עקרונות שפות תכנות - תרגיל 3
Sequence Operations & ADTs
Interpreter for Substitution Model

תאריך הגשה: 22.6

תשובות מילוליות יש לכתוב כהערות בקובץ ה scheme שאתם מגישים

חלק 1 – 25% Sequence Operations & ADTs

בשאלות בחלק זה הניחו כי הקלטים תקינים. לכל פו' כתבו בהערה טיפוס ותנאי קדם, אלא אם כן כתוב אחרת בסעיף.

ענו על **אחת** מהשאלות הבאות לבחירתכם

שאלה 1 - 25 נק'

הפונקציה fold היא sequence operation ומוגדרת כך:

```
(define (fold f initial lst)
  (if (null? lst)
      initial
      (fold f
            (f initial (car lst)) (cdr lst))))
```

א. כתבו את הטיפוס של הפונקציה fold

ב. כתבו פו' בשם $append\_lis$ המקבלת כקלט רשימה לא ריקה של רשימות לא-מקוננות ולא ריקות של מספרים, **ומשתמשת ב fold** כדי להחזיר רשימה אחת לא מקוננת, המכילה שרשור של כל המספרים.
לדוגמא ((1 2) (3) (4 5 6))' $append\_lis$) תחזיר (1 2 3 4 5 6)

ג. **בעזרת הפונקציה מהסעיף הקודם, ובעזרת map** ממשו פו' בשם $flatten$ המקבלת כקלט רשימת מספרים מקוננת, בעלת קינון שאינו מוגבל (כל איבר יכול להיות מקונן בדרגת קינון כלשהי או לא מקונן), ומחזירה רשימה אחת ללא קינון המכילה את כל איברי הרשימה המקוננת.
למשל ((1 3) (((4) 5) 6 ((7)) 8)' $flatten$) תחזיר (1 2 3 4 5 6 7 8)

נגדיר עץ בעזרת הכללים הבאים:

(1) איבר בודד (שאינו רשימה) הוא עץ
(2) רשימה שבה לפחות שני איברים – האיבר הראשון חייב להיות איבר בודד (לא רשימה), ושאר האיברים יקראו **בנים** של העץ, כאשר כל אחד מהם הוא עץ

להלן מספר דוגמאות לעצים:

5
"abc"
(3 4)

בן יחיד: 4 שהוא איבר בודד כאשר 3 אינו בן

((4 4) 7 (#t (4 8 9 #t))

שלוש הבנים הם (4 4) ,7 ,(#t 9 8 4) כאשר רק הבן השני הוא איבר בודד, כאשר #t אינו בן

להלן תזכורת לפונקציה filter המשאירה ברשימה רק איברים המקיימים את התנאי pred

```
(define (filter pred li)
  (cond ((null? li) li)
        ((pred (car li))
         (cons (car li)
               (filter pred (cdr li))))
        (else
         (filter pred (cdr li)))))
```

ד. ממשו את ה ADT הבא עבור עצים כפי שהוגדרו (לא לשכוח טיפוס ותנאי-קדם):

פרדיקטים : tree?, has_children? האחד לזיהוי עץ והשני לבדיקה אם לעץ נתון יש בנים. שימו לב שהבדיקה tree? צריכה לבדוק את כל מה שפורט לעיל (למשל שכל הילדים הם עצים). אין להשתמש בתגים. מומלץ להשתמש ב filter,map אך לא חובה

שלפנים:

(1) get-datum המקבל עץ ומחזיר את התוכן, שהוא העץ עצמו במקרה של איבר בודד, או הערך **שאינו** בן בעץ (כאשר יש בנים). למשל get-datum עבור העץ בדוגמא הראשונה יחזיר 5, ובדוגמא האחרונה יחזיר #t

(2) get-children תקבל עץ ותחזיר את בניו. אם אין בנים, נחזיר null

בנאי: אין צורך לממש. הניחו כי העצים נתונים כרשימות מקוננות.

ה. **בעזרת הממשק** ממשו **בצד הלקוח** את הפונקציות הבאות (עם טיפוס ותנאי קדם):

- get-all-data תקבל עץ ותחזיר את כל הנתונים שבעץ כרשימה שטוחה. למשל עבור העץ ((4 4) 7 (#t (4 8 9 #t)) נחזיר את הרשימה (4 4 7 #t 9 8 4 #t). שימו לב לדמיון לפונקציה flatten, ונסו לכתוב פונקציה קרובה ככל האפשר לפונקציה הקודמת.
- get-max-data תקבל עץ שיתכן שמכיל מספרים. אם יש מספרים, נחזיר את המספר הגדול המוכל בעץ. אם אין מספרים נחזיר #f. למשל עבור העץ מהדוגמא הקודמת נחזיר 9.

שאלה 2 – 25 נק'

א. ממשו, במימוש לבחירתכם אך **ללא** שימוש באופרטור list, את ה ADT הבא לטיפול באובייקטים מסוג **triple** של מספרים (כתבו טיפוס לכל פונקציה)

- הבנאי (make-triple a b c) יקבל כקלט שלושה איברים כלשהם וישמור אותם
- השלפנים get1, get2, get3 יקבלו כקלט אובייקט שלשה ויחזירו בהתאמה את a, את b או את c
- פרדיקט triple? יקבל אובייקט כלשהו ויחזיר #t אם הוא שלשה

ב. לאחר המימוש, נתון כי לא אי אפשר להשתמש ב cons,car,cdr ויש לממשם עבור אובייקט **pair** חדש ע"י שימוש בפונקציות שכתבתם בסעיף א' - כדי לא ליצור התנגשות שמות, ממשו את new_cons, new_car, new_cdr. הפונקציות יתנהגו בדיוק כמו cons,car,cdr רק שהמימוש חייב להיעשות בצד הספק ע"י שלשה. (למשל בקוד של new_cons תופיע קריאה ל make-triple). אין צורך לכתוב חוזים בסעיף זה.

ג. בעזרת שימוש בקוד שבסעיפים א,ב (ניתן להשתמש בו באחד מהם או בשניהם), אך **ללא** שימוש ב cons (המקורי) או ב list, ממשו ADT עבור אובייקט **point**, הנתון להלן:
(make-point x y) תקבל שני מספרים ממשיים ותשמור אותם. השתמשו בחבילת התגים.
(get-x p), (get-y p) שלפנים להחזרת x,y בהתאמה
(point? Obj) פרדיקט

ד. בעזרת אובייקט הנקודה רוצים לממש ADT עבור אובייקט **line** שאותו ניתן לבנות ע"י 2 בנאים שונים:
(make-line p1 slope) בנאי שיקבל נקודה ושיפוע (מספר ממשי)
(make-line2 p1 p2) בנאי שיקבל שתי נקודות
(get-slope line) שלפן להחזרת שיפוע הקו. תזכורת: שיפוע בעזרת שתי נקודות נתון בנוסחא: $\frac{y2-y1}{x2-x1}$
כתבו בהערה אם חישוב ה slope שמימשתם הוא חרוץ או עצל, והסבירו בקצרה.
(get-p1 line) שלפן להחזרת p1 (שימו לב כי אין צורך לממש שלפן עבור p2)

ה. נתונה רשימה שבה כל איבר יכול להיות זוג, שלשה, נקודה או קו. בעזרת *filter* ממשו פונקציה בשם (only-with-x x li) שתקבל כקלט רשימה כזו ותחזיר רשימה שבה קיימים רק האיברים מהרשימה המקורית המכילים בתוכם את x. הערך x **אינו** מתיחס לשיפוע של קו.

חלק 2- 75% – המשערך המלא

ענו על 3 השאלות הבאות. כשאתם מגישים את התרגיל, רכזו את כל השינויים מהשאלות שמימשתם במימוש יחיד של הפונקציות המתאימות. למשל אם אתם מממשים את or-positive וגם את get-procedure-body הוסיפו את שתי השאילתות המתאימות למימוש יחיד של הפונקציה _eval, שאותו תגישו. כנ"ל לגבי פונקציות אחרות שתשנו (למשל derive)

שאלה 3

שאלה זו עוסקת בביטויי or-positive ובמשערך של מודל ההחלפה. בפתרון השאלה, **פרט לסעיף האחרון**, הניחו כי **לא** קיים מימוש עבור or ב-Scheme (כלומר **אין** להסתמך על מימוש קיים של ביטויי or).

מטרתו של ביטוי or-positive היא לבדוק כי לפחות אחד הארגומנטים שלו בעל ערך חיובי. הביטוי מוגדר כך:

```
(or-positive exp1 exp2  …  expn)
```

ערכו של ביטוי or-positive מתקבל באופן הבא: ראשית, משערכים את הביטוי `exp1`. אם הערך **חיובי**, מחזירים ערך true. אחרת, עוברים לביטוי exp2 – אם ערכו חיובי נחזיר true, אחרת נמשיך בדומה עד לביטוי האחרון. בביטוי האחרון נחזיר true אם הוא חיובי, ו false אם אינו חיובי.

בביטוי תקין קיים לפחות ארגומנט exp אחד (n>=1). אין להשתמש ב or בשערוך הביטוי, פרט לסעיף האחרון.

לדוגמא: שערוך `(or-positive 1 (+ 1 1) 3)` יחזיר **true**, שערוך `(or-positive (- 3 5))` יחזיר false.

א. האם הביטוי or-positive חייב להיות special-form או שניתן לממשו ע"י פרוצדורה? הסבירו.

ב. ממשו את הפרוצדורות הבאות (**חובה** להשתמש בחבילת התגים):

- הבנאי `make-or-positive`, המקבל רשימת ביטויים כארגומנט
- שלפנים (selectors):

- `get-expressions` הפועל על ביטוי or-positive שלם
- `first-exp, second-exp` ו- `rest-exps` הפועלים על exps

- והפרדיקטים `or-positive?` ו-`last-exp?`.
כתבו גם את **הטיפוס** של כל אחת מהם בצד **הממש**.

ג. כתבו את **כל** השינויים שצריך לבצע בקוד המשערך כדי לתמוך בביטויי or-positive **כביטויי גרעין** (שוב, הניחו כי **אין** בשפה תמיכה ב-`or`) **מבלי** לממש עדיין את `eval-or-positive` .

ד. ממשו את `eval-or-positive`.

ה. עתה, הניחו כי **קיימים** ביטויי or בשפה. כתבו את ביטוי ה-or שאליו תגזרו את הביטוי (or-positive 1 (+1 1) 3) וכתבו **בעזרת map** פרוצדורה לביצוע גזירה סינטקטית של ביטויי or-positive לביטויי or. חתימת הפרוצדורה צריכה להיות

```
(or-positive->or <exp>)
```

שאלה 4

רוצים לאפשר למשתמש לבחון את הגוף ואת רשימת המשתנים של פרוצדורה. לשם יש להגדיר שני ביטויים חדשים מהצורה

```
(get-procedure-body <exp>)
```

ביטוי זה מקבל כארגומנט ביטוי יחיד, משערך אותו, ובמידה והתקבל ערך מסוג פרוצדורת משתמש, מחזיר את הגוף שלה.

הביטוי הבא יחזיר את רשימת הפרמטרים של הפרוצדורה

```
(get-procedure-params <exp>)
```

להלן המקרים האפשריים לאחר השיערוך של <exp> :

- אם הערך המוחזר הוא פרוצדורת משתמש – יש להחזיר את ה body שלה (או את רשימת הפרמטרים)
- אם הערך המוחזר הוא פרוצדורה פרימיטיבית – יש להדפיס "primitive! Implementation hidden"
- אם הערך אינו פרוצדורה, יש להדפיס "error: non-procedure"

```
(define f (lambda (x y) (* x y)))
(get-procedure-body  f)
⇨  (* x y)

(get-procedure-params  f)
⇨  (x y)

(get-procedure-body (lambda (x) (h x)) )
⇨  (h x)

(get-procedure-body +)
```

```
⇨ "primitive! Implementation hidden"

(get-procedure-body 1)
⇨ "error: non-procedure"

(get-procedure-body (+ 1 1))
⇨ "error: non-procedure"
```

א. כתבו ADT עבור ביטויי get-procedure-body, get-procedure-params שימו לב כי זהו אינו ה ADT עבור closures אם כי הוא צריך לפעול בדומה

ב. הוסיפו תמיכה בביטויי get-procedure-params, get-procedure-body כביטויי **גרעין** (כתבו את <u>כל</u> השינויים הנדרשים).

ג. האם הביטוי **חייב** להיות ביטוי גרעין או שניתן לממש אותו כביטוי נגזר? הסבירו

שאלה 5

רוצים להוסיף לשפה הגדרת **פרוצדורות בשני משתנים** ע"י ביטוי בשם define-f2. ביטוי זה הוא מהצורה:

```
(define-f2  <f-name> <var1> <var2> <body>)
```

ביטוי זה מתנהג בדומה לביטוי **define**, אך מיועד להגדרת פרוצדורות עם שני משתנים בלבד. שמות המשתנים יגיעו אחרי שם הפרוצדורה, ללא סוגריים, ואחריהם יגיע ה body.

להלן מספר דוגמאות:

```
(define-f2  g  x  y (+ x y 1))
g
=> #<procedure>
(g 2 3)
=> 6

(define-f2  h a b (* a a b))
(h 2 5)
=> 20
```

א. ממשו לפי כותרות הפרומצדורות שלהלן את ה-ADT לביטויי **define-f2** ב-ASP. הפרוצדורה תיקרא f-name, הפרמטרים יקראו var1, var2 והגוף ייקרא body.

```
(make-define-f2 f-name var1 var2 body)

(define-f2? exp)

(get-fname define-f2)

(get-var1 define-f2)

(get-var2 define-f2)
```

```
(get-body define-f2)
```

ב. השלימו את כל הנדרש כדי לתמוך בביטויי define-f2 כביטויי גרעין

ג. נניח שרוצים להשתמש בגזירה במקום בביטוי גרעין. כתבו **שני** ביטויים שונים אליהם אפשר לגזור את הביטוי:

```
(define-f2  g  x  y
           (if (> x y) x (g (+ x 1) (- y 1))))
```

ד. ממשו את פרוצדורת הגזירה **define-f2->define** הגוזרת את ביטוי ה **define-f2** המתקבל בקלט, לביטוי **define** שקול

בהצלחה