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

Εαρινό Εξάμηνο 2025

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι

Άσκηση 1

Καταληκτική ημερομηνία και ώρα ηλεκτρονικής υποβολής: 14/4/2025, 23:59:59

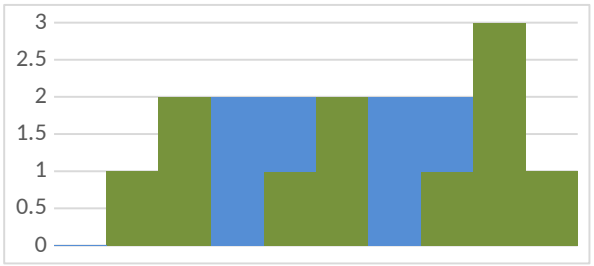
Αποθέματα νερού (0.25 βαθμοί)

Σε μια επαρχία που πλήττεται από μακρά περίοδο ξηρασίας, οι τοπικές αρχές αναζητούν τρόπους να αξιοποιήσουν το νερό της βροχής. Πιο συγκεκριμένα, θέλουν να υπολογίσουν πόσο νερό μπορεί να συγκεντρωθεί στις "κοιλότητες" που σχηματίζονται ανάμεσα στους λόφους μιας περιοχής. Ως περιοχή θεωρείται μια ακολουθία από **N** λόφους ($1 \leq N \leq 20000$)**,** όπου κάθε λόφος i έχει ένα ύψος $\mathbf{h_i}$ ($0 \leq h_i < 100000$). Η βροχόπτωση γεμίζει τα "κενά" μεταξύ των λόφων με νερό, και το ζητούμενο είναι να βρεθεί ο μέγιστος δυνατός όγκος του νερού που μπορεί να συγκρατηθεί.

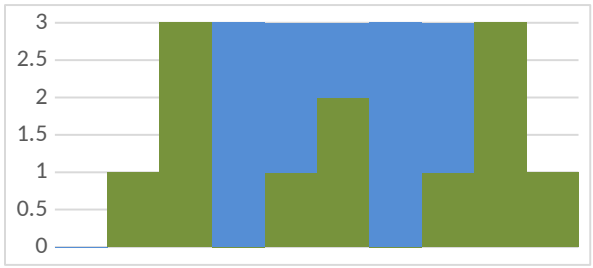
Οι τοπικές αρχές σας προσλαμβάνουν για να γράψετε ένα πρόγραμμα σε **συναρτησιακή ML** το οποίο διαβάζει τα απαραίτητα δεδομένα από ένα αρχείο, υπολογίζει το μέγιστο δυνατό όγκο νερού (σε "μονάδες") που μπορεί να συγκρατηθεί ανάμεσα στους λόφους μιας περιοχής, και εκτυπώνει το αποτέλεσμα. Με τον όρο "συναρτησιακή ML" εννούμε ότι το πρόγραμμά σας **δεν** μπορεί να χρησιμοποιεί references ή βιβλιοθήκες της ML (π.χ. Array) που βασίζονται σε references.

Τα στοιχεία εισόδου διαβάζονται από ένα αρχείο κειμένου του οποίου η πρώτη γραμμή περιέχει το μήκος N μιας περιοχής. Η δεύτερη γραμμή περιέχει N αριθμούς χωρισμένους με κενά που αντιστοιχούν στο ύψος κάθε λόφου. Για παράδειγμα, τα αρχεία **map1.txt** και **map2.txt** περιέχουν τους παρακάτω χάρτες περιοχών (η εντολή **cat** είναι εντολή του Unix). Όπως φαίνεται και σχηματικά, στην πρώτη περιοχή μπορούν να αποθηκευτούν 6 μονάδες νερού, ενώ στην δεύτερη 11.

```
$ cat map1.txt
10
0 1 2 0 1 2 0 1 3 1
```




```
$ cat map2.txt
10
0 1 3 0 1 2 0 1 3 1
```



Τα προγράμματα σας θα πρέπει να τρέχουν όπως φαίνεται παρακάτω.

Σε MLton ή σε OCaml

```
$ ./rain map1.txt
6


$ ./rain map2.txt
11
```

Σε SML/NJ

```
- rain "map1.txt";
6
val it = () : unit

- rain "map2.txt";
11
val it = () : unit
```

Συμβόλαια εργασίας (0.25+0.25 = 0.5 βαθμοί)

Μια εταιρεία τεχνολογίας σκοπεύει να προσλάβει **N** ($1 \leq N \leq 100000$) ηλεκτρολόγους μηχανικούς. Η εταιρία μπορεί να προσφέρει δύο τύπους συμβολαίων σε κάθε εργαζόμενο: **Συμβόλαιο Α** ή **Συμβόλαιο Β**. Κάθε συμβόλαιο έχει διαφορετικό κόστος για την εταιρία και παρέχει διαφορετικές παροχές στον εργαζόμενο, επηρεάζοντας έτσι την παραγωγικότητα του και το κέρδος της εταιρίας.

Η εταιρία μπορεί να κάνει ακριβώς **M** συμβόλαια τύπου Α και **K = N - M** συμβόλαια τύπου Β. Για κάθε εργαζόμενο i, η επιλογή του συμβολαίου Α επιφέρει καθαρό κέρδος στην εταιρία $\mathbf{a}_i$ και η επιλογή του συμβολαίου Β επιφέρει καθαρό κέρδος $\mathbf{b}_i$ ($0 \leq a_i, b_i < 100000$). Ζητείται να γράψετε δύο προγράμματα (ένα σε **C/C++** και ένα σε **ML**) τα οποία διαβάζουν τα απαραίτητα δεδομένα από ένα αρχείο κειμένου και υπολογίζουν τη μέγιστη τιμή του συνολικού κέρδους που μπορεί να επιτύχει η εταιρεία από την κατανομή των συμβολαίων. Με άλλα λόγια, το πρόγραμμά σας θα πρέπει να επιλέγει σε ποιους μηχανικούς θα δοθούν τα **M** συμβόλαια Α και σε ποιους τα **K** συμβόλαια Β, ώστε να μεγιστοποιείται το συνολικό κέρδος. Στην άσκηση αυτή δεν υπάρχει κάποιος περιορισμός όσον αφορά το κομμάτι της ML που μπορείτε να χρησιμοποιήσετε στον κώδικά σας.

Τα στοιχεία εισόδου διαβάζονται από ένα αρχείο του οποίου η πρώτη γραμμή περιέχει τους ακεραίους **M** και **K**. Κάθε μία από τις επόμενες **N** γραμμές περιέχει δύο ακεραίους χωρισμένους με κενά όπου ο πρώτος είναι το καθαρό κέρδος της εταιρείας αν ο εργαζόμενος υπογράψει το Συμβόλαιο Α και ο δεύτερος το καθαρό κέρδος αν ο εργαζόμενος υπογράψει το Συμβόλαιο Β.

Για παράδειγμα, τα περιεχόμενα του αρχείου **contracts.txt** είναι τα εξής:

```
$ cat contracts.txt
2 2
5 3
10 9
6 1
2 2
```

Σε αυτό το παράδειγμα, ο βέλτιστος τρόπος κατανομής είναι να δοθούν συμβόλαια Α στον πρώτο και στον τρίτο εργαζόμενο, και συμβόλαια Β στον δεύτερο και στον τέταρτο. Το μέγιστο κέρδος είναι 22.

Τα προγράμματα σας θα πρέπει να τρέχουν όπως φαίνεται παρακάτω.

Σε C/C++, MLton, ή σε OCaml

```
$ ./hire contracts.txt
22
```

Σε SML/NJ

```
- hire "contracts.txt";
22
val it = () : unit
```

ML σε ML (0.5 βαθμοί)

Έχει σχεδόν βραδιάσει, κι εσείς εξακολουθείτε να βρίσκεστε στη βιβλιοθήκη της ΣΗΜΜΥ. Αποφασίζετε ότι έχετε κουραστεί να μελετάτε τόσες διαφορετικές γλώσσες προγραμματισμού – ήρθε η ώρα να φτιάξετε τη δική σας! Γνωρίζετε ότι η ML είναι ιδιαίτερα βολική για την υλοποίηση γλωσσών, οπότε αποφασίζετε να τη χρησιμοποιήσετε για το εγχείρημα σας.

Ξεκινάτε ορίζοντας ένα *αφηρημένο συντακτικό δέντρο* (Abstract Syntax Tree) που αναπαριστά αριθμητικές και λογικές (Boolean) εκφράσεις:

```
datatype expr = IntLit of int             (* integer literals *)
              | BoolLit of bool           (* boolean literals *)
              | Bop of bop * expr * expr  (* binary operators *)
              | ITE of expr * expr * expr (* if-then-else expressions *)
```

Όπου ο τύπος **bop** αναπαριστά τους παρακάτω δυαδικούς τελεστές:

```
datatype bop = Plus | Minus | Mult | Or | And | Xor | Eq | Lt | Gt
```

Για την αναπαράσταση των τιμών που προκύπτουν από την αποτίμηση των εκφράσεων ορίζετε έναν νέο τύπο **value**, ο οποίος μπορεί να είναι είτε ένας ακέραιος είτε μια λογική τιμή.

```
datatype value = VInt of int | VBool of bool
```

Στη συνέχεια, γράφετε μια συνάρτηση **eval** (τύπου **expr -> value**) που αποτιμά τις εκφράσεις της παραπάνω γλώσσας:

```
fun eval (IntLit n) = VInt n
  | eval (BoolLit b) = VBool b
  | eval (Bop (bop,e1,e2)) =
      (case (eval e1, eval e2) of
         (VInt n1, VInt n2) =>
           (case bop of
              Plus => VInt (n1 + n2)
            | Minus => VInt (n1 - n2)
             …
            | _ => raise RuntimeError "Operand type mismatch")
       | (VBool b1, VBool b2) =>
           (case bop of
              And => VBool (b1 andalso b2)
             …
            | _ => raise RuntimeError "Operand type mismatch")
       | (_, _) => raise RuntimeError "Operand type mismatch")
  | eval (ITE (e1,e2,e3)) = …
```

Ωστόσο, παρατηρείτε ότι αν οι εκφράσεις δεν έχουν ορίσματα αναμενόμενων τύπων (π.χ. μια έκφραση που κάνει πρόσθεση ενός ακεραίου με μία λογική τιμή), τότε η αποτίμησή τους θα εγείρει μια εξαίρεση. Μια συμφοιτήτριά σας, που επίσης μελετάει για το μάθημα «Γλώσσες Ι», σας προτείνει να γράψετε έναν ***type checker*** που να ελέγχει ότι οι εκφράσεις έχουν σωστούς τύπους και κατ' επέκταση η αποτίμηση τους δε θα οδηγήσει ποτέ σε εξαίρεση κατά το χρόνο εκτέλεσης.

Ξεκινάτε ορίζοντας έναν τύπο στην ML για τους πιθανούς διαφορετικούς τύπους της γλώσσας σας, δηλαδή τους ακεραίους και τους Booleans.

```
datatype typ = Int | Bool
```

Στη συνέχεια, γράφετε την συνάρτηση **typeCheck**, με τύπο **expr -> typ,** η οποία επιστρέφει τον τύπο μια έκφρασης, εάν αυτός υπάρχει, αλλιώς εγείρει την εξαίρεση **TypeError** όταν συναντά ασυμβατότητες τύπων.

```
fun typeCheck (IntLit _) = Int
    | typeCheck (BoolLit _) = Bool
    | typeCheck (Bop (bop,e1,e2)) =
        (case bop of
           (Plus | Minus | Mult) =>
              (case (typeCheck e1, typeCheck e2) of
                 (Int, Int) => Int
                 _ => raise TypeError "Operator expects Int arguments")
           (Or | And | Xor) => …
           (Lt | Gt | Eq) => …)
    | typeCheck (ITE (e1, e2, e3)) =
        (case (typeCheck e1, typeCheck e2, typeCheck e3) of
           (Bool, t_then, t_else) =>
              if t_then = t_else then t_then
              else raise TypeError "Branches must have the same type"
           | _ => raise TypeError "Condition must be Bool")
```

Έτσι, αν η **typeCheck** επιστρέψει επιτυχώς, μπορείτε να είστε βέβαιοι εκ των προτέρων ότι η **eval** δεν θα εγείρει εξαίρεση λόγω ασυμβατότητας τύπων στους τελεστές.

Είστε χαρούμενοι με πρώτα σας αποτελέσματα, όμως παρατηρείτε ότι η γλώσσα σας είναι πολύ περιορισμένη. Για παράδειγμα, δεν έχει μεταβλητές ή συναρτήσεις! Αποφασίζετε να επεκτείνετε την γλώσσα σας με μεταβλητές, ορισμούς συναρτήσεων (επισημειωμένες με τον τύπο του ορίσματος), και ορισμούς **let**, όμοια με τα αντίστοιχα της ML. Οι εκφράσεις της γλώσσας σας μπορούν πλέον να είναι συναρτήσεις και κατ' επέκταση να έχουν και τύπους συναρτήσεων. Οι νέοι σας ορισμοί έχουν ως εξής:

```
datatype typ = … | Arrow of typ * typ

datatype expr = … | Var of string | LetIn of string * expr * expr
                  | Fun of string * typ * expr | App of expr * expr
```

Οι μεταβλητές όμως περιπλέκουν λίγο τα πράγματα. Παρατηρείτε τα εξής:

- Για να υπολογίσετε τον τύπο ή την τιμή μιας έκφρασης χρειάζεστε ένα *περιβάλλον* που να αντιστοιχίζει τις διαθέσιμες μεταβλητές στον τύπο ή την τιμή τους αντίστοιχα. Επειδή για την ώρα δεν σας πολυνοιάζει η αποδοτικότητα των συναρτήσεων σας, αποφασίζετε να υλοποιήσετε τα περιβάλλοντα με χρήση λιστών. Με αυτόν τον τρόπο, ένα περιβάλλον αναπαρίσταται από τον τύπο **(string * 'a) list**, δηλαδή μια λίστα από ζεύγη όπου το πρώτο στοιχείο είναι το όνομα μιας μεταβλητής και το δεύτερο ο τύπος ή η τιμή της. Π.χ., στο περιβάλλον **[(x, Int), (y, Int)]** η έκφραση **Bop(Plus, Var "x", Var "y")** έχει τύπο **Int**, και στο περιβάλλον **[(x, VInt 5), (y, VInt 6)]** η ίδια έκφραση αποτιμάται στην τιμή **(VInt 11)**.

  Για ευκολία, ορίζετε και ένα συνώνυμο τύπου ώστε να αναφέρεστε στον τύπο ενός περιβάλλοντος πιο συνοπτικά:

  ```
  type 'a env = (string * 'a) list
  ```

  Πλέον, η συνάρτηση **typeCheck** έχει τύπο **typ env -> expr -> typ** και η συνάρτηση **eval** έχει τύπο **value env -> expr -> value**.

- Η γλώσσα που δημιουργήσατε έχει συναρτήσεις πρώτης τάξης, κατ' επέκταση η τιμή μιας έκφρασης μπορεί να είναι μια συνάρτηση. Αρχίζετε να σκέφτεστε πως πρέπει να αναπαρασταθεί η τιμή μιας συνάρτησης, και γράφετε τα παρακάτω παραδείγματα στη γλώσσα σας για να καταλάβετε καλύτερα το πρόβλημα.

  let foo = (let x = 6*6 in fun z => z + x) in foo 6

  let bar = (fun x => fun y => x * y + x) in let baz = bar 21 in baz 1

  Αναρωτιέστε ποια θα πρέπει να είναι η τιμή των μεταβλητών **foo** και **baz**. Και οι δύο είναι συναρτήσεις, όμως η τιμή τους δεν μπορεί να είναι απλά ο ορισμός της συνάρτησης, δηλαδή **fun z => z + y** και **fun y => x * y + x** αντίστοιχα, καθώς αυτός περιέχει μεταβλητές οι οποίες είναι ορισμένες στο εξωτερικό της συνάρτησης και πρέπει να είναι δεμένες με κάποια τιμή.

  Η φίλη σας, που συνεχίζει να μελετάει στην βιβλιοθήκη, και έχει ήδη διαβάσει το παρακάτω μάθημα, σας λέει ότι μια συνάρτηση αποτιμάται σε ένα **closure**, δηλαδή μια συνάρτηση μαζί με το περιβάλλον της το οποίο περιέχει τις τιμές των μεταβλητών που είναι ορισμένα στο περιβάλλον τη στιγμή της δημιουργίας της συνάρτησης.

  Επεκτείνετε λοιπόν τον ορισμό των τιμών με closures, ως εξής:

  ```
  datatype value = … | VClo of value env * string * expr
  ```

  Έτσι η τιμή της μεταβλητής **foo** είναι:

  ```
  VClo([("x", VInt 36)], "z", Bop(Plus, Var "z", Var "x"))
  ```

  και η τιμή της μεταβλητής **baz** είναι:

  ```
  VClo([("x", VInt 21)], "y", Bop(Mult, Bop(Plus, Var "x", Var "y"),
  Var "x"))
  ```

Η άσκηση σας ζητά να συμπληρώσετε τους ορισμούς του type checker και του interpreter για την γλώσσα που σχεδιάσατε. Σας δίνετε ο σκελετός του αρχείου *σε SML* και σε OCaml. Για τη διευκόλυνση σας, σας δίνονται αρχεία που περιέχουν test cases και ελέγχουν τα προγράμματα σας σε SML και σε OCaml.

Παραδοτέα: το αρχείο **MLin.sml** (SML) ή το αρχείο **MLin.ml** (OCaml) συμπληρωμένα κατάλληλα. Δεν μπορείτε να αλλάξετε τους ορισμούς των τύπων και τους τύπους των συναρτήσεων που υπάρχουν σε αυτά τα αρχεία ή να τα μετονομάσετε. Μπορείτε, αν σας εξυπηρετεί, να ορίσετε βοηθητικές συναρτήσεις.

Περαιτέρω οδηγίες για τις ασκήσεις

- Μπορείτε να δουλέψετε σε ομάδες το πολύ δύο ατόμων, τόσο σε αυτή όσο και στις επόμενες σειρές ασκήσεων. Όμως, έχετε υπ' όψη σας ότι, αν δεν περάσετε το μάθημα φέτος, οι βαθμοί των προγραμματιστικών ασκήσεων κρατούνται μόνο για όσους δεν τις έκαναν σε ομάδα αλλά τις έκαναν μόνοι τους. (Για όλους τους υπόλοιπους, οι βαθμοί των ασκήσεων κρατούνται.)

- Δεν επιτρέπεται να μοιράζεστε τα προγράμματά σας με συμφοιτητές εκτός της ομάδας σας ή να τα βάλετε σε μέρος που άλλοι μπορούν να τα βρουν (π.χ. σε κάποια σελίδα στο διαδίκτυο, σε ιστοσελίδες συζητήσεων, …). Σε περίπτωση που παρατηρηθούν «περίεργες» ομοιότητες σε προγράμματα, ο βαθμός των εμπλεκόμενων φοιτητών σε *όλες τις σειρές ασκήσεων* γίνεται αυτόματα μηδέν ανεξάρτητα από το ποια ομάδα... «εμπνεύστηκε» από την άλλη.

- Μπορείτε να χρησιμοποιήσετε «βοηθητικό» κώδικα (π.χ. κώδικα ταξινόμησης, κάποιο κώδικα που διαχειρίζεται κάποια δομή δεδομένων) που βρήκατε στο διαδίκτυο στα προγράμματά σας, με την προϋπόθεση ότι το πρόγραμμά σας περιέχει σε σχόλια την παραδοχή για την προέλευση αυτού του κώδικα και ένα σύνδεσμο (link) σε αυτόν.

- Τα προγράμματα σε C/C++ πρέπει να είναι σε ένα αρχείο και να μπορούν να μεταγλωττιστούν χωρίς warnings με gcc/g++ (version ≥ 12.2.0) με εντολές της μορφής, π.χ.

```
gcc –std=c99   -Wall –Werror -O3 -lm –o rain rain.c
g++ –std=c++11 -Wall –Werror -O3 -lm –o rain rain.cpp
```

- Τα προγράμματα σε ML πρέπει επίσης να είναι σε ένα αρχείο και να δουλεύουν σε SML/NJ ≥ v110.79 ή σε MLton ≥ 20210117 (στην 1η και 2η άσκηση) ή σε Objective Caml version ≥ 4.13.1. Το σύστημα ηλεκτρονικής υποβολής σας επιτρέπει να επιλέξετε μεταξύ αυτών των διαλέκτων της ML.

- Η υποβολή των προγραμμάτων θα γίνει ηλεκτρονικά μέσω του helios και για να μπορέσετε να τα υποβάλλετε και να βαθμολογηθείτε για αυτά, τα μέλη της ομάδας σας (και οι δύο) θα πρέπει να έχετε εγγραφεί στο μάθημα στο helios. Θα υπάρξει σχετική ανακοίνωση για την ακριβή διαδικασία υποβολής όταν ανοίξει το σύστημα. Τα προγράμματά σας πρέπει να διαβάζουν την είσοδο όπως αναφέρεται και δεν πρέπει να έχουν κάποιου άλλους είδους έξοδο διότι δεν θα γίνουν δεκτά από το σύστημα στο οποίο θα υποβληθούν.