# עבודה 3 בעקרונות שפות תכנות

## שאלה 1

סעיף 1

Special form הוא ביטוי מורכב שיש לו חוק חישוב אחר מאשר ביטוי מורכב רגיל (application expression) בו מחשבים את הביטוי הראשון (האופרטור) והאופרנדים (באופן בלתי תלוי) ולאחר מכן מפעילים את האופרטור (Closure or primitive operator) עם ה-Values של האופרנדים.

Let לא מחושב באותו האופן המתואר לעיל ולכן let הוא Special form.

סעיף 2

טעויות סמנטיות היכולות להיגרם בעת הפעלת תוכנית L3:

1. הפעלה לא תקינה של אופרטורים פרימיטיביים ו-Closure. דוגמאות:
   1. חלוקה באפס.
   2. Type לא נכון של הפרמטרים (למשל העברת ערכים בוליאניים להפעלת +).
   3. מס' לא נכון של פרמטרים (למשל הפעלה ללא פרמטרים של -).
2. חישוב לא תקין של ביטויי Special form.  
   לדוגמה: הערכת תנאי (הרכיב הראשון של if) שאיננה מחזירה ערך בוליאני.
3. Evaluation של ביטוי מורכב שהרכיב הראשון שלו איננו Primitive operator, איננו Closure ואיננו Special form keyword.  
   לדוגמה: (3 1 4).
4. ניסיון חישוב של משתנה חופשי. לדוגמה: (L3 (+ x 3)).

סעיף 3

סעיף 3.1

הרחבת ה-AST של L3 על מנת לתמוך בקבלת values כדי להימנע משימוש ב-valueToLitExp:

;; <program> ::= (L3 <exp>+) // Program(exps:List(Exp))

;; <exp> ::= <define> | <cexp> | <value> / DefExp | CExp

;; <value> ::= <number> / NumExp(val:number)

| <boolean>

| <string>

| <prim-op>

| <Closure>

| SymbolSExp

| EmptySExp

| CompoundSExp

;; <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(params:VarDecl[], body:CExp[]))

;; | ( 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[]))

;; <binding> ::= ( <var> <cexp> ) / Binding(var:VarDecl, val:Cexp)

;; <prim-op> ::= + | - | \* | / | < | > | = | not | and | or | eq? | string=?

;; | cons | car | cdr | pair? | number? | list

;; | boolean? | symbol? | string? ##### L3

;; <num-exp> ::= a number token

;; <number> ::= a number value

;; <bool-exp> ::= #t | #f

;; <str-exp> ::= a sequence of characters between double quotes

;; <var-ref> ::= an identifier token

;; <var-decl> ::= an identifier token

;; <sexp> ::= symbol | number | bool | string |

;; (<sexp>+ . <sexp>) | ( <sexp>\* ) ##### L3