מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: א
תאריך: 13/7/2020
שמות המרצים: מני אדלר, בן אייל, גיל אינציגר, מיכאל אלחדד
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה, שנה ב', סמסטר ב'
משך המבחן: 3 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.
אל תשכחו לכתוב את הטיפוס וחתימה של כל הפונקציות.

א. [10 נק]
כתבו ב-Scheme את הפונקציה scan המקבלת פונקציה של שני ארגומנטים, ערך התחלתי ורשימה, ומחזירה רשימה של reduced values משמאל.
דוגמאות:

```
> (scan * 1 '(1 2 3 4))
'(1 1 2 6 24)

> (scan * 1 '(1 2 3 4 5))
'(1 1 2 6 24 120)

> (scan + 0 '(1 2 3 4 5))
'(0 1 3 6 10 15)

> (scan / 1 '(1 2 3 4 5))
'(1 1 1/2 1/6 1/24 1/120)

> (scan + 0 '())
'(0)

;; Signature: ____________________________________________
;; Type: ____________________________________________
(define scan
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
_______
```

ב. [10 נק]

כתבו ב-TypeScript פונקציה **גנרית** zipWith המקבלת פונקציה f של שני ארגומנטים ושתי רשימות באותו אורך, ומחזירה Result של הרשימה הנוצרת ע"י הפעלת f על איברים מתאימים ברשימות. אין להשתמש במוטציה או לולאות. ניתן להשתמש בפונקציות העזר של רשימות שראינו בכיתה:

```
isEmpty<T>(list: T[]): boolean
first<T>(list: T[]): T
rest<T>(list: T[]): T[]
cons<T>(f: T, r: T[]): T[]
makeOk<T>(value: T): Result<T>
makeFailure<T>(message: string): Result<T>
bind<T, U>(r: Result<T>, f: (x: T) => Result<U>): Result<U>
```

דוגמאות:

```
zipWith((x, y) => x + y, [1, 2, 3], [4, 5, 6]);
// => { tag: 'Ok', value: [ 5, 7, 9 ] }
zipWith((x, y) => x + y, [1, 2, 3], [4, 5]);
// => { tag: 'Failure', message: "Lists are not the same length" }

const zipWith = ____________________________________________________________
=>
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
____________________________________________________________________________
_____
```

שאלה 2: תחביר וסמנטיקה [30 נקודות]

נתונה המחלקה Pair ב ++C (לא נדרשת כל היכרות עם ++C בשאלה, זו רק דוגמא להמחשה):

```
float den = 2;

class Pair {

   int a,b;

   public:

   Pair(int i, int j) { a = i; b = j; }

   int getFirst() { return a; }
   int getSecond() { return b; }
   float f() { return (a + b) / den; }
};
```

בשאלה זו נתמקד במימוש הגדרת מחלקות, כמו המחלקה Pair, בשפה L4

א. [5 נקודות]
השלימו את מימוש הפרוצדורה make-pair ב-L4, כפי שנלמדה בכיתה, המייצרת מופע של מעין 'אובייקט' מסוג Pair (דוגמא לשימוש בו מופיע מיד אחרי הפרוצדורה)

```
(define den 2)
(define make-pair
  (lambda (a b)
    (lambda (msg)
      ____________________________________
      ____________________________________
      ____________________________________
     )
   )
)

(define p34 (make-pair 3 4))
(p34 ‘first) → 3
(p34 ‘second) → 4
```

```
(p34 'f) → 3.5
```

ב. [5 נקודות]

הוחלט להוסיף צורה מיוחדת (special form) חדשה לשפה L4 לשם הגדרת מחלקות במנותק מהגדרת המופעים שלהם - class.

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, תוגדר כך:

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

התוספת לתחביר המופשט והקונקרטי **מודגשת**:

```
<program> ::= (L4 <exp>+) / Program(exps:List(exp))
<exp> ::= <define> | <cexp>              / DefExp | CExp
<define> ::= ( define <var> <cexp> )       / DefExp(var:VarDecl, val:CExp)
<var> ::= <identifier>                    / VarRef(var:string)
<cexp> ::= <number>                       / NumExp(val:number)
      |  <boolean>                      / BoolExp(val:boolean)
      |  <string>                      / StrExp(val:string)
      |  ( lambda ( <var>* ) <cexp>+ )   / ProcExp(args:VarDecl[], body:CExp[]))
      |  ( class ( <var>+ ) ( <binding>+ ) )   / ClassExp(args:VarDecl[], bindings:Binding[]))
      |  ( if <cexp> <cexp> <cexp> )     / IfExp(test: CExp, then: CExp, alt: CExp)
      |  ( let ( <binding>* ) <cexp>+ )  / LetExp(bindings:Binding[], body:CExp[]))
      |  ( quote <sexp> )               / LitExp(val:SExp)
      |  ( <cexp> <cexp>* )             / AppExp(operator:CExp, operands:CExp[]))
      |  ( letrec ( binding*) <cexp>+ )  / LetrecExp(bindings:Bindings[], body: CExp)
<binding>  ::= ( <var> <cexp> )           / Binding(var:VarDecl, val:Cexp)
<prim-op>  ::= + | - | * | / | < | > | = | not |  eq? | string=?
             | cons | car | cdr | list | pair? | list? | number?
             | boolean? | symbol? | string?
<num-exp>  ::= a number token
<bool-exp> ::= #t | #f
<str-exp>  ::= "tokens*"
<var-ref>  ::= an identifier token
```

<var-decl> ::= an identifier token
<sexp> ::= symbol | number | bool | string | (<sexp>*)

השלימו את מימוש הייצוג התחבירי (ה AST) של הצורה המיוחדת החדשה בקוד של הפארסר:

```
export interface ClassExp {_____________________________________________________ }

export const makeClassExp = (_____________________________________________): ClassExp =>
  ____________________________________________________________________________;

export const isClassExp = (x: any): x is ClassExp => ________________________________;
```

ג. [5 נקודות]
חוק החישוב עבור המבנה ClassExp מוגדר כדלהלן:

הערך של ביטוי ClassExp הוא ערך מסוג חדש **Class**, הכולל את המידע הרלבנטי להפעלתו בהמשך (בדומה לערך Closure עבור ProcExp)

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

השלימו את הגדרת הממשק Class:

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

הניחו כי הפרוצדורות makeClass, isClass ממומשות בהתאם.

השלימו את הפרוצדורה evalClass הממשת חוק חישוב זה:

```
const applicativeEval = (exp: CExp, env: Env): Result<Value> =>
    ...
    isClassExp(exp) ? evalClass(exp, env) :
    ...

const evalClass = (exp: ClassExp, env: Env): Result<Class> =>
    ____________________________________________________________________;
```

ד. [10 נקודות]

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. לדוגמא: ההפעלה של המחלקה Pair, מיד לאחר הגדרתה, עם הפרמטרים 3,4, מחזירה את אותה פונקציה שההפעלה (make-pair 3 4) החזירה בסעיף א.

```
(define den 2)
(define Pair
        (class (a b) ((first (lambda () a)) (second (lambda () b)) (f (lambda () (/ (+ a b) den))))))
(define p34 (Pair 3 4))

(p34 'first)
→ 3

(p34 'second)
→ 4

(p34 'f)
→ 3.5
```

הפרוצדורה applyClass באינטרפרטר מטפלת במקרה זה של הפעלת מחלקה לשם יצירת 'מופע'. לדוגמא, עבור הביטוי: (Pair 3 4)

השלימו את מימוש הפרוצדורה applyClass (יש למטה חומר עזר לנוחיותכם)

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


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);
```

_____

_____

_____

_____

}

חומר עזר: ניתן להניח שקיימת בקוד האינטרפרטר פונקציה makeCondExp המקבלת 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))
```

ה. [5 נקודות]

האם הצורה המיוחדת החדשה class היא syntactic abbreviation? נמקו בקצרה

התייחסו בתשובתכם ליכולת להגדיר 'אובייקט' כמו p34 במצבים שונים, עם הצורה המיוחדת class ובלעדיה. וכן לשאלה העקרונית האם צורות מיוחדות חדשות בשפה L4 הן קיצור תחבירי.

_____

_____

_____

_____

_____

שאלה 3: מערכת טיפוסים [20 נקודות]

א. [4 נק]

תארו את ההבדל בין שתי שיטות להגדרת נכונות של פונקציה:

- Precondition verification
- Type correctness

התייחסו לאספקטים:

- באילו כלים ניתן להשתמש כדי לבדוק את התכונה (precondition או type correctness)
- איזו תכונה ניתן לבדוק בצורה סטטית ואיזו בצורה דינמית

נתונות ההגדרות הבאות (שהוגדרו בכיתה):

```
const first = <T>(x: T[]): T => x[0];
const rest = <T>(x: T[]): T[] => x.slice(1);
const isEmpty = <T>(x: T[]): boolean => x.length === 0;
type Result<T> = Ok<T> | Failure;
interface Ok<T> {tag: "Ok"; value: T;}
interface Failure {tag: "Failure"; message: string;}
const makeOk = <T>(value: T): Result<T> =>
    ({ tag: "Ok", value: value });
const makeFailure = <T>(message: string): Result<T> =>
    ({ tag: "Failure", message: message });
const isOk = <T>(r: Result<T>): r is Ok<T> =>
    r.tag === "Ok";
const isFailure = <T>(r: Result<T>): r is Failure =>
    r.tag === "Failure";
const bind = <T, U>(r: Result<T>, f: (x: T) => Result<U>): Result<U> =>
    isOk(r) ? f(r.value) : r;
```

ברצוננו להגדיר פונקציה עם precondition:

```
// @Precondition: l is not empty
const first1 = (l: number[]): number => first(l) + first(l)
```

עם ההגדרה הזו, ניתן לקרוא לפונקציה first1 עם פרמטר לא נכון ללא התראה:

```
const d1 = first1([1,2,3]);
// Precondition is not met - and there is no type error
const d2 = first1([]);
```

כדי למנוע טעויות כאלה, ננסה לנצל את מערכת הטיפוסים בכמה דרכים שונות:

ב. [3 נק.]

נגדיר type חדש לייצוג רשימות לא ריקות:

```
type NonEmptyList = number[];
const isNonEmptyList = (x: any): x is NonEmptyList =>
      Array.isArray(x) && x.length > 0;
const first2 = (lst: NonEmptyList): number => first(lst) + first(lst);
```

האם הקריאה הבאה עוברת type checking? הסברו

```
first2([]);
```

ג. [3 נק]
ממש גירסה של הפונקציה בעזרת Result:

```
const first3 = (l: number[]): Result<number> =>

      ____________________________________________

      ____________________________________________
```

ד. [3 נק]
מציעים לממש את הפונקציה בצורה שמשקפת את ה-precondition בחתימה של הפונקציה על סמך ההגדרה האינדוקטיבית של רשימות:

```
type List = EmptyListI | NonEmptyListI;
type EmptyListI = [];
type NonEmptyListI = {first: number; rest: number[]};
```

ממשו את הפונקציה בצורה בטוחה (type safe):

```
const first4 = ____________________________________

      ____________________________________________

      ____________________________________________
```

ה. [4 נק]

בצעו קריאה בטוחה מהפרוצדורות caller1, caller2, caller3, caller4 לכל אחת מארבע הגרסאות first1, first2, first3, first4 בהתאמה. כל אחת מהפרוצדורות מקבלת רשימה של מספרים ומדפיסה את הערך של האיבר הראשון כפול 2 כאשר הוא קיים או לא עושה כלום כאשר הוא לא קיים:

```
const caller1 = (l: number[]): void => {
    // Use first1

      ___________________________________________

      ___________________________________________
}
```

```
const caller2 = (l: number[]): void => {
    // Use first2

      ____________________________________________

      ____________________________________________

}

const caller3 = (l: number[]): Result<void> => {
    // Use first3

      ____________________________________________

      ____________________________________________

}

const caller4 = (l: number[]): void => {
    // Use first4

      ____________________________________________

      ____________________________________________
}
```

ו. [3 נק]

מה הגרסה המומלצת מבין first1, first2, first3, first4 לכתיבת קוד בטוח - נמקו

_____

_____

_____

_____

שאלה 4: מבני בקרה [20 נקודות]

א. [8 נק]

האם הביטויים הבאים הם ב-tail form? אם לא הסבירו למה לא

```
(cond ((f x) (+ x x))
      ((> x 0) 0))                    _____________________________________

(cond ((+ x x) (f x))
      ((> x 0) (- x)))                _____________________________________

(lambda (x) (* x (f x)))              _____________________________________

(lambda (x) (f (* x x)))              _____________________________________
```

ב. [12 נק]

כתבו פונקציה המפצלת רשימה נתונה לשתי רשימות לפי פרדיקט לא פרימיטיבי pred$ בסגנון CPS. ה-continuation מקבל שני ארגומנטים: רשימת האיברים מן הרשימה אשר קיימו את הפרדיקט ורשימת אלו שלא. על שתי הרשימות יתבצע המשך החישוב. השתמשו רק ואך בפונקציות פרימיטיביות.

```
(define even?$
  (lambda (n c)
    (c (even? n))))

; Signature: split$(pred$ lst c)

; Type: ____________________________________________________________
;
;       ____________________________________________________________
; Purpose: Returns the application of the continuation c on two lists:
; 1. A list of members for which the predicate pred$ holds.
; 2. A list of members for which it doesn't.
; Examples:
; (split$ even?$ '(1 2 3 4 5 6 7) (lambda (x y) (list x y)))
; => '((2 4 6) (1 3 5 7))
(define split$
```

```
(lambda (pred$ lst c)
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________))
```

שאלה 5: תכנות לוגי [15 נקודות]

א. [5 נק]

כתבו חוקים ועובדות ב-Prolog שמייצגים את המידע הבא בעזרת הפרדיקטים:

killer/1 dead/1 married/2 eats/2 kills/2 tasty/1 sweet/1

והקבועים esau, avi, sarah, itshak.

1. **עשו הוא רוצח**
2. **אבי ושרה התחתנו**
3. **מי שהורג מישהו הוא רוצח**
4. **יצחק אוכל כל דבר שהוא טעים <u>או</u> מתוק**
5. **מי שנרצח מת**

%1 _____

%2 _____

%3 _____

%4 _____

%5 _____

ב. [4 נק]

מהו ה-mgu unifier של כל אחד מהזוגות הבאים. כאשר אין unifier כתבו "אין" והסבירו למה

```
1. food(bread,X)  =  food(Y,sausage)          _____________________

2. food(bread,X,beer)  =  food(Y,sausage,X)   _____________________

3. [X,1|Z]  =  [1,X,2,4]                      _____________________

4. food(X)  =  X                              _____________________
```

ג. [6 נק]

מייצגים עצים בינארים בעזרת הפונקטורים leaf/1 ו- tree/2 - למשל:

```
tree(leaf(1), tree(leaf(2), leaf(3)))
```

כתבו פרוצדורה swap/2 שמגדירה יחס בין שני עצים בינארים המתקיים כאשר אחד הוא תמונת מראה של השני. למשל:

```
?- swap(tree(tree(leaf(1), leaf(2)), leaf(4)),T).
T = tree(leaf(4), tree(leaf(2), leaf(1))).
```

_____

_____

בהצלחה!