תאריך הבוחן: 19.5.2017 שעה: 9:00
שם המורה: פרופ' מיכאל אלחדד, פרופ' מירה בלבן, ד"ר דנה פיסמן, ד"ר מני אדלר, ד"ר ירון גונן.
בוחן ב: עקרונות שפות תכנות
מס. הקורס: 202-1-2051
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה, מתמטיקה ומדעי המחשב
שנה: ב' סמסטר: ב'
משך הבוחן: 2 שעות

שאלה 1 Applicative/normal operational semantics

נתונה התוכנית הבאה בשפת **Scheme**.

```
(define y 7)
(define double (lambda (x) (display x) (newline) (* x 2)))
(define g (lambda (x) (if (= y (double y)) 'eq 'neq)))
(g (double 1))
```

א. [**6** נק']

מה יודפס ומה יהיה הערך שיוחזר בשערוך התוכנית ע"י applicative-eval?

_____
_____
_____
_____

ב. [**6** נק']

מה יודפס ומה יהיה הערך שיוחזר בשערוך התוכנית ע"י normal-eval?

_____
_____
_____
_____

שאלה 2 – Lexical Address

נתון הביטוי הבא בשפת **Scheme**.

```
((lambda (x y)
    (display x)
    (newline)
    (display y)
    (newline)
    ((lambda (x z)
        (newline)
        (let ((y 3) (z x))
              (if (> y x)
                  (display y)
                  (display z))))
     x y)) 1 9)
```

א. [**4** נק'] לכל מופע של אחד מתתי הביטויים בקבוצה **{x,y,z}** רשום/רשמי את סוגו (var-ref, var-decl).

```
((lambda (x_____ y_____)
  (display x_____)
  (newline)
  (display y_____)
  (newline)
  ((lambda (x_____ z_____)
      (newline)
      (let ((y_____ 3) (z_____ x_____))
            (if (> y_____ x_____)
                (display y_____)
                (display z_____))))
   x_____ y_____)) 1 9)
```

ב. [**6** נק'] לכל מופע של אחד מתתי הביטויים בקבוצה **{x,y,z}** מסוג var-ref ציין/י את ה lexical-address שלו. (למופעים מסוגים אחרים רשום "non var-ref").

להזכירך המבנה של **lexical address** הינו **[x : m n]** כאשר **m** הוא המרחק ל **scope** בו מוגדר המשתנה **x** ו **n** הוא המיקום של **x** ברשימת הפרמטרים.

```
((lambda (x_____ y_____)
  (display x_____)
  (newline)
  (display y_____)
  (newline)
  ((lambda (x_____ z_____)
      (newline)
      (let ((y_____ 3) (z_____ x_____))
            (if (> y_____ x_____)
                (display y_____)
                (display z_____))))
   x_____ y_____)) 1 9)
```

ג. [**6** נק'] בביטוי הבא בשפת Scheme מצויינים **ה** lexical addresses של המשתנים המופיעים **כ**var-decl , אבל שמות המשתנים מחוקים.

הוסף/הוסיפי לביטוי שמות משתנים שיתאימו ל lexical-addresses המצויינים.

```
> (define foo
    (lambda (__ __)
        (- (+ [__ : 0 0]
              ((lambda (__) (* [__ : 0 0] [__ : 1 0]))
               [__ : 0 1]))
        [__ : 0 1])))
> (foo 5 2)
13
> (foo 10 7)
73
```

שאלה 3 [6 נק'] – Abstract Syntax Tree

כתוב/כתבי את הביטוי שיתקבל ע"י הפעלת ה Parser על הביטוי הבא :

```
((lambda (x) (if (> x 0) x 'neg)) 5)
```

_____

_____

_____

שאלה 4 – Substitution and Renaming

נתון הביטוי הבא בשפת **Scheme**.

```
(define bar (lambda (x)
  (display x)
  (newline)
  ((lambda (x)
      (display (list x y))
      (newline)
      (let ((x 1) (y 2))
            (if (> y x)
                (display y)
                (display x))))
   x)))
(bar 5)
```

א. [**4** נק'] בצע **Renaming** לביטוי כך שלכל המשתנים המופיעים כ**var-decl** יהיה שם שונה. רשום/רשמי את הביטוי המתקבל:

```
(define bar (lambda (___)
  (display ___)
  (newline)
  ((lambda (___)
      (display (list ___ ___))
      (newline)
      (let ((___ 1) (___ 2))
            (if (> ___ ___)
                (display ___)
                (display ___))))
    ___)))
(bar 5)
```

ב. [**4** נק'] בצע את ה **Substitution** המתבקש לאור שערוך הביטוי **(bar 5)**.
רשום/רשמי את הביטוי המתקבל:

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

<u>שאלה 5 – Functional Abstractions</u>

א. [**6** נק'] הפונקציה **high-some(pred)** מחזירה פונקציה שמקבלת רשימה ומחזירה את הערך הבוליאני **true** אם לפחות אחד האיברים ברשימה מקיים את הפרדיקט **pred**.
לדוגמה

```
> high-some(even)([1,2,3])
true
> high-some(prime)([10,20,30])
false
```

--- כתוב/כתבי מימוש לפונקציה **high-some** בשפת התכנות **<u>Typescript</u>**.

--- השתמש/י ב **functional abstractions** שלמדנו בכיתה **map, filter, reduce**.
--- ציין/י את ה **full-type** של הפונקצייה.

function high-some (

_____
_____
_____
_____
_____
_____
_____

ב. [**6** נק'] הפונקציה **high-every(pred)** מחזירה פונקציה שמקבלת רשימה ומחזירה את הערך הבוליאני **true** אם כל אחד מהאיברים ברשימה מקיים את הפרדיקט **pred**.
לדוגמה

```
> high-every(even)([1,2,3])
false
> high-every(prime)([2,3,5])
true
```

--- כתוב/כתבי מימוש לפונקציה **high-some** בשפת התכנות **Scheme**.
--- השתמש/י ב **functional abstractions** שלמדנו בכיתה **map, filter, foldr**.
--- ציין/י את ה **full-type** של הפונקצייה.

(define high-every

_____
_____
_____
_____
_____
_____
_____

שאלה 6 – Reduce

א. [**6** נק'] ממש/י ב **Typescript** פונקצייה שמחזירה את מכפלת כל המספרים במערך (רשימה) נתון בעזרת **reduce**.
--- ציין/י את ה **full-type** של הפונקצייה.
לדוגמה

```
> product([1,2,3])
6
```

_____
_____
_____
_____
_____
_____

ב. [**4** נק'] מה צריך להיות הערך של **product ([])**?
נמק/י

_____
_____
_____
_____

ג. [**4** נק'] ממש/י ב **Typescript** פונקצייה שממשת את פעולת "עצרת" ע"י שימוש ב **range** מספריית **ramda** וב **product** מהסעיף הקודם.
--- ציין/י את ה **full-type** של הפונקצייה.

```
> factorial(3)
6

> import * as R from 'ramda';

R.range(1,5)

[ 1,2,3,4 ]
```

_____
_____
_____
_____
_____
_____

ד. [**6** נק'] ממש/י ב **Typescript** את הפונקצייה **map(.)** על ידי הפונקצייה **reduce(.)**
--- ציין/י את ה **full-type** של הפונקצייה.

ה. [**6** נק'] ממש/י ב **Typescript** את הפונקצייה **filter(.)** על ידי הפונקצייה **reduce(.)**
--- ציין/י את ה **full-type** של הפונקצייה.

שאלה 7 – Typing

א. [**4** נק'] תאר/י שני יתרונות פוטנציאלים לשפות תכנות שהן **Untyped**

ב. [**4** נק'] מה ההבדל בין **nominal typing** ל- **structural typing**?

ג. [4 נק'] האם יכולים להיות שני **types** כך שאחד הוא **subtype** של השני ב- **Typescript** אך לא ב - **Java** . אם כן תן/י דוגמה. אם לא נמק/י.

ד. [4 נק'] האם יכולים להיות שני **types** כך שאחד הוא **subtype** של השני ב- **Java** אך לא ב -**Typescript** . אם כן תן/י דוגמה. אם לא נמק/י.

ה. [4 נק'] השלם למה שווים הביטויים הבאים:

$$\{a, b\} \uplus \{x, y\} =$$

$$\{a, b, c, d\} \uplus \{c, d, e\} =$$

ו.

ו. [4 נק'] הוסף/הוסיפי הצהרות של **Types** לתוכניות הבאות ב **Typescript** :

```
const d2 :_______
      = (x) => (2*x)

const Church: ____________________________________________
     = (f, n) => n === 0 ? (x) => x :
                           (x) => f(Church(f,n-1)(x))

Church(d2,8)(1)
```

```
import {map} from 'ramda';

interface Date {
    year : __________;
    month : __________;
    day : __________;
}

interface person {
    birthDate : __________;
    name : __________;
}

const Year : ______ =
      date => date.year;

const computeAges : ____________________________________ =
      persons => map(p=>2017-p.birthDate.year, persons);

{
    let persons = [{name:"avi", birthDate:{year:1990, month:7, day:10}},
                   {name:"batia", birthDate:{year:1994, month:3, day:2}}];
    computeAges(persons)
}
```