בבחינה 3 שאלות. עליכם לענות על כל השאלות.

שאלה 1 (40 נקודות)

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

נרצה להוסיף לשפת "וישגור" מנגנון של memoization עבור קריאות לפרוצדורות בדומה לקיים בשפת התכנות Apache Groovy.

מנגנון memoization מאפשר לנהל זיכרון מטמון (cache) עבור הפעלות של פרוצדורות. כאשר רוצים להפעיל פרוצדורה בודקים תחילה בזיכרון המטמון האם קיימת תוצאת הפעלה קודמת של הפרוצדורה עם אותו ארגומנט, אם כן מחזירים את התוצאה שנמצאת בזיכרון המטמון, אחרת מפעילים ומחשבים את הפרוצדורה ומחזירים את תוצאתה ובנוסף מוסיפים את תוצאת ההפעלה לזיכרון המטמון.

זיכרון המטמון הוא זיכרון קטן בגודל קבוע, ומנוהל לפי LRU (Least Recently Used), כלומר, כאשר זיכרון המטמון מתמלא ורוצים להוסיף לו רשומה חדשה, תחילה יש צורך לפנות מקום לרשומה החדשה ע"י מחיקת הרשומה בזיכרון המטמון שלא הייתה בשימוש לאחרונה, ואז מוסיפים את הרשומה החדשה לתחילת זיכרון המטמון. ואם יש מקום בזיכרון המטמון, אז אין צורך במחיקה ופינוי מקום אלא רק בהוספה של הרשומה החדשה.

מנגנון ה- LRU מנוהל בצורה כזו שכאשר מנסים להפעיל פרוצדורה שתוצאת הפעלתה כבר מאוחסנת בזיכרון המטמון, תוצאת ההפעלה מוחזרת מהמטמון וגם מקודמת לתחילת המטמון על מנת לציין שהשתמשנו בה לאחרונה. ואם התוצאה לא נמצאת במטמון, הפרוצדורה מופעלת ומחושבת ותוצאת הפעלתה מתווספת לתחילת המטמון.

היתרון של מנגנון memoization הוא בחיסכון בזמן חישוב קריאות לפרוצדורות. אם בתוכנית פרוצדורה נקראת מספר רב של פעמים עם אותו ארגומנט, מנגנון memoization חוסך את הצורך לחשב את הפרוצדורה מחדש. דבר המשפר את ביצועי זמן הריצה של התוכנית. דוגמה מובהקת ליתרון זה היא בחישוב של סדרת פיבונאצ'י באופן רקורסיבי שם יש חזרה על חישובים של איברים רבים בסדרה שכבר חושבו.

נדגים תחילה דוגמה של הרצת תוכנית במצב בחדש, ולאחריה יש הסבר מה נדרש מכם לממש במחברת הבחינה.

המשך השאלה בעמוד הבא

להלן דוגמת הרצה של תוכנית במצב החדש:

```
> (run "let p= proc (x) -( x, 2)
        in
          let q = proc (y) -(y ,10)
          in
           let temp = -((p 5) , (q 30))
           in
            -((p 5) ,  -((p 8), (q 30)))")
procedure call result #(struct:num-val 3) found in cache :-)
procedure call result #(struct:num-val 20) found in cache :-)
(num-val 17)
> |
```

<u>הסבר הדוגמה:</u>

בדוגמה זו מוגדרת פרוצדורה p שמחזירה תוצאת הפחתה ב-2 מפרמטר אותו היא מקבלת. וכן מוגדרת פרוצדורה q המחזירה תוצאת הפחתת 10 מפרמטר אותו היא מקבלת. לאחר מכן מוגדר משתנה temp המקבל תוצאת חיסור בין 2 תוצאות של הפעלות פרוצדורות. הפרוצדורה הראשונה היא הפעלת p על 5 , מאחר וזיכרון המטמון ריק בשלב זה, פרוצדורה זו מופעלת ומחושבת ומחזירה ערך של 3, תוצאה זו גם מתווספת לזיכרון המטמון. לאחר מכן מופעלת q על 30, ומאחר ובזיכרון המטמון אין תוצאה של הפעלה זו, הפרוצדורה q מופעלת ומחושבת על 30, ומחזירה תוצאה של 20, ובנוסף תוצאה זו גם מתווספת לזיכרון המטמון. כעת מגיעים לחלק ה-body של ה-let האחרון, ושם מחשבים ביטוי חיסור שמורכב ממספר הפעלות של פרוצדורות, ניתן לראות שתוצאת הפעלת p על 5 , והפעלת q על 30 כבר נמצאים בזיכרון המטמון, ולכן הם נשלפו והוחזרו משם (וגם קיבלנו הדפסת חיווי על כך), ואילו הפעלת p על 8 לא נמצאת במטמון ולכן היא מחושבת ומתווספת למטמון.

המשך השאלה בעמוד הבא

מה נדרש מכם לממש בשאלה זו?

<u>בשאלה זו אין שינוי בתחביר השפה, אלא יש צורך בעדכון ההתנהגות של ביטוי call-exp להפעלת פרוצדורה כך שיפעל על פי מנגנון memoization. ומימוש מבנה נתונים גלובלי המייצג זיכרון מטמון.</u>

זיכרון המטמון ימומש **כמבנה נתונים גלובלי** המדמה זיכרון מטמון המנוהל לפי LRU. זיכרון המטמון שמור במשתנה גלובלי בשם **the-cache**. גודל זיכרון המטמון נתון ע"י המשתנה **cache-size**. זיכרון המטמון ימומש בצורת **רשימה שאיבריה הם תתי רשימות**. כל תת רשימה זוכרת תוצאה של הפעלת פרוצדורה על ארגומנט מסוים. המבנה של כל תת רשימה מורכב מ-3 איברים: פרוצדורה (ייצוג לפי טיפוס הנתונים המופשט proc), ארגומנט שערכו מיוצג כ-expval, ותוצאת הפעלת הפרוצדורה על הארגומנט שגם מיוצגת כ-expval.
כך נראה המבנה של זיכרון המטמון:

((proc arg result) (proc arg result) .....)

מימוש זיכרון המטמון מספק את הפעולות הבאות:

init-cache - פרוצדורה ללא פרמטרים המאתחלת את זיכרון המטמון לראשונה להיות ריק.

search-cache – מקבלת פרוצדורה prc (מיוצגת לפי proc) וארגומנט arg (כ-expval) ומחפשת האם קיימת בזיכרון המטמון תוצאת הפעלה של prc עם arg. במידה וכן, מוחזר אינדקס (החל מ-0) היכן ברשימה המייצגת את המטמון נמצאה התוצאה, אחרת מוחזר ערך בוליאני של #f.

rearrange-cache – מקבלת אינדקס idx (החל מ-0) של איבר בזיכרון המטמון, **ומעדכנת** את זיכרון המטמון הגלובלי למצב שאותו איבר מאינדקס idx מועבר לתחילתו של המטמון.

remove-at – פעולה המקבלת רשימה lst ואינדקס n (החל מ-0), ומוחקת את האיבר מהרשימה. הפעולה מחזירה את הרשימה החדשה.

add-cache – מקבלת פרוצדורה prc (מיוצגת כ- proc) וארגומנט arg (מיוצג כ-expval) ותוצאת הפעלה result (מיוצגת כ- expval) ומוסיפה אותם כאיבר חדש לתחילת המטמון. הפרוצדורה בודקת תחילה שקיים מקום במטמון, ואם לא, דואגת לפנות מקום ע"י מחיקת האיבר האחרון בזיכרון המטמון.

המשך הסבר דרישות המימוש בעמוד הבא

הפרוצדורה value-of-program עודכנה כבר, כך שתדאג לאתחל את זיכרון המטמון למצב ריק.
להלן מימושה המעודכן :

```
(define value-of-program
  (lambda (pgm)
    (init-cache)  ; initialize the-cache to empty
    (cases program pgm
      (a-program (exp1)
        (value-of exp1 (init-env))))))
```

להלן ההגדרות של זיכרון המטמון :

```
(define cache-size 20)
(define the-cache 'uninitialized)
```

<u>עליכם לממש במחברת הבחינה את מימוש הפעולות על זיכרון המטמון כפי שתוארו לעיל, וכן את עדכון ההתנהגות של ביטוי call-exp להפעלת פרוצדורות כך שיפעל לפי מנגנון memorization בעזרת זיכרון המטמון.</u>

<u>מימוש זיכרון המטמון</u> (25 נקודות) – השלימו את כתיבת הפרוצדורות הבאות במחברת הבחינה:

```
; init-cache: () -> update the-cache to empty cache
  (define init-cache
    (lambda ()
      _____________________________
      _____________________________
      _____________________________

      ))
```

המשך השאלה בעמוד הבא

```
;search-cache : return #f if prc,arg not in cache
;                or index(from 0) found in the cache
(define search-cache
 (lambda (prc arg)
   _____________________________
   _____________________________
   _____________________________

 ))

; rearrange-cache: update the-cache by moving the element
;                  at index idx to the beginning of the cache
(define rearrange-cache
  (lambda (idx)
   _____________________________
   _____________________________
   _____________________________

  ))


;remove-at: removing element at index n
;           from list lst and return new list
 (define (remove-at lst n)
   _____________________________
   _____________________________
   _____________________________

 )


; add-cache: update the-cache by adding
;            new procedure call result to the cache
(define add-cache
  (lambda (prc arg result)
   _____________________________
   _____________________________
   _____________________________

 ))
```

המשך תיאור דרישות מימוש בעמוד הבא

<u>עדכון התנהגות ביטוי call-exp</u> (15 נקודות) - השלימו את העדכון במחברת הבחינה:

נדרש לממש עדכון של ביטוי הפעלת פרוצדורה באופן שתחילה בודקים האם קיימת תוצאה בזיכרון המטמון של הפעלה קודמת של הפרוצדורה עם אותו ארגומנט, אם כן, יש להחזיר את התוצאה מהמטמון ולקדמה לתחילת המטמון, וכן להדפיס חיווי שהתוצאה הושגה מהמטמון. אם במטמון לא קיימת תוצאה של הפעלה קודמת של הפרוצדורה עם אותו ארגומנט, אז נדרש לחשב את הפרוצדורה, את תוצאת ההפעלה יש להחזיר וכן להוסיפה לתחילת המטמון. השלימו את העדכון הדרוש הבא של call-exp במחברת הבחינה:

```
;new implementation for call-exp with memoization
(call-exp (rator rand)
   _____________________________
   _____________________________
   _____________________________

)
```

שאלה 2 (30 נקודות)

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

ברצוננו להרחיב שפה זו עם יכולת לשנות לפרוצדורה שכבר הוגדרה את התנהגותה בזמן ריצה ע"י החלפת גוף הפרוצדורה.

לשם כך, נוסיף לשפה ביטוי חדש בשם update-proc-exp.

להלן הדקדוק המגדיר את הביטוי החדש שברצוננו להוסיף:

Expression ::= *(identifier) = Expression

update-proc-exp (id exp)

ביטוי **update-proc-exp** פועל על פרוצדורה שכבר הוגדרה (שמה נתון ע"י id) ומעדכן את גוף הפרוצדורה לביטוי חדש הנתון ע"י exp. במקרה ולא מוגדרת פרוצדורה כזו יש להדפיס הודעת שגיאה מתאימה.

ביטוי **update-proc-exp** אינו מחזיר תשובה (ולכן יש להחזיר ערך פיקטיבי)

המשך השאלה בעמוד הבא

להלן דוגמה :

```
> (run "let p = proc (w) zero?(w)
        in
          begin
            *(p) = -(w,10);
            (p 100)
          end")
(num-val 90)
>
```

בדוגמה זו, מוגדרת פרוצדורה בשם p המקבלת פרמטר w ומחזירה תשובה בוליאנית.
לאחר מכן, גוף הפרוצדורה מעודכן כך שיחזיר מספר.
ולבסוף מופעלת הפרוצדורה p על ארגומנט שערכו 100. ולכן ערכה הסופי של התוכנית הוא
(num-val 90)

דוגמה נוספת:

```
> (run "let p = 200
        in
          begin
            *(p) = -(w,10);
            (p 100)
          end")
interp.scm:114:22: update-proc: p is not a procedure
>
```

המשך השאלה בעמוד הבא

מה עליכם לממש בשאלה זו?

1. **(10 נקודות) עדכון תחביר השפה עם הדקדוק של הביטוי החדש update-proc-exp**

השלימו במחברת הבחינה את קטע הקוד הבא לעדכון דקדוק הביטוי החדש:

```
(expression
      (_____________________________)
 update-proc-exp)
```

2. **(20 נקודות) מימוש התנהגות של הביטוי החדש update-proc-exp**

השלימו במחברת הבחינה את קטע הקוד הבא שאמור להתווסף לפרוצדורה value-of לטיפול בהתנהגות של הביטוי החדש:

```
(update-proc-exp (__________)

  _____________________________________
  _____________________________________


 )
```

שאלה 3 (30 נקודות)

להלן נתונה תוכנית בשפת "ויקישי" (INFERRED):

```
if (proc (x: ?) x zero?(9))
then
   proc (a: ?) a
else
   proc (b: ?) b
```

ברצוננו לקבוע מהו טיפוס התוכנית (אם קיים). לשם כך, עליכם להשתמש באלגוריתם שנלמד בפרק 7 להקשת הטיפוס.

א. (7 נקודות)

הקצו לביטוי הנתון בשאלה ולתתי הביטויים שלו משתני-טיפוס (Type Variables) מתאימים.

ב. (8 נקודות)

חברו משוואות מתאימות לביטוי הנתון ולתתי הביטויים שלו לפי משתני הטיפוס שהקצתם בסעיף א'.

ג. (15 נקודות)

פתרו את המשוואות שהרכבתם בסעיף ב' צעד אחר צעד, תוך שימוש באלגוריתם שנלמד בפרק 7 בדומה למתואר בספר בעמודים 252-258. בסיום פתרון המשוואות רשמו מהו הטיפוס שהוקש עבור התוכנית במידה וקיים.

ב ה צ ל ח ה !