שאלות לתרגול לקראת בחינה 2024ב

שאלה 1:

בספר הלימוד בעמודים 113-120 מתוארת שפת **"וירמוז"**(IMPLICIT-REFS).

ברצוננו לשנות בשפה זו את אופן ההגדרה של פרוצדורות כך שבתחילת ה-body שלהן תהיה אפשרות להגדרת משתנים מקומיים סטטיים (static variables). משתנה מקומי סטטי מאותחל בקריאה הראשונה לפרוצדורה, ובכל קריאה אחרת לפרוצדורה (השניה והלאה) ערכו ההתחלתי שווה לערך האחרון שלו מהקריאה הקודמת.(במילים אחרות, זהו משתנה משותף לכל הקריאות לפרוצדורה).

נשנה את אופן ההגדרה של פרוצדורות על פי הדקדוק הבא המאפשר הגדרת משתנים מקומיים סטטיים:

Expression ::= proc (***Identifier***){static ***Identifier***= ***Expression***}*'***Expression***

proc-exp (var statvars statvals body)

נסביר דקדוק זה. לפרוצדורה יש פרמטר יחיד (var) לאחר מכן, יכולים להופיע 0 או יותר משתנים לוקליים סטטיים (statvars) ולכל משתנה ערך התחלתי הניתן ע"י (statvals) ולבסוף מופיע ביטוי שהוא גוף הפרוצדורה (body).

להלן דוגמה להמחשת השימוש באופן החדש להגדרת פרוצדורות:

```
letp = proc (x)
static y = 5
static z = 10
begin
set y = -(z,x);
set z = -(x,4);
        -(y,z)
end
in
-((p 12), (p 2))
```

ערכו של ביטוי זה הוא: 18- , בקריאה הראשונה (p 12) מוחזר ערך של 10- , ואילו בקריאה השנייה (p 2) מוחזר ערך של 8.

שאלה 2:

להלן נתונה תוכנית בשפת "וישגור" :

(proc (w) (proc (x) -(x,7) w) 10)

מה תהיה תוצאת הרצת התוכנית?

א) התוכנית בכלל לא עוברת של שלב הפרסור
ב) התוכנית כן עוברת את שלב הפרסור אך זורקת שגיאה במהלך הריצה
ג) (num-val 3)
ד) (num-val 10)
ה) (proc-val …)

שאלה 3:

להלן נתונה תוכנית בשפת "וישגור" :

((proc (x) proc (x) (x 12) 7) proc (x) -(x,8))

מה תהיה תוצאת הרצת התוכנית?

א) התוכנית בכלל לא עוברת של שלב הפרסור
ב) התוכנית כן עוברת את שלב הפרסור אך זורקת שגיאה במהלך הריצה
ג) (num-val 3)
ד) (num-val 7)
ה) (proc-val …)
ו) (num-val 4)
ז) (bool-val #t)

שאלה 4:

להלן תיאור של התחביר המוחשי (concrete syntax) והתחביר המופשט (abstract syntax) של שפת "ויהי" (שפת LET) כפי שמופיע בספר הלימוד בעמוד 60.

Program ::= *Expression*
`a-program (exp1)`

Expression ::= *Number*
`const-exp (num)`

Expression ::= `-(`*Expression* `,` *Expression*`)`
`diff-exp (exp1 exp2)`

Expression ::= `zero?` `(`*Expression*`)`
`zero?-exp (exp1)`

Expression ::= `if` *Expression* `then` *Expression* `else` *Expression*
`if-exp (exp1 exp2 exp3)`

Expression ::= *Identifier*
`var-exp (var)`

Expression ::= `let` *Identifier* `=` *Expression* `in` *Expression*
`let-exp (var exp1 body)`

Figure 3.2 Syntax for the LET language

ברצוננו להרחיב את השפה עם ביטוי חדש בשם cond.
להלן הדקדוק המגדיר את התחביר של cond :

Expression ::= cond { ***Expression*** ==> ***Expression***}* end

cond-exp (exps1 exps2)

בביטוי cond הביטויים הרשומים משמאל לסימן ==> משוערכים לפי הסדר עד אשר ערכו של אחד מהם הוא true. ערכו של כל ביטוי ה- cond שווה לערכו של הביטוי הרשום מימין לסימן ==> השייך לביטוי משמאל שערכו היה true. במקרה שערכם של כל הביטויים הרשומים משמאל לסימן ==> הוא false תודפס הודעת שגיאה.
להלן דוגמא להמחשת השימוש בביטוי החדש :

```
let x=5
  in cond
       zero? (x)       ==> 7
       zero?  (-(x,5)) ==> 12
       zero?  (-(x,2)) ==> 70
     end
```

ערכו של כל ביטוי ה-let בדוגמא זו הוא 12.

<u>שאלה 5:</u>

שאלה זו עוסקת בשדרוג של שפת **"וישגור"** (שפת PROC) המתוארת בספר הלימוד בפרק 3 בעמודים 74-81.
שפה זו מאפשרת הגדרה והפעלה של פרוצדורות לא רקורסיביות.

ברצוננו להרחיב את השפה עם ביטוי חדש בשם fold.

להלן הדקדוק המגדיר את התחביר של fold:

Expression ::= fold ***Expression Expression*** [{ ***Expression***}$^{*(,)}$]

fold-exp (proc1 acc vals)

ביטוי **fold**, מכיל את המרכיבים הבאים: ביטוי (**proc1**) שאמור להיות פרוצדורה עם פרמטר יחיד. ביטוי נוסף (**acc**) המכיל ערך התחלתי, ורשימת ביטויים המופרדים בפסיק ורשומים בתוך סוגריים מרובעות.

ביטוי **fold** מפעיל את הפרוצדורה **proc1** על כל אחד מאיברי הרשימה **vals**, וצובר את תוצאות ההפעלות (פעולת חיבור) ל- **acc**. לבסוף מוחזר ערכו של הצובר **acc** כערכו של כל ביטוי **fold**.

להלן דוגמה להמחשת השימוש בביטוי החדש:

```
let x= fold  proc (y) –(y , -1)  0  [12,–(4,7),9]

in

  -(x,5)
```

ביטוי ה-let מגדיר משתנה בשם x ושם בו את תוצאת ביטוי ה-fold שהיא 21, מכיוון שהפעלת הפרוצדורה (המחזירה את הפרמטר שקיבלה בתוספת 1) על כל אחד מאיברי הרשימה הנתונה, תוך שימוש בצובר התחלתי שערכו 0 יוצרת את הסכום 21.

ולכן ערכו של כל ביטוי ה-let בדוגמה זו הוא 16.

שאלה 6:

להלן נתונה תוכנית בשפת "**וישגור**" :

```
proc (w)
let x=10
in
if zero? (-(x,3))  then (w x) else (w 8)
```

תארו במילים את תוצאת ריצת התוכנית.

שאלה 7:

שאלה זו עוסקת בשדרוג של שפת "וישגור" (שפת PROC) המתוארת בספר הלימוד בפרק 3 בעמודים 74-81.

בשאלה זו נרצה להרחיב את השפה, כך שיתאפשר לעבוד עם tuples (בדומה לקיים בשפת פייתון).

tuple היא בעצם n-יה סדורה, המאפשרת להחזיק יחד מספר ערכים מסוגים שונים, וכן ניתן לפרק מתוך ה-tuple חלק מהמרכיבים שלו על פי בחירתנו לתוך משתנים.

נרצה להרחיב את שפת "וישגור" עם השינויים הבאים :

- ביטוי חדש המחזיר tuple
- עדכון התחביר ואופן החישוב של ביטוי let כך שידע לעבוד מול tuple, ולאפשר הגדרת מספר משתנים זמניים שערכם מתקבל מחילוץ מרכיבים מתוך tuple

להלן הדקדוקים המגדירים את השינויים :

Expression **::=** < { ***Expression*** }$^{+(,)}$ >

tuple-exp (exps)

Expression **::=** let ***temps*** = ***Expression*** in ***Expression***

temps **::=** ***identifier*** | [{_***identifier***}$^{+(,)}$]

let-exp (tmps, exp1, body)

הסבר על המרכיבים של הביטויים החדשים:

- ביטוי tuple-exp נכתב בסוגריים משולשים הכוללים בתוכם אוסף של ביטויים (לפחות אחד) המופרדים בפסיקים. הביטוי יחזיר tuple המייצג את אוסף תוצאות רשימת הביטויים.
- לביטוי let החדש, מבנה דומה לביטוי let הקיים כבר בהגדרות השפה, פרט לכך שבמקום הגדרת משתנה זמני בודד כפי שהיה עד כה, נוכל לרשום במקום זאת מרכיב המוגדר ע"י temps. מרכיב temps מאפשר לבחור להציב או שם משתנה בודד כפי שהיה עד כה, או שמות משתנים שיקבלו ערכים המחולצים מתוך

tuple. שמות המשתנים ניתנים בתוך סוגריים מרובעות, ובתוכן רשימת שמות משתנים מופרדים בפסיקים עם מבנה תואם ל- tuple ממנו מחלצים ערכים לתוך המשתנים, אם ברצוננו לא לחלץ מרכיב/ים מסוימים מתוך tuple אזי במקום שם משתנה ייכתב _ (מקף תחתון) וכך נוכל להתעלם מחלק ממרכיבי ה-tuple ולחלץ רק את חלקם. בתוך הסוגריים המרובעות חייב להופיע לפחות שם אחד של משתנה, לא ניתן שכל השמות יהיו _ (מקף תחתון).

<u>דוגמאות להמחשת אופן השימוש במרכיבים החדשים:</u>

<u>דוגמה 1:</u>

```
> (run
      "let t1=<20,proc (x) -(x,1), 50>
       in
        let [_,p,_] = t1
        in (p 200)")
(num-val 199)
>
```

<u>הסבר דוגמה 1:</u>

בדוגמה 1, ב- let החיצוני מוגדר משתנה בשם t1 (כמו בתחביר המקורי) וערכו הוא tuple המכיל 3 מרכיבים, הראשון המספר 20, השני פרוצדורה, והשלישי המספר 50. בביטוי ה- let הפנימי, מוגדר משתנה p וערכו הוא המרכיב השני מתוך ה-tuple שהוגדר קודם (t1). מאחר ואנו לא מעוניינים לחלץ מתוך ה-tuple את המרכיב הראשון והשלישי שלו, נרשמו במקומות אלה _ (מקפים תחתונים). ולכן, ה-let הפנימי הגדיר משתנה p שערכו הוא פרוצדורה.

ב-body של ה- let הפנימי מופעלת הפרוצדורה ש-p מייצג על הערך 200, ופרוצדורה זו מחזירה 199.

<u>שימו לב</u>, ה- let החיצוני אינו פעולת חילוץ מרכיבים מ- tuple (אין שם שימוש בסוגריים מרובעות), אלא הגדרת משתנה בשם t1 שערכו הוא כל ה-tuple

דוגמה 2:

```
> (run
     "let [_,a,_,_,b,_]= <15,25,35,45,55,65>
      in  -(b,a)")
(num-val 30)
>
```

הסבר דוגמה 2:

בדוגמה 2, ביטוי ה-let מגדיר 2 משתנים זמניים, a ו- b שאת ערכם הם מקבלים מחילוץ המרכיבים השני והחמישי מתוך ה- tuple <15,25,35,45,55,65>, ולכן הערך של משתנה a הוא (num-val 25) וערכו של משתנה b הוא (num-val 55), ולכן תוצאת ה- body היא 25-55, וערכה של כל התוכנית הוא (num-val 30)

שימו לב, בחילוץ איברים מ- tuple, נדרש שצד ימין יהיה ביטוי שתוצאתו מסוג tuple, וצד שמאל יהיה בעל מספר מרכיבים תואם לזה ב-tuple.

יש לשים לב ולבדוק מקרים של שימוש לא תקין בתחביר (כגון: רשימות שלא מכילות לפחות איבר אחד, חילוץ מתוך ערך שאינו tuple, חילוץ במבנה שאינו תואם למבנה ה- tuple וכדומה)
במקרים של שימוש לא תקין, יש להדפיס הודעות שגיאה מתאימות.

ממשו והטמיעו את השינויים הדרושים בתוך שפת "**וישגור**" (**שפת PROC**).

הקפידו להסביר **היכן בדיוק** נדרשים שינויים ותוספות בקבצי המפרש, **ומהם** השינויים והתוספות.

<u>שאלה 8</u>

בספר הלימוד בעמודים 113-120 מתוארת שפת "**וירמוז**"(IMPLICIT-REFS).

ברצוננו להרחיב שפה זו עם ביטוי חדש בשם switch-exp

להלן הדקדוק המגדיר את הביטוי החדש שברצוננו להוסיף:

Expression ::= switch ***Expression*** {
{***Type identifier*** when (***Expression***) => ***Expression***}$^{+(,)}$
default => ***Expression***
}

Type ::= number | boolean | function

switch-exp (e1 typs ids bools exps

שימו לב, Type המוגדר בשאלה זו, אינו קשור כלל ל- Type שפגשנו בפרק 7.

<u>הסבר על מרכיבי הביטוי ואופן פעולתו:</u>

ביטוי **switch-exp** מורכב מהמרכיבים הבאים:

- e1 – הוא ביטוי שאת תוצאתו רוצים לבדוק.
- typs – רשימה של טיפוסים מסוג Type
- ids – רשימה של identifiers
- bools - רשימת ביטויים שתוצאת כל אחד מהם בוליאנית
- exps – רשימת ביטויים כלשהם
- defexp – ביטוי כלשהו

ביטוי switch-exp מחשב את ערכו של e1 ולאחר מכן מחפש באינדקסים תואמים מתחילת הרשימות typs, ids, bools, exps , האם קיימת התאמה בין טיפוס הערך של e1 לטיפוס באינדקס הנבדק (למשל אם ערכו של הביטוי הוא מסוג num-val ובשורה הטיפוס הוא number אז יש התאמה, כנ"ל עבור bool-val ו- boolean, ועבור proc-val ו-function), אם אין התאמה, יש לעבור לאינדקס הבא ברשימות. אם נמצאה התאמה, יש לחשב את ערכו של הביטוי הנמצא באינדקס הנוכחי ברשימה bools, באמצעות סביבה הבנויה מהרחבה של הסביבה הנוכחית, עם משתנה חדש בשם כפי המופיע ברשימת ids באינדקס הנוכחי, וכריכתו לערך של הביטוי e1. אם חישוב הביטוי הניב תשובה בוליאנית של false יש להמשיך הלאה לאינדקס הבא, ואם חישוב הביטוי הניב תשובה בוליאנית של true, יש להפסיק את כל החישוב של ביטוי ה-switch ולהחזיר את ערכו של הביטוי הנמצא באינדקס הנוכחי ברשימה exps.
אם לא נותרו איברים ברשימות (כי לא נמצא באף אינדקס ברשימות איברים המקיימים את התנאים לעיל) במקרה זה, ביטוי switch-exp יחזיר את ערכו של הביטוי defexp

להלן דוגמאות להמחשת השימוש בביטוי switch-exp :

דוגמה 1:

```
> (run
   "let x=15
     in
      let y= switch proc (b) -(b,4)
             {
               boolean i when (if i then zero? (x) else zero?(0) ) => 90,
               number  a when ( zero?(x)) => 100,
               function f when ( zero? (-(x,15))) => (f 300),
               number  i when ( zero? (-(i,11))) => 200
               default => 400
             }
       in
          y")
(num-val 296)
>
```

בדוגמה 1, ביטוי switch מקבל לבדיקה ביטוי שערכו פרוצדורה, ביטוי switch בודק את השורה הראשונה, ושם כתוב boolean ואין כאן התאמה לסוג הביטוי הנבדק (כי הוא פרוצדורה ולא בוליאני), ולכן עוברים לבדוק את השורה הבאה, שאף היא אינה מתאימה (כתוב שם number). בשורה השלישית נמצאת המילה function ולכן שורה זו תואמת לטיפוס הנבדק, ולכן בודקים כעת האם הביטוי שכתוב לאחר המילה when ערכו הוא true, ואכן, הביטוי ((x,15)-) zero? הוא true כי ערכו של x הוא אכן 15, לכן מצאנו שורה תואמת לכל התנאים, וערכו של כל ביטוי ה-switch יהיה כערכו של הביטוי (f 300) המחזיר 296 ולכן ערכו של y המוחזר הוא 296.

דוגמה 2:

```
> (run
   "let x=15
     in
      let y= switch -(x,4)
             {
               boolean i when (if i then zero? (x) else zero?(0) ) => 90,
               number  a when ( zero?(a)) => 100,
               number  i when ( zero? (-(i,11))) => 200,
               function f when ( zero? (x)) => 300
               default => 400
             }
       in
          y")
(num-val 200)
>
```

בדוגמה 2, ערכו של הביטוי הנבדק ע"י switch הוא מסוג num-val ולכן רק שורות מסוג number באות בחשבון, השורה הראשונה מסוג number אינה מתאימה משום שערכו של הביטוי שלאחר המילה when הוא false (כי ערכו של a הוא 11 ולא 0) ולכן שורה זו אינה מתאימה. השורה השלישית היא גם מסוג number ובה ערכו של הביטוי לאחר המילה when הוא true, כי ערכו של i הוא אכן 11. ולכן מצאנו שורה מבוקשת וערכו של כל ביטוי ה-switch יהיה 200, ולכן ערכה של כל התוכנית הוא כערכו של y שקיבל 200.

דוגמה 3 :

```
> (run
   "let x=15
     in
      let y= switch proc (b) -(b,4)
             {
               boolean i when (if i then zero? (x) else zero?(0) ) => 90,
               number  a when ( zero?(x)) => 100,
               function f when ( zero? (x)) => (f 300),
               number  i when ( zero? (-(i,11))) => 200
               default => 400
             }
       in
          y")
(num-val 400)
>
```

בדוגמה 3, אף אחת מהשורות אינה מתאימה לביטוי הנבדק (או שאינה מהטיפוס המתאים או שהתנאי הבוליאני הוא false) ולכן מוחזר ערכו של הביטוי משורת ה-default שהוא 400

דוגמה 4 :

```
> (run
   "let x=15
     in
      let y= switch proc (b) -(b,4)
             {

               default => 400
             }
       in
          y")
interp.scm:115:20: Error : switch cases is missing
>
```

בדוגמה 4, נרשמה רק שורת ה-default ועל פי הדקדוק הנתון בשאלה נדרשת לפחות שורה אחת של תנאים בנוסף ל-default, ועל כן, הביטוי אינו נכון תחבירית, ולכן התקבלה הודעת שגיאה יזומה על כך שחסרים מקרי בדיקה.

ממשו והטמיעו את ביטוי switch-exp בתוך שפת **"וירמוז"** (**שפת IMPLICIT-REFS**).
הקפידו להסביר **היכן בדיוק** נדרשים שינויים ותוספות בקבצי המפרש, **ומהם** השינויים והתוספות.

יש לשים לב ולבדוק מקרים של שימוש לא תקין בתחביר.
במקרים של שימוש לא תקין, יש להדפיס הודעות שגיאה מתאימות.

<u>שאלה 9:</u>

בספר הלימוד בעמודים 113-120 מתוארת שפת **<u>"וירמוז"(IMPLICIT-REFS).</u>**

ברצוננו להרחיב שפה זו עם יכולות חדשות שיאפשרו הגדרה של generator ושימוש בו, בדומה לקיים בשפת פייתון.

נסביר תחילה מהו generator?

Generator הוא אוסף של ערכים, בהם ניתן להשתמש פעם אחת בלבד. לאחר שימוש בערכי ה-generator, הוא מתרוקן, ובשלב זה, לא ניתן לחלץ עוד ערכים ממנו.

להלן הדקדוק המגדיר את הביטויים החדשים שיש להטמיע בשפה:

Expression ::=
generator (*Identifier*) : [{ *Expression* }*] yield *Expression*

gen-exp (var exps retexp)

Expression ::= ::*Identifier*

return-exp (gen)

Expression ::= ??*Identifier*

empty-exp (gen)

<u>הסבר על מרכיבי הביטויים ואופן פעולתם:</u>

ביטוי **gen-exp** מורכב מהמרכיבים הבאים:

- var – שם של משתנה המוכר רק במסגרת פעולת ה-generator ובכל פניה ל-generator הוא יקבל את הערך הבא מרשימת הערכים שעוד נותרו ב- exps
- exps – רשימת ביטויים שאת ערכן (אחד בכל פעם) יש לקשור ל-var בכל פעם שפונים ל-generator על מנת לחלץ את האיבר הבא.
- retexp – ביטוי המתאר את הערך שיש להחזיר מה- generator בכל פעם שפונים אליו.

הביטוי בונה ומחזיר את ה-generator. הביטויים השונים המוזכרים בהגדרת ה- generator יחושבו על פי הסביבה בעת הגדרת ה-generator (כלומר static-binding)

שימו לב, שיש כאן טיפוס נתונים חדש לשפה.

ביטוי **return-exp** מורכב מהמרכיבים הבאים:

- gen – הוא שם מזהה (identifier) של generator

הביטוי מחזיר את הערך הבא מ-gen במידה וקיים, אם לא נותרו ערכים תודפס הודעת שגיאה מתאימה.
במידה ו-gen אינו generator, תודפס הודעת שגיאה מתאימה.

ביטוי **empty-exp** מורכב מהמרכיבים הבאים:

- gen – הוא שם מזהה (identifier) של generator

הביטוי בודק האם gen הוא generator במצב ריק, ומחזיר תשובה בוליאנית בהתאם.
במידה ו-gen אינו generator, תודפס הודעת שגיאה מתאימה.

להלן דוגמאות להמחשת השימוש:

בכל הדוגמאות ניתן להניח שהשפה שודרגה עם פעולת כפל, בדומה להגדרה ומימוש של פעולת חיסור הקיימת בהגדרות השפה.

<u>דוגמה 1:</u>

```
> (run "
        let g= generator(x): [5 2 ] yield *(x,x)
        in
          -(::g,::g)")
(num-val 21)
>
```

בדוגמה 1, מוגדר generator בשם g , המורכב מ-2 ערכים, ומחזיר את ריבועי ערכיו. במסגרת ביטוי החיסור מוחזר בתחילה 25 (5 בריבוע) ולאחר מכן, 4 (2 בריבוע), והתוצאה היא ההפרש ביניהם שהוא 21 (על פי 4-25).
הערה: לאחר החזרת הערך 4 מ-g, ה- generator התרוקן.

<u>דוגמה 2:</u>

```
> (run "
        let g= generator(x): [5 ] yield *(x,x)
        in
          -(::g,::g)")
interp.scm:123:41: Generator is empty
```

בדוגמה 2, מוגדר generator בשם g , המורכב מערך יחיד, ומחזיר את ריבועו. במסגרת ביטוי החיסור מוחזר בתחילה 25 (5 בריבוע), בשלב זה g התרוקן, ולכן ניסיון לחלץ שוב איבר מ-g מחזיר שגיאה.

דוגמה 3:

```
> (run "let i=10
        in
           let g= generator(x): [5 ] yield *(x,x)
           in
             -(::i,::g)")
interp.scm:132:30: Gen-extract: i Not a generator
```

בדוגמה 3, בשלב פעולת החיסור, מנסים תחילה לחלץ מ-i ערך, אך i אינו generator ולכן התקבלה שגיאה מתאימה על כך.

דוגמה 4:

```
> (run "
      let sum=0
      in
       letrec f(g)= if ??g then sum
                           else begin
                                  set sum= -(sum,-(0,::g));
                                  (f g)
                                end
       in
         (f generator(x): [5 3] yield *(x,x))")
(num-val 34)
>
```

בדוגמה 4, מוגדר משתנה sum, לאחריו מוגדרת פרוצדורה רקורסיבית בשם f המקבלת פרמטר בשם g מסוג generator, ומחזירה את סכום ערכיו. בדוגמה זו, f מופעלת על ה-generator המכיל את הערכים [5 3] ומחזיר את ריבועיהם. ולכן, הוחזר 34 שהוא 3^2+5^2.

שימו לב לשימוש בכל הביטויים החדשים ולאופן השימוש בהם כפי שהוגדר בשאלה.

<u>ממשו והטמיעו</u> את הביטויים החדשים בתוך שפת **<u>"וירמוז"</u>** (**<u>שפת IMPLICIT-REFS</u>**).
הקפידו להסביר **<u>היכן בדיוק</u>** נדרשים שינויים ותוספות בקבצי המפרש, **<u>ומהם</u>** השינויים והתוספות.

שימו לב, יש להקפיד על עבודה עם חוקי וכללי השפה הנתונה בשאלה (שפת "וירמוז" – Implicit-Refs)

יש לשים לב ולבדוק מקרים של שימוש לא תקין בתחביר.
במקרים של שימוש לא תקין, יש להדפיס הודעות שגיאה מתאימות כפי שהודגם.

<u>שאלה 10:</u>

להלן תוכנית בשפת "וישגור" :

```
let Cola=0
in
  let Fanta=1
  in
    (proc (a) zero?(if a then Cola else Fanta)  zero?(Cola))
```

מה תהיה תוצאת ריצת התוכנית?

א) Cola
ב) (num-val 0)
ג) Fanta
ד) (num-val 1)
ה) תוצאה מסוג (…. proc-val)
ו) שגיאה בזמן ריצה
ז) אף אחת מהתשובות