תאריך הבוחן: 10.5.2018
שם המרצה: מני אדלר,מיכאל אלחדד, ירון גונן
מבחן בקורס: עקרונות שפות תכנות
מס' קורס: 202-1-2051
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה
שנה: ב' סמסטר: ב'
משך הבוחן: 2 שעות
חומר עזר: אסור

הנחיות כלליות:

1) ההוראות במבחן מנוסחות בלשון זכר, אך מכוונות לנבחנים ולנבחנות כאחד.
2) מבחן הכתוב בעיפרון חלש המקשה על הקריאה, **לא יבדק**
3) יש לענות על כל השאלות בגוף המבחן בלבד (בתוך השאלון). מומלץ לא לחרוג מהמקום המוקצה.
4) אם אינך יודע את התשובה, ניתן לכתוב "לא יודע" ולקבל 20% מהניקוד על הסעיף/השאלה.

שאלה 1 [30 נק'] AST

ניתנה הגדרת התחביר של שפת L2 ב-BNF וב-AST:

```
<exp> ::= <define> | <cexp>
<define> ::= ( define <var-decl> <cexp> ) / DefExp(var:VarDecl, val:CExp)
<cexp> ::= <number>                      / NumExp(val:number)
        |  <boolean>                     / BoolExp(val:boolean)
        |  <prim-op>                     / PrimOp(op:string)
        | <var-ref>                      / VarRef(var:string)
        | (lambda (<var-dec>*) <cexp>+) / ProcExp(args: VarDecl[], body)
        | (<cexp> <cexp>*)               / AppExp(rator:CExp, rands:CExp[]))
<prim-op>  ::= + | - | * | / | < | > | =
<num-exp>  ::= a number token
<bool-exp> ::= #t | #f
<var-ref>  ::= an identifier token
<var-decl> ::= an identifier token
```

```
export type Exp = DefineExp | CExp;
export type CExp = NumExp | BoolExp | PrimOp | VarRef | AppExp;
export interface DefineExp {tag: "DefineExp"; var: VarDecl; val: CExp};
export interface NumExp {tag: "NumExp"; val: number};
export interface BoolExp {tag: "BoolExp"; val: boolean};
export interface PrimOp {tag: "PrimOp", op: string};
export interface VarRef {tag: "VarRef", var: string};
export interface VarDecl {tag: "VarDecl", var: string};
export interface ProcExp {tag: "ProcExp", args: VarDecl[]; body: CExp[]};
export interface AppExp {tag: "AppExp", rator: CExp, rands: CExp[]};
```

ברצוננו להרחיב את השפה L2 עם פונקציה חדשה בשם bound? שבודקת אם משתנה קשור לערך ומתנהגת לפי הדוגמא הבאה

```
(bound? x) ➔ #f
(define x 1) ➔ void
(bound? x) ➔ #t
((lambda (x) (bound? x)) 1) ➔ #t
```

1.א [3 נק'] סוג הביטוי

האם ניתן להגדיר bound? כפונקציה פרימיטיבית חדשה או כביטוי מיוחד חדש (special form) - נמק

1.ב [6 נק'] מה הערך הצפוי לביטוים הבאים:

בהנתן שהאינטרפרטר ממומש לפי **מודל ההחלפה (substitution model)**:

```
(bound? +)
```

```
(bound? #t)
```

```
(bound? (+ 1 2))
```

נמק:

1.ג [7 נק'] הרחב את הBNF ואת הAST:

הרחב את ה-BNF ואת ה-AST של L2 כדי לתמוך ב-?bound כנדרש:

```
<cexp> ::= <number>                  / NumExp(val:number)
        | <boolean>                  / BoolExp(val:boolean)
        | <prim-op>                  / PrimOp(op:string)
        | <var-ref>                  / VarRef(var:string)
        | (lambda (<var>*) <cexp>*) / ProcExp(args:VarDecl[], body: CExp[])
        | (<cexp> <cexp>*)           / AppExp(rator:CExp, rands:CExp[])


<prim-op>  ::= + | - | * | / | < | > | =


export type CExp = NumExp | BoolExp | PrimOp | VarRef | ProcExp | AppExp
```

1.ד [14 נק'] הרחב את L2eval

הרחב את L2eval כדי לתמוך בחישוב ?bound כנדרש:

```
export type Value = number | boolean | PrimOp | Error;

const applyEnv = (env: Env, v: string): Value =>
    isEmptyEnv(env) ? Error(`var not found ${v}`) :
    env.var === v ? env.val :
    applyEnv(env.nextEnv, v);

const L2eval = (exp: CExp | Error, env: Env): Value =>
    isError(exp)  ? exp :
    isNumExp(exp) ? exp.val :
    isBoolExp(exp) ? exp.val :
    isPrimOp(exp) ? exp :
    isVarRef(exp) ? applyEnv(env, exp.var) :
    isProcExp(exp) ? makeClosure(exp.args, exp.body) :
    isAppExp(exp) ? applyProc(L2eval(exp.rator, env),
                              map((r) => L2eval(r, env), exp.rands),
                              env) :
   // If a new computation rule is needed - add here
```

```
    Error(`Bad L2 AST ${exp}`)

const applyProcedure = (proc: Value, args: Value[], env: Env): Value =>
    isError(proc) ? proc :
    !hasNoError(args) ? Error(`Bad argument`) :
    isPrimOp(proc) ? applyPrimitive(proc, args) :
    isClosure(proc) ? applyClosure(proc, args, env) :
    Error("Bad procedure");

const applyPrimitive = (proc: CExp, args: Value[]): Value =>
    ! isPrimOp(proc) ? Error("Not a primitive") :
    proc.op === "+" ? args[0] + args[1] :
    proc.op === "-" ? args[0] - args[1] :
    proc.op === "*" ? args[0] * args[1] :
    proc.op === "/" ? args[0] / args[1] :
    proc.op === ">" ? args[0] > args[1] :
    proc.op === "<" ? args[0] < args[1] :
    proc.op === "=" ? args[0] === args[1] :
    // If a new primitive is needed add here





    Error("Bad primitive op " + proc.op);
```

שאלה 2 [15 נק'] Higher-order Functions

הגדר את הפונקציה **some** בשפת L4:
Some מקבלת 2 פרמטרים – פונקציה שמחזירה ערך בולאני (predicate) ורשימה. היא מחזירה #t כאשר אחד מהאיברים ברשימה מקיים את הפרדיקט.

2.א [5 נק'] הגדר את החתימה ו-type של some

```
;; Purpose: #t if one of the elts in the list satisfies the predicate

;; Signature: some(_____________________________________________

;; Type: ____________________________________________________
```

2.ב [5 נק'] תן 2 דוגמאות שימוש:

```
;; Examples:
;; 1. Show behavior of some on an empty list

;; ____________________________________________ ➔

;; _________________

;; 2. Show behavior on a non-empty list

;; ____________________________________________ ➔

;; _________________
```

2.ג [5 נק'] ממש את some

```
(define empty? (lambda (x) (eq? x '())))
(define some (lambda (_________________

_____________________________________________________

_____________________________________________________
```

שאלה 3 [15 נק'] Rename and Substitute

התבונן בהפעלה של ה-substitution (במובן המתמטי) הבאה:

E o s = ((lambda (x)
(+ x 12
((lambda (y w) (+ y x w)) z)))) o {z = 24, w = 12}

I. Rename E:

Renamed E = ((lambda (_____

_____

_____

II. Rename s:

{_____}

III. Substitute:

E o s = ((lambda _____

_____

_____

IV. Original expression:

כתוב ביטוי ב-L3 שהחישוב שלו במודל ההחלפות דורש את הפעלת ה-substitution E o s:

_____

_____

_____

שאלה 4 [15 נק'] disjoint union types

השלם את הקוד הבא ב-TypeScript לפי ה-pattern של ה- disjoint union types:

```
interface Point { tag: "Point"; x: number; y: number};
interface Circle { tag: "Circle"; center: Point; radius: number};
interface Square { tag: "Square"; upperLeft: Point; side: number};
type Shape = Circle | Square;

const isSquare = ________________________________________;

const isCircle = ________________________________________;

const isShape = _________________________________________;

const makeSquare = ______________________________________;

const makeCircle = ______________________________________;

// Compute the area of a geometric shape
const area = (s: Shape): number =>

_______________________________

_______________________________
```

שאלה 5 [10 נק'] lexical address

5.א השלם את שמות המשתנים בביטוי הבא בהתאם לכתובת הלקסיקלית שלהם:

```
(lambda (x y)

  ((lambda (x) ([+ free] [___ : 0 0] [____ : 1 1]))

   ([+ free] [_____ : 0 0] [_____ : 0 0])) 1)
```

5.ב השלם את הכתובות הלקסיקליות בביטוי הבא:

```
(lambda (a b c)
    (if ([eq? _______] [b _______] [c  _____])
          ((lambda (c)
             ([cons _____] [a _______] [c _______]))
           [a _________])
          [b _______]))
```

שאלה 6 [15 נק'] normal/applicative order

6.א רשום 3 סוגים של חישובים שגורמים לסטרטגיות חישוב -normal ו-applicative להתנהג בצורות שונות:

_____

_____

_____

6.ב רשום יתרון אחד וחיסרון אחד של applicative יחסית ל-normal:

יתרון:_____

_____

חיסרון:_____

_____