) Στο πρώτο παράδειγμα, οι διαδοχικοί όροι `[4, -1, 2, 1]` δίνουν άθροισμα 6 και αυτό είναι το μέγιστο δυνατό. Στο δεύτερο παράδειγμα, όλοι οι όροι είναι αρνητικοί, οπότε το καλύτερο που μπορεί να επιτευχθεί είναι αν δεν πάρουμε κανέναν όρο (κενή υπο-λίστα, άθροισμα μηδέν).

3. Συμπερασμός τύπων στην ML (4 * 0.25 = 1 βαθμός)

Συμπληρώστε τον πίνακα της σελίδας απαντήσεων με τους τύπους των παρακάτω συναρτήσεων:

```
fun foo x y z = z (y x + 1)
fun bar x y   = x (bar y x)
fun doh x y z = z (x y) (y + 1)
fun ugh x y z = x z (y z)
```

4. Εμβέλειες (0.25 + 0.25 + 0.5 = 1 βαθμός)

α) Εξηγήστε τη διαφορά μεταξύ στατικής και δυναμικής εμβέλειας.
β) Δώστε ένα μικρό πρόγραμμα που συμπεριφέρεται διαφορετικά με τις δύο μορφές εμβέλειας.
γ) Για ποιο λόγο η χρήση δυναμικής εμβέλειας υπονοεί την ανάγκη για δυναμικό έλεγχο τύπων; Εξηγήστε με ένα παράδειγμα.

5. Πέρασμα παραμέτρων (4 * 0.25 = 1 βαθμός)

Έστω το παρακάτω πρόγραμμα σε μια υποθετική γλώσσα προγραμματισμού που μοιάζει με τη C. Προσέξτε ότι η αρίθμηση των στοιχείων των πινάκων ξεκινάει από το μηδέν.

```
int A[2] = {0, 0};

void f(int x, int y) {
  x++; A[0]++; y++;
  printf("%d %d %d %d ", x, y, A[0], A[1]);
}

void main() {
  A[0] = 1;
  f(k, A[k]);
  printf("%d %d %d\n", k, A[0], A[1]);
}
```

Συμπληρώστε τον πίνακα της σελίδας απαντήσεων με τις επτά τιμές που θα εκτυπώσει το πρόγραμμα για καθεμία από τις παρακάτω τεχνικές περάσματος παραμέτρων της συνάρτησης `f`.

α) κλήση με τιμή (call by value)
β) κλήση κατ' αναφορά (call by reference)
γ) κλήση με τιμή-αποτέλεσμα (call by value-result)
δ) κλήση κατ' όνομα (call by name)

6. Προγραμματισμός σε Prolog (0.5 + 0.5 + 1 = 2 βαθμοί)

α) Γράψτε σε Prolog τον ορισμό του κατηγορήματος **`running_sum(L, S)`**. Το κατηγόρημα αυτό πρέπει να επιτυγχάνει όταν τα L και S είναι λίστες ακέραιων αριθμών και η λίστα S περιέχει τα τρέχοντα αθροίσματα των όρων της λίστας L. Δηλαδή, το N-οστό στοιχείο της λίστας S θα πρέπει να είναι ίσο με το άθροισμα των N πρώτων στοιχείων της λίστας L. Παράδειγμα χρήσης:

```
?- running_sum([5, 3, 7, -2, 0, 4], S).
S = [5, 8, 15, 13, 13, 17].
```

β) Πόσο ευέλικτο είναι το κατηγόρημα **`running_sum/2`** που δώσατε ως απάντηση στο υποερώτημα (α) αυτού του θέματος; Μπορεί να χρησιμοποιηθεί με δύο δοσμένες λίστες L και S; Με δοσμένη λίστα S και άγνωστη λίστα L; Με δύο άγνωστες λίστες; Χρησιμοποιώντας το κατηγόρημα **`var(X)`** της Prolog, που επιτυγχάνει αν και μόνο αν τη στιγμή της αποτίμησής του ο όρος X είναι μία αδέσμευτη μεταβλητή, τροποποιήστε κατάλληλα (αν χρειάζεται) το **`running_sum/2`** ώστε, επιπλέον του παραδείγματος του υποερωτήματος (α), να μπορεί να χρησιμοποιηθεί και ως εξής:

```
?- running_sum(L, [5, 8, 15, 13, 13, 17]).
S = [5, 3, 7, -2, 0, 4].
```

γ) Ένα γνωστό δύσκολο (NP-complete) πρόβλημα είναι το "subset sum". Σύμφωνα με μία παραλλαγή του, δίνεται ένα σύνολο αριθμών και ζητείται να βρεθεί αν υπάρχει ένα υποσύνολό του με δοθέν άθροισμα στοιχείων.

Γράψτε σε Prolog τον ορισμό του κατηγορήματος **`subset_sum(Set, Sum, Subset)`**. Το πρώτο και το τρίτο όρισμα θα είναι λίστες αριθμών και το δεύτερο θα είναι αριθμός. Το κατηγόρημα πρέπει να επιτυγχάνει όταν η λίστα Subset είναι υποσύνολο της λίστας Set και το άθροισμα των στοιχείων της είναι ίσο με Sum.

Μπορείτε να θεωρήσετε ότι το πρώτο όρισμα (Set) δεν θα περιέχει αδέσμευτες μεταβλητές. Επίσης, αν σας εξυπηρετεί, μπορείτε να θεωρήσετε ότι το Set και το Subset δεν θα περιέχουν περισσότερες φορές τον ίδιο αριθμό και ότι τα μόνα «υποσύνολα» Subset που μας ενδιαφέρουν είναι οι λίστες που προκύπτουν από την αρχική λίστα Set με τη διαγραφή μηδέν ή περισσότερων αριθμών (και όχι όλες οι δυνατές μεταθέσεις αυτών).

Ακολουθούν δύο παράδειγματα χρήσης:

```
?- subset_sum([17, -1, 42], Sum, Subset).
Sum = 58, Subset = [17, -1, 42] ;
Sum = 16, Subset = [17, -1] ;
Sum = 59, Subset = [17, 42] ;
Sum = 17, Subset = [17] ;
Sum = 41, Subset = [-1, 42] ;
Sum = -1, Subset = [-1] ;
Sum = 42, Subset = [42] ;
Sum = 0,  Subset = [].
```

```
?- subset_sum([8, 12, 42, 17, 5, 24], 37, Subset).
Subset = [8, 12, 17] ;
Subset = [8, 5, 24] ;
false.
```

Καλή επιτυχία!