מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: ב
תאריך: 10/8/2020
שמות המרצים: מני אדלר, בן אייל, גיל אינציגר, מיכאל אלחדד
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה, שנה ב', סמסטר ב'
משך המבחן: 3:15 שעות
חומר עזר: אסור

הנחיות כלליות:

- יש לענות על כל השאלות בגיליון התשובות. מומלץ לא לחרוג מן המקום המוקצה.
- אם אינכם יודעים את התשובה, ניתן לכתוב 'לא יודע' ולקבל 20% מהניקוד על הסעיף/השאלה.

שאלה 1:	תכנות פונקציונלי	20 נק'
שאלה 2:	תחביר וסמנטיקה	30 נק'
שאלה 3:	מערכת טיפוסים	20 נק'
שאלה 4:	מבני בקרה	20 נק'
שאלה 5:	תכנות לוגי	15 נק'
סה"כ		105 נק'

בהצלחה!

שאלה 1: תכנות פונקציונלי [20 נקודות]

בסעיפים הבאים (א., ו-ב.) אין להגדיר פונקציות עזר ואין להשתמש בפונקציות שלא ניתנו בשאלה.
אל תשכחו לכתוב את הטיפוס והחתימה של כל הפונקציות.

א. כתבו בשפת Scheme את הפונקציה `drop-while` המקבלת פרדיקט ורשימה, ומחזירה את אותה הרשימה ללא הרישא שעונה על הפרדיקט. ניתן להשתמש בפונקציה הפרימיטיבית `not` וכן פונקציות הרשימות `empty?, car, cdr`.
דוגמאות:

```
> (drop-while even? '(0 2 4 5 6 7 8))
'(5 6 7 8)

> (drop-while even? ‘(1 2 3 4 5))
‘(1 2 3 4 5)


;; Signature:
_____________________________________________________________________
;; Type:
_____________________________________________________________________
(define drop-while
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
```

ב. כתבו בשפת TypeScript את הפונקציה uncurry המקבלת:

- פונקציה f של ארגומנט אחד המחזירה פונקציה של ארגומנט אחד

הפונקציה uncurry מחזירה פונקציה של שני ארגומנטים שמחזירה אותה תוצאה כמו f.
אין להשתמש במוטציה או לולאות.
דוגמאות:

```
const curriedAdd = (x: number) => (y: number) => x + y;
curriedAdd(1)(3); // => 4
const add = uncurry(curriedAdd);
add(1, 3); // => 4

const curriedJoin = (x: string) => (y: string) => `${x} ${y}`;
curriedJoin("Hello")("World"); // => "Hello World"
const join = uncurry(curriedJoin);
join("Hello", "World"); // => "Hello World"

const uncurry = ____________________________________________________________
=>
_____________________________________________________________________________
_____________________________________________________________________________
```

שאלה 2: תחביר וסמנטיקה [30 נקודות]

לשפה L4 נוספה צורה מיוחדת (special form) חדשה לשם הגדרת מחלקות.
מבנה ה class כולל את רשימת השדות של המחלקה, ואת המתודות שלה (במבנה של bindings - זוגות של שם המתודה והגדרת הפרוצדורה שלה). המחלקה Pair, לדוגמא, מוגדרת כך:

```
(define den 2)
(define Pair
  (class (a b)
        (
          (first (lambda () a))
          (second (lambda () b))
          (f (lambda () (/ (+ a b) den)))
        )
  )
)
```

כדי לייצר מופע של class נתון, יש 'להפעיל' אותו עם פרמטרים עבור השדות (כמו פעולת הבנאי). פעולה זו מחזירה פונקציה המקבלת symbol המציין את המתודה להפעלה ומבצעת מתודה זו.

```
(define p34 (Pair 3 4))

(p34 ‘first)
→ 3

(p34 ‘second)
→ 4

(p34 ‘f)
→ 3.5
```

המימוש של הצורה החדשה (כפי שניתן בפיתרון של מועד א - אין צורך להכיר אותו מראש, אנו מציינים זאת רק לשם נוחות):

<u>תחביר</u>

```
export interface ClassExp {
  tag: "ClassExp",
  args: VarDecl[],
  bindings: Binding[];
}
```

```
export const makeClassExp = (args: VarDecl[], bindings: Binding[]): ClassExp => {tag:
"ClassExp", args: args, bindings: bindings; }

export const isClassExp = (x: any): x is ClassExp => x.tag === 'ClassExp';
```

סמנטיקה

```
export type Value = SExpValue | Closure | Class;

export interface Class {
   tag: "Class";
   args: VarDecl[];
   bindings: Binding[];
   env: Env;
}

export const makeClass = (args: VarDecl[], bindings: Binding[], Env env): Class => {tag:
"Class", args: args, bindings: bindings, env : Env; }

export const isClass = (x: any): x is Class => x.tag === 'Class';

const applicativeEval = (exp: CExp, env: Env): Result<Value> =>
   ...
   isClassExp(exp) ? evalClass(exp, env) :
   isAppExp(exp) ? safe2((op: Value, args: Value[]) => apply(op, args))
                   (applicativeEval(exp.rator, env),
                    mapResult((rand: CExp) =>
                      applicativeEval(rand, env), exp.rands)) :
    ...

const evalClass = (exp: ClassExp, env: Env): Result<Class> =>
  makeOk(makeClass(exp.args, exp.bindings, env));

const apply = (op: Value, args: Value[]): Result<Value> =>
   isPrimOp(op) ? applyPrimitive(op, args) :
   isClosure(op) ? applyClosure(op, args) :
   isClass(op) ? applyClass(op, args) :
   makeFailure(`Bad procedure ${JSON.stringify(proc)}`);


const applyClass = (cls: Class, args: Value[]): Result<Closure> => {
```

```
    const cases : LitExp[] = map((b: Binding) =>
        makeLitExp(b.var.var), cls.bindings);
    const actions : CExp[] = map((b: Binding) => b.val, cls.bindings);
    const params : VarDecl[] = [makeVarDecl('msg')];
    const body : CExp[] = [makeCondExp(makeVarRef('msg'),cases, actions];
    const extEnv : Env =
          makeExtEnv(map((v: VarDecl) => v.var, cls.args), args, cls.env)))
    return makeOk(makeClosure(params,body, extEnv));
}
```

]

לא חשוב או נדרש לשאלה, אך הפונקציה makeCondExp מקבלת VarRef, רשימת 'מקרים' עבורו (CExps), ורשימת 'פעולות' (CExps), פעולה עבור כל אחד מהמקרים, ומחזירה מבנה של CondExp בהתאם.

לדוגמא:

```
makeCondExp(makeVarRef('x'), [makeLitExp('i'), makeLitExp('j')], [makeNumExp(1),
makeNumExp(2)]);
⇒
AST of the expression:
(cond ((eq? x 'i) 1)
        ((eq? x 'j) 2))
```

[

שני הסעיפים הבאים מתייחסים לנקודת המוצא של הקוד הנ"ל. הם אינם תלויים זה בזה, וניתן לענות על כל אחד מהם בנפרד.

א. הוחלט להוסיף את הצורה החדשה class גם לשפה L3, ובפרט לאינטרפרטר שלה כפי שמומש בכיתה במודל ההחלפה/הצבה (substitution model) ב applicative order.

1.א

- עדכנו את הממשק Class עבור מודל ההצבה.
- עדכנו את הפרוצדורה evalClass עבור מודל ההצבה.
- השלימו את המתודה applyClass עבור מודל ההצבה:

```
export interface Class {
   tag: "Class";



}
```

```
const evalClass = ____________________________________________ =>

____________________________________________
```

```
const applyClass = (cls: Class, args: Value[]): Result<Closure> => {
   const cases : LitExp[] = map((b: Binding) =>
                          makeLitExp(b.var.var), cls.bindings);
   const actions : CExp[] = map((b: Binding) => b.val, cls.bindings);
   const params : VarDecl[] = [makeVarDecl('msg')];
   const body : CExp[] = renameExps(
                     [makeCondExp(makeVarRef('msg'),cases, actions]);
   const litArgs = map(valueToLitExp,__________);
   return makeOk(makeClosure(_____________________________________________));
}
```

[10 נקודות]

2.א

היכן מאוחסנים השדות של המחלקה עם הערכים שלהם במודל הסביבות (המימוש למעלה של L4) והיכן הם מאוחסנים במודל ההצבה (המימוש שלכם ב א.1 עבור L3)

[5 נקודות]

ב. נדרש לממש את הצורה המיוחדת class גם עבור normal order בשפה L4 (כמו שראיתם בתרגיל 3 - במודל הסביבות)

1.ב

עדכנו את הקוד של L4normalApplyProc, applyClass, NonEmptyEnv, makeEnv, applyEnv בהתאם:

```
const L4normalApplyProc = (op: Value, args: CExp[], env: Env): Result<Value> => {
   if (isPrimOp(op)) {
      const argVals: Result<Value[]> = mapResult((arg) => L4normalEval(arg, env), args);
      return bind(argVals, (args: Value[]) => applyPrimitive(op, args));
   } else if (isClosure(op)) {
      const vars = map((p) => p.var, op.params);
      return L4normalEvalSeq(op.body, makeExtEnv(vars, args, op.env));
   }  else if (isClass(op))
         applyClass(_____________________________)
   } else {
```

```
        return makeFailure(`Bad proc applied ${proc}`);
    }
};

const applyClass = (_____________________________________): Result<Closure> => {
     const cases : LitExp[] = map((b: Binding) =>
          makeLitExp(b.var.var), cls.bindings);
     const actions : CExp[] = map((b: Binding) => b.val, cls.bindings);
     const params : VarDecl[] = [makeVarDecl('msg')];
     const body : CExp[] = [makeCondExp(makeVarRef('msg'),cases, actions];
     const extEnv : Env =
           makeExtEnv(map((v: VarDecl) => v.var, cls.args), args, cls.env)))
     return makeOk(makeClosure(params,body, extEnv));
}

export interface NonEmptyEnv {
    tag: "Env";
    var: string;
    val: ____ ;
    nextEnv: Env;
}

export const makeEnv = (_____________________________________________): NonEmptyEnv =>
    ({tag: "Env", var: v, val: val, nextEnv: env});

export const applyEnv = (env: Env, v: string): Result<Value> =>
    isEmptyEnv(env) ? makeFailure("var not found " + v) :
    env.var === v ? makeOk(_____________________________________________________) :
    applyEnv(env.nextEnv, v);
```

[9 נקודות]

ב.2

נתונה המחלקה Math

```
(define norm 2)
(define Math
   (class (x y)
          (
            (add (lamda () (+ x y))
            (square (lambda () (* x x)))
            (normalize (lambda () (/ y norm)))
          )))
```

הדגימו את שלושת התופעות הבאות ע"י קוד המגדיר מופע של Math ומבצע עליו את אחת המתודות:

- מקרה בו החישוב ב applicative order יעיל יותר מהחישוב ב normal order:

(define m1 _____)
(m1 _____)

- מקרה בו החישוב ב normal order מסתיים בהצלחה, בעוד החישוב ב applicative order גורר שגיאת זמן ריצה:

(define m2 _____)
(m2 _____)

- מקרה בו החישוב ב normal order מסתיים בהצלחה, בעוד החישוב ב applicative order אינו מסתיים:

(define m3 _____)
(m3 _____)

[6 נקודות]

שאלה 3: מערכת טיפוסים [20 נקודות]

3.א typing statements
[10 נקודות]

טענה לגבי טיפוסים (typing statement) היא נוסחה שמחברת בין סביבת טיפוסים לקביעת טיפוס של ביטוי בשפה, שנכתבה:

Tenv |- e:T

למשל:

{x:Number} |- (+ 3 x):Number

לכל אחד מה-typing statements הבאים - תענו true או false אם הטענה נכונה או לא - אם לא, תסבירו למה

{f:[T1->T2]} |- (f 7): T2 _____

_____

{f:[T->T]} |- (f x):T _____

_____

{f:[T1->T2], g:[T2->T3], x:T1} |- (g (f x)):T3 _____

_____

{g:[Empty -> T], x: T} |- (g x):T _____

_____

{x:[T->T]} |- (lambda (x) (x x)):[T->T] _____

_____

3.ב type inference
[10 נקודות]

כתבו את רשימת משתני טיפוס ורשימת המשוואות הנגזרות כאשר מבצעים את האלגוריתם של הסקת טיפוסים על הביטוים הבאים (אין צורך לפתור את המשוואות):

דוגמא עבור הביטוי:

((lambda (f x) (f x)) + 4)

```
Expression                    Variable
==============================  ========
((lambda (f x) (f x)) + 4)     T0
(lambda (f x) (f x))           T1
(f x)                          T2
f                              Tf
x                              Tx
+                              T+
4                              Tnum4
```

Construct type equations:

```
Expression                    Equation
==============================  ==========================
((lambda (f x) (f x)) + 4)     T1 = [T+ * Tnum4 -> T0]
(lambda (f x) (f x))           T1 = [Tf * Tx -> T2]
(f x)                          Tf = [Tx -> T2]
+                              T+ = [Number -> Number]
4                              Tnum4 = Number
```

תורכם:

1.ב.3

(lambda (n) : boolean
    (if (= n 0)
        #f
        (even? (- n 1))))

```
Expression                    Variable
==============================  ========
```

```
Expression                      Equation
============================    ==========================
```

שאלה 4: מבני בקרה [20 נקודות]

4.א. [6 נקודות]

כתבו שלוש סיבות מדוע שימוש ב-Promises עדיף על callbacks בתכנות אסינכרוני.
התייחסו בתשובתכם למושגים הקשורים לעקרונות שפות תכנות. רק "פשוט יותר" או "קל יותר" אינן תשובות מספקות.

4.ב.

4.ב.1 [4 נקודות]

כתבו פונקציה שמתנהגת כמו הרכבה של map ו-filter בשם map-filter לפי ההגדרה הבאה:

```
;; Signature: map-filter(f pred? lst)
;; Purpose: collect the values (f x) for x in l such that (pred x) is true.
;; Type: ____________________________________________________
;; Example: (map-filter square even? '(1 2 3 4)) → '(4 16)
(define map-filter
   (lambda (f pred? lst)
```

4.ב.2 [4 נקודות]

כתבו גרסה איטרטיבית (tail recursive) של אותה פונקציה שמשתמשת ב-accumulator:

```
;; Signature: map-filter-iter(f pred? lst)
;; Purpose: collect the values (f x) for x in l such that (pred x) is true in an iterative manner
;; Example: (map-filter-iter square even? '(1 2 3 4)) → '(4 16)
(define map-filter-iter
   (lambda (f pred? lst)
      (letrec ((iter (lambda (lst acc)
```

3.ב.4 [6 נקודות]

כתבו גרסת CPS של map-filter:

```
;; Signature: map-filter$(f$ pred?$ lst cont)
;; Purpose: CPS version of map-filter
;; Type: [[T1 * [T2 -> T3] -> T3] *
;;        [T1 * [Boolean -> T3] -> T3] *
;;        List(T1) *
;;        [List(T2) -> T3] -> T3]
;; Example: (map-filter$ square$ even?$ '(1 2 3 4) id) → '(4 16)
(define map-filter$
    (lambda (f$ pred?$ lst cont)
```

שאלה 5: תכנות לוגי [15 נקודות]

א. כתבו פרוצדורה sum_to/2 שמגדירה יחס בין מספר טבעי n בייצוג Church לבין סכום המספרים מ-0 עד n בייצוג Church. לדוגמה:

```
?- sum_to(s(s(s(0))), Sum).
Sum = s(s(s(s(s(s(0))))))).
```

ניתן להשתמש בפרוצדורה plus המוגדרת כך:

```
nat(0).
nat(s(X)) :- nat(X).

% Signature: plus(X, Y, Z)/3
% Purpose: Z = X + Y
plus(0, X, X) :- nat(X).
plus(s(X), Y, s(Z)) :- plus(X, Y, Z).
```

_____
_____
_____
_____
_____
_____

ב. כתבו את ה-most general unifier של כל אחד מהזוגות הבאים. אם אין, כתבו "אין" והסבירו למה.

1. `[X|[Y|Z]] = [1, 2, 3, 4].`
   _____
2. `f(g(X, Y), f(Z)) = f(g(f(Z), U), f(g(X))).`
   _____
3. `[1|[[X, X|Y]|Z]] = [1, [2, T], 4].`
   _____
4. `f(g(X), Y, p(Z)) = f(g([1]), [X|W], p(Z)).`
   _____

בהצלחה!