Assignment 3 theoretical questions

חלק 1:

1. ביטוי let בשפה L3 הוא Special Form, מכיוון שהוא דורש חוק חישוב שונה מהחישוב הרגיל לביטויים מורכבים (ביטויי הפעלה).
2. נתן 4 סוגי שגיאות סינטקטיות:
   א. חישוב ביטוי מסוג $VarRef$ שאין לו הכרזה בקוד.
   ב. הפעלת $Cexp$ שאינו $ProcExp$ או $PrimOp$ בחישוב ביטוי הפעלה.
   ג. הפעלת $ProcExp$ על יותר פרמטרים מהנדרש בביטוי הפעלה.
   ד. הפעלת $PrimeOp$ על פרמטרים לא חוקיים (לדוגמא, הפעלת + על $\#t$ ו-$\#f$).
3. א. נגדיר <value> ו-*<sexp-value>*:

   <value> ::= <sexp-vlaue>

   <sexp-value> ::= <sexp> | <prim-op> | <closure>

   ונוסיף את <value> כ sub-type של <cexp>.

   ב. המקום היחידי אותו נצטרך לשנות הינו הפונקציה applyClosure, שכן כעת לא נצטרך לבצע המרה של Value ל-LitExp. משתמע מכך שגם נוכל למחוק את הפונקציה valutToLitExp.

   ג. היתרון של valueToLitExp – בשיטה זו אנו מפרידים בין ערכים לביטויים, מה ששומר על עקביות במהלך תרגום השפה.

   היתרון של שינוי ה types – נצטרך לכתוב פחות קוד בשיטה זו. בנוסף לכך, בשיטה זו אנחנו לא יוצרים ביטויים יש מאין במהלך החישוב.

4. הסיבה לכך היא שבשיטה הנורמלית, אנחנו לא מחשבים את ה-args לפני הפעלת ה closure עליהם. משום שפונקציית substitute מקבלת []CExp נוכל פשוט לבצע את ההחלפה על args, ולא נצטרף לעשות valueToLitExp עליהם, שכן הם לא Value, אלא הם כבר CExp.

5. <u>תוכנית שמהירה יותר בחישוב נורמלי:</u>

   ((define f (lambda (x) 7))

   (f (+ (* 1 5) (+ 1 4)))

   <u>תוכנית שמהירה יותר בחישוב אפליקטיבי:</u>

   ((define f (lambda (x) (+ x x)))

   (f (* 2 5))

חלק 3:

1. הבעיה בצורת החישוב הנוכחית מול חישוב במודל ה-lazy של השפה scheme, מופיעה בחישוב ביטויי define. בחישוב של ביטויים מסוג זה, במודל lazy ה-val נשמר בסביבה כמו שהוא והחישוב שלו נדחה לשימוש עתידי (אם הוא מסוג LetExp, IfExp, AppExp) לעומת זאת במודל שלנו ה-val מחושב מיד, כמו בחישוב applicative.

הפתרון הוא לשמור את ה-val של ה-define כערך מורכב, בדומה ל-closure, אשר נחשב אותו בשלב מאוחר יותר, במידת הצורך.

3. במימוש שלנו, closure נשאר sub-type של Value.