שאלה 1

בתרגיל זה נמשיך במשימת כתיבת מפרש לשפת Lisp בשפת ML. התרגיל עוסק בייצוג של המבנה "סביבה". סביבה היא אוסף של כריכות (bindings) בין שמות לערכים בתוכנית. נממש סביבה כמילון שמקשר בין שם ל-S-Expression. בנוסף נממש מחסנית של סביבות כדי לאפשר קינון של סביבות ותמיכה בהגדרות פונקציות עם פרמטרים.

ייצוג הסביבה ב-ML

סביבה בודדת תהיה מיוצגת ע"י פונקציה שמקבלת שם מטיפוס string ומחזירה את הערך המקושר לאותו שם. סביבה של תוכנית Lisp היא מחסנית (list) של פונקציות שכל אחת מהן מייצגת סביבה. חיפוש כריכה בסביבת התוכנית משמעותו לעבור על כל הסביבות במחסנית ולחפש את הכריכה.

1. הוסיפו את הגדרת הטיפוס של SExpression שראיתם בתרגיל בית 3, ושתי חריגות Empty ו-Undefined, לראש של קובץ הפתרון שלכם:

```
datatype Atom = NIL | SYMBOL of string;
datatype SExp = ATOM of Atom | CONS of (SExp * SExp);
exception Empty;
exception Undefined;
```

2. ממשו את הפונקציה:

```
val initEnv = fn : unit -> string -> SExp
```

פונקציה זו מחזירה פונקציה שמייצגת סביבה ריקה (אין בה bindings). כל קריאה לפונקציה המוחזרת עם מחרוזת כלשהי תזרוק את החריגה Undefined.

3. ממשו את הפונקציה:

```
val define = fn : string -> (string -> 'a) -> 'a -> string -> 'a
```

הפונקציה מקבלת בצורת curried את הפרמטרים הבאים:

- מחרוזת המייצגת את שם המקושר לכריכה.
- פונקציה המייצגת את הסביבה הישנה.
- הערך שמקושר בכריכה.

והיא תחזיר פונקציה שמייצגת את הסביבה לאחר הגדרת הכריכה.

הערה: במידה וקיימות מספר הפעלות של define עבור אותו השם המקושר לכריכה, על ההפעלה האחרונה ביותר להיות היחידה שמשפיעה (כלומר עליה להסתיר את הקישורים של ההפעלות הקודמות עבור שם זה).

<u>דוגמאות הרצה:</u>

```
- val env : (string -> SExp) = initEnv();
val env = fn : string -> SExp
- val env = define "x" env (ATOM (SYMBOL "5"));
val env = fn : string -> SExp
- env "x";
val it = ATOM (SYMBOL "5") : SExp
- env "y";
uncaught exception Undefined
- val env = define "y" env (ATOM (SYMBOL "7"));
val env = fn : string -> SExp
- env "x";
val it = ATOM (SYMBOL "5") : SExp
- env "y";
val it = ATOM (SYMBOL "7") : SExp
- env "z";
uncaught exception Undefined
```

4. ממשו את הפונקציה:

```
val emptyNestedEnv = fn : unit -> (string -> SExp) list
```

פונקציה זו מחזירה רשימה שמכילה את הסביבה הריקה בלבד. מומלץ להוסיף type constraints למימוש.

5. ממשו את **הפונקציות** הבאות:

```
val pushEnv = fn : 'a -> 'a list -> 'a list

val popEnv = fn : 'a list -> 'a list

val topEnv = fn : 'a list -> 'a
```

הפונקציות האלה מבצעות פעולות פשוטות על מחסנית הסביבות.

- **pushEnv** : הוספת סביבה כלשהי לראש המחסנית.
- **popEnv** : הסרת הסביבה שנמצאת בראש המחסנית.
- **topEnv** : תחזיר את הסביבה שבראש המחסנית.

שימו לב שאלו הפעולות הסטנדרטיות שאתם מכירים והטיפוס הוא רשימה גנרית.

הפונקציות **topEnv** ו-**popEnv** יזרקו את החריגה **Empty** במקרה שהמחסנית ריקה.

6. ממשו את הפונקציה:

```
val defineNested =
 fn : string -> (string -> 'a) list -> 'a -> (string -> 'a) list
```

הפונקציה מגדירה כריכה חדשה על הסביבה שבראש המחסנית.

הפרמטרים:

- מחרוזת המייצגת את שם המקושר לכריכה.
- מחסנית סביבות.
- הערך שמקושר בכריכה.

כלומר, הפונקציה משנה את הערך עבור הכריכה שבראש המחסנית להיות הערך שבפרמטר מספר 3, אך לא משנה את אף אחת מהכריכות האחרות.
במידה ומחסנית הסביבות היא ריקה צריכה להיזרק השגיאה Empty.

7. ממשו את הפונקציה:

```
val find = fn : string -> (string -> 'a) list -> 'a
```

פונקציה זו מקבלת שם ומחסנית סביבות ומחזירה את הערך המקושר לשם זה (בסביבה החיצונית ביותר).
אם מחסנית הסביבות שהתקבלה היא ריקה, עליכם לזרוק את החריגה Undefined.
בנוסף, אם לשם שהתקבל בקלט אין אף ערך מקושר בסביבה, עליכם לזרוק את החריגה Undefined.

כעת שימו לב, ייתכן ותיתקלו בחריגות שזרקתם בפונקציות מהסעיפים הקודמים.
על מנת לתפוס חריגה בשפת ML עליכם להשתמש במילה השמורה handle כפי שלמדתם.

<u>דוגמאות הרצה:</u>

```
- val env : (string -> SExp) list = emptyNestedEnv ();
val env = [fn] : (string -> SExp) list

- val env = defineNested "message" env (ATOM (SYMBOL "Hello, World"));
val env = [fn] : (string -> SExp) list

- find "message" env;
val it = ATOM (SYMBOL "Hello, World") : SExp

- val env = pushEnv (initEnv ()) env;
val env = [fn,fn] : (string -> SExp) list

- find "message" env;
val it = ATOM (SYMBOL "Hello, World") : SExp

- val env = defineNested "message" env (ATOM (SYMBOL "NOPE"));
val env = [fn,fn] : (string -> SExp) list

- find "message" env;
val it = ATOM (SYMBOL "NOPE") : SExp

- val env = popEnv env;
val env = [fn] : (string -> SExp) list

- find "message" env;
val it = ATOM (SYMBOL "Hello, World") : SExp
```

שאלה 2

הגענו לחלקו האחרון של מימוש המפרש של שפת Lisp. בחלק זה נרצה לממש את הפונקציה eval כך שהיא שתקבל סביבה וS-Expression ותחזיר לנו:

- את השערוך של הביטוי תחת הסביבה.
- את הסביבה החדשה לאחר פעולת השערוך.

בעזרת הפונקציה הזו יחד עם הפונקציה parse שמימשנו בתרגילי בית הקודמים ניתן כעת לממש את הפונקציה REPL ולקבל מפרש אינטראקטיבי מלא לשפת Lisp. אנחנו נסתפק רק במימוש תקין של פונקציית eval ולא נממש את הפונקציה REPL.

1. בראש קובץ הפתרון של התרגיל, הוסיפו

```
use "hw4_q1.sml"
```

כדי להביא את הגדרת הטיפוסים ואת המימוש של תרגיל 1.

הפונקציה eval

2. הגדירו את החריגה הבאה בתוכנית שלכם:

```
exception LispError;
```

חריגה זו תשמש אתכם בכל סוגי השגיאות של **eval**. אסור לכם לזרוק כל חריגה אחרת מ-eval.

חריגה זו תזרק מהמימוש הפנימי של eval (פונקציית עזר, מכוון ש-eval עצמה תקבל מחרוזת ותדאג לחישוב SExp מתאים ע"י **parse (tokenize str)**) לכן כש-eval תתקל בחריגה זו, היא תחזיר symbol שמכיל "lisp-error". ראו דוגמאות בהמשך.

3. הוסיפו את הפרסר שלכם לקוד ע"י שימוש בפקודה **use**. על הפקודה להופיע מעל להכרזת החריגה למעלה.

לנוחיותכם, סיפקנו עבורכם hw4_q2.sml שמכיל מבנה כללי וכמה פונקציות שימושיות.

הגדירו את הפונקציה eval שמקבלת ביטוי מטיפוס SExp וסביבה מקוננת (כפי שהגדרנו קודם) והיא מחזירה טאפל של שיערוך הביטוי מעל הסביבה ואת הסביבה המעודכנת.
החתימה שלה:

```
val eval = fn : string -> (string -> SExp) list -> SExp *
(string -> SExp) list
```

אופן פעולת הפונקציה מתואר בהמשך.

שיערוך ביטויים בשפת Lisp

כעת נגדיר כיצד על הפונקציה eval לשערך ביטויים מטיפוס S-Expression בשפת Lisp.

1. אם הביטוי הוא אטום אזי:
   a. אם הוא NIL או **שהוא מייצג מספר טבעי** אז השערוך שלו יהיה האטום עצמו.
   b. אם הוא symbol אז השערוך שלו יהיה השערוך של הערך המתאים לו בסביבה (בהמשך נגדיר כיצד להגדיר כריכות בסביבה).

2. אם הוא רשימה אזי זו הפעלה של פונקציה:
   a. האיבר הראשון ברשימה הוא שם הפונקציה.
      i. **(בונוס)** אם הפונקציה הוגדרה ע"י קריאות רקורסיביות בעזרת (label) יש לשלוף אותה מהסביבה.
      ii. אם הפונקציה הוגדרה ע"י מתכנני השפה (אתם) אין צורך לשמור אותה בסביבה ואתם יכולים להכניסה כחלק ממימוש הפונקציה eval. בהמשך נגדיר את הפונקציות שמוגדרות מראש בשפה.
   b. האיברים הבאים ברשימה הם הפרמטרים האקטואליים.
      i. **(בונוס)** במקרה של a.i - לפני הפעלת הפונקציה יש להכניס כריכות בין הפרמטרים האקטואליים לבין הפרמטרים הפורמליים בscope חדש כלומר יש להכניס סביבה חדשה למחסנית הסביבות.
   c. לאחר מכן יש לשערך את הביטוי המגדיר את הפונקציה על מחסנית הסביבות המעודכנת.
      i. **(בונוס)** במקרה של a.i - בסוף יש להוציא את הסביבה החדשה ממחסנית הסביבות, כלומר, להסיר את הכריכות בין הפרמטרים הפורמליים לפרמטרים האקטואליים של הפונקציה.

3. כל דבר אחר הוא שגיאה ועליכם לזרוק עליו LispError.

הפונקציות המוגדרות מראש בLisp

1. הפונקציה **cons**: יוצרת dotted pair. מקבל שני פרמטרים, הראשון הוא הראש, השני הוא הזנב.
   <u>דוגמאות:</u>

```
- first (eval "(cons (quote a) 3)" env);
val it = CONS (ATOM (SYMBOL "a"),ATOM (SYMBOL "3")) : SExp
- sexp_to_string[1] it;
val it = "(a . 3)" : string
- first (eval "(cons 1 (cons 2 nil))" env);
val it = CONS (ATOM (SYMBOL "1"),CONS (ATOM (SYMBOL "2"),ATOM
NIL)) : SExp
- sexp_to_string it;
val it = "(1 2)" : string
```

2. הפונקציה **car**: מחזירה איבר ראשון בזוג.
   <u>דוגמאות:</u>

```
- first (eval "(car (cons 2 3))" env);
val it = ATOM (SYMBOL "2") : SExp
- sexp_to_string it;
val it = "2" : string
- first (eval "(car nil)" env);
val it = ATOM (SYMBOL "lisp-error") : SExp
- sexp_to_string it;
val it = "lisp-error" : string
```

3. הפונקציה **cdr**: מחזירה איבר שני בזוג.
   <u>דוגמאות:</u>

```
- first (eval "(cdr (cons 2 3))" env);
val it = ATOM (SYMBOL "3") : SExp
- sexp_to_string it;
val it = "3" : string
```

4. הפונקציה **quote**: מקבלת פרמטר יחיד ומחזירה אותו בלי שערוך (מה זאת אומרת על שאר הפונקציות?).

---

[1] הפונקציה sexp_to_string מסופקת בקובץ הטסטים.

דוגמאות:

```
- first (eval "(quote lol)" env);
val it = ATOM (SYMBOL "lol") : SExp
- first (eval "(quote (1 2 3))" env);
val it =
  CONS
    (ATOM (SYMBOL "1"),
     CONS (ATOM (SYMBOL "2"),CONS (ATOM (SYMBOL "3"),ATOM NIL)))
: SExp
- sexp_to_string it;
val it = "(1 2 3)" : string
- sexp_to_string (first (eval "(quote (cons 2 3))" env));
val it = "(cons 2 3)" : string
```

5. הפונקציה **atom**: מקבלת פרמטר יחיד ומחזירה t[2] אם הוא אטום, אחרת nil.

דוגמאות:

```
- first (eval "(atom 2)" env);
val it = ATOM (SYMBOL "t") : SExp
- first (eval "(atom (cons 2 3))" env);
val it = ATOM NIL : SExp
```

6. הפונקציה **null**: מקבלת רשימה ומחזירה t אם היא ריקה. אחרת היא תחזיר nil.

דוגמאות:

```
- first (eval "(null (quote ()))" env);
val it = ATOM (SYMBOL "t") : SExp
- first (eval "(null (quote 2))" env);
val it = ATOM NIL : SExp
```

7. הפונקציה **eq**: מקבלת שני פרמטרים ומחזירה t אם שניהם אטומים זהים, אחרת nil.

דוגמאות:

```
- first (eval "(eq 2 2)" env);
val it = ATOM (SYMBOL "t") : SExp
- first (eval "(eq 2 3)" env);
val it = ATOM NIL : SExp
- first (eval "(eq (cons 1 2) (cons 1 2))" env);
```

[2] כל ערך חזרה שהוא לא nil תופס, אך לפשטות ביקשנו t

```
val it = ATOM NIL : SExp
```

8. הפונקציה **cond**: מקבלת רשימה של (condition exp), ומחזירה את ה-exp השייך לזוג הראשון שעבורו condition הוא **לא nil**. במקרה שאין זוג כזה, יש להחזיר nil.
<u>דוגמאות:</u>

```
- first (eval "(cond ((eq 2 2) 5))" env);
val it = ATOM (SYMBOL "5") : SExp
- first (eval "(cond ((eq 2 4) 3) ((eq 2 3) 4))" env);
val it = ATOM NIL : SExp
- first (eval "(cond ((eq 2 3) 3) ((eq 3 4) 3) ((quote
am-i-nil?) 4))" env);
val it = ATOM (SYMBOL "4") : SExp
```

9. פונקציות מתמטיות **+, -, *, /, mod**: כל פונקציה מקבלת שני פרמטרים של סימבולים מספריים **(ניתן להניח תקינות)** ומחזירה את הפעלת האופרטור המתמטי המתאים עליהם. **שימו לב שניתן להגדיר פונקציית עזר (mutually recursive) שתטפל במקרים ותחסוך בכתיבת קוד.**
<u>דוגמאות:</u>

```
- first (eval "(+ 1 2)" env);
val it = ATOM (SYMBOL "3") : SExp
- first (eval "(- 1 2)" env);
val it = ATOM (SYMBOL "~1") : SExp
- first (eval "(* 2 3)" env);
val it = ATOM (SYMBOL "6") : SExp
- first (eval "(/ 6 2)" env);
val it = ATOM (SYMBOL "3") : SExp
- first (eval "(mod 5 2)" env);
val it = ATOM (SYMBOL "1") : SExp
```

10. פונקציות השוואות **=, =/, >, <**: כל פונקציה מקבלת שני פרמטרים של סימבולים מספריים **(ניתן להניח תקינות)** ומחזירה את הפעלת האופרטור המתאים עליהם. **שימו לב שניתן להגדיר פונקציית עזר (mutually recursive) שתטפל במקרים ותחסוך בכתיבת קוד.**
<u>דוגמאות:</u>

```
- first (eval "(= 1 1)" env);
val it = ATOM (SYMBOL "t") : SExp
- first (eval "(= 1 2)" env);
```

```
val it = ATOM (SYMBOL "nil") : SExp
- first (eval "(/= 1 2)" env);
val it = ATOM (SYMBOL "t") : SExp
- first (eval "(/= 1 1)" env);
val it = ATOM (SYMBOL "nil") : SExp
- first (eval "(< 1 2)" env);
val it = ATOM (SYMBOL "t") : SExp
- first (eval "(< 2 1)" env);
val it = ATOM (SYMBOL "nil") : SExp
- first (eval "(> 2 1)" env);
val it = ATOM (SYMBOL "t") : SExp
- first (eval "(> 1 2)" env);
val it = ATOM (SYMBOL "nil") : SExp
```

11. הפונקציה **lambda**: הפונקציה משמשת להגדרת פונקציות אנונימיות (ללא שם). היא מקבלת רשימה של פרמטרים פורמליים וביטוי גוף, ומחזירה פונקציה שיכולה להיות מופעלת על ידי נתינת ערכים לפרמטרים הפורמליים.

כלומר בעת הפעלת הפונקציה האנונימית, הערכים צריכים להיות מקושרים לשמות של הפרמטרים הפורמליים בסביבת ההפעלה.
בנוסף, יש לדאוג לכך שהקישורים לא ישפיעו על הסביבה המוחזרת מ-eval אחרי הביצוע הסופי (ראו דוגמה).
<u>דוגמאות:</u>

```
- first (eval "((lambda (x) x) 2)" env);
val it = ATOM (SYMBOL "2") : SExp
- first (eval "((lambda (x y) (cons x y)) 2 3)" env);
val it = CONS (ATOM (SYMBOL "2"),ATOM (SYMBOL "3")) : SExp
- first (eval "((lambda (x y) (cons x y)) 2 (quote 3))" env);
val it = CONS (ATOM (SYMBOL "2"),ATOM (SYMBOL "3")) : SExp
- first (eval "((lambda (x y) (+ x y)) 2 3)" env);
val it = ATOM (SYMBOL "5") : SExp
- first (eval "((lambda (x y) (+ x y)) 2 3 4)" env);
val it = ATOM (SYMBOL "lisp-error") : SExp
- first (eval "((lambda (x) ((lambda (y) (cons x y)) (quote b)))
(quote a))" env);
val it = CONS (ATOM (SYMBOL "a"),ATOM (SYMBOL "b")) : SExp
```

<u>הרכבה + בדיקת אי שינוי סביבה:</u>

```
- val (sexp, new_env) = (eval "((lambda (x) \
```

```
                                  \((lambda (y) \
                                   \(+ x y)) \
                                    \3)) \
                                     \2)" env);
val sexp = ATOM (SYMBOL "5") : SExp
val new_env = [fn] : (string -> SExp) list

- find "x" new_env;

uncaught exception Undefined
```

12. **בונוס** הפונקציה label: הפונקציה משמשת להגדרת פונקציות רקורסיביות (עם שם). היא מקבלת רשימה של שם וביטוי lambda, ומחזירה פונקציה שיכולה להיות מופעלת על ידי נתינת ערכים לפרמטרים הפורמליים.

כלומר בעת הפעלת הפונקציה, השם צריך להיות מקושר לביטוי ה-lambda ובשערוך קריאה לפונקציה עם ערכים, הערכים צריכים להיות מקושרים לשמות של הפרמטרים הפורמליים בסביבת ההפעלה.
בנוסף, יש לדאוג לכך שהקישורים לא ישפיעו על הסביבה המוחזרת מ-eval אחרי הביצוע הסופי.
<u>דוגמאות:</u>

```
- sexp_to_string (first (eval "\
     \((label fact \
          \(lambda (n) \
            \(cond ((eq n 0) 1) \
                   \(t (* n (fact (- n 1))))))) \
      \5)" env)) = "120";
val it = true : bool
- sexp_to_string (first (eval "((label sum-list \
                                \(lambda (lst) \
                                 \(cond ((null lst) 0) \
                                     \(t (+ (car lst) (sum-list (cdr lst))))))) \
                               \(quote (1 2 3 4 5)))" env)) = "15";
val it = true : bool
```

```
- sexp_to_string (first (eval "((label sum-to-n \
                                  \(lambda (n acc) \
                                   \(cond ((eq n 0) acc) \
                                       \(t (sum-to-n (- n 1) (+ acc n)))))) \
                                 \10 0)" env)) = "55";
val it = true : bool
- sexp_to_string (first (eval "((label fact \
                                \(lambda (n) \
                                   \(cond ((eq n 0) 1) \
                                      \(t (* n (fact (- n 1))))))) \
                                 \((lambda (x) \
                                 \((lambda (y) \
                                   \(cond \
                                      \((eq y 1) 1) \
                                      \((eq y 2) 2) \
                                      \(t 3))) \
                                 \x)) \
                               \2))" env)) = "2";
val it = true : bool
```