מס' ת.ז. _____

תואר: ראשון\שני

**מבחן בשפות תכנות (234319) מועד ב'**

סמסטר חורף תשע"ב

ד"ר טל כהן, יואב קנטור, יורי משמן, שי מרקנטי

21 במרץ 2012

**הנחיות**

1. נא לענות בגוף השאלון בכתב יד ברור, ולחרוג כמה שפחות מהשטח שסומן לתשובה.
2. משך המבחן: שלוש שעות. לא תינתן הארכת זמן!
3. במבחן זה 6 שאלות (בעמודים 1-14).
4. יש להשתמש בעט כחול או שחור בפתרון המבחן.
5. מותר להשתמש בחומר עזר.
6. לכל שאלה מוקצה מקום לתשובה בגוף המבחן. המקום המוקצה הוא בד"כ גדול בהרבה מהמקום הנדרש.
7. לא לשכוח מס' ת.ז. ומסלול לימודים (תואר ראשון/שני) בראש השאלון וכמו כן מס' ת.ז. בראש כל דף בחינה.
8. בדיקת הבחינה תיערך באופן אנונימי.
9. סך הנקודות בבחינה היינו 100 נקודות.

בהצלחה!

| שאלה | ציון | מתוך |
|---|---|---|
| 1 | | 20 |
| 2 | | 20 |
| 3 | | 20 |
| 4 | | 12 |
| 5 | | 10 |
| 6 | | 18 |

## שאלה 1

בשאלה זו נפתור באמצעות שפת פרולוג את בעיית הקובייה ההונגרית.
הבעיה מוגדרת בצורה הבאה:

- נתונה קובייה, שמימדיה הם 2x2x2 - כלומר, כל פאה מחולקת לארבעה תאים.
- כל תא בקובייה הוא מקבוצת הצבעים: {אדום, כחול, ירוק, צהוב, לבן, כתום}.
- ישנם בדיוק ארבעה תאים מכל צבע.
- פאה תקרא **פתורה**, אם כל ארבעת התאים שבה הם באותו צבע.
- קובייה תקרא **פתורה**, אם כל הפאות בקובייה פתורות.
- הקובייה נתונה במצב התחלתי כלשהו, וניתן לבצע שתי פעולות על מנת לשנות את מצבה:
  - <u>סיבוב עמודה (i)</u> - בפעולה זו מתרחשים השינויים הבאים בקובייה:

1. כל פאה מעבירה את העמודה ה-i לפאה הבאה אחריה בסדר הבא:
   קדמית ← תחתונה ← אחורית ← עליונה ← קדמית
2. אחת הפאות הצדדיות (בהתאם לעמודה אותה מסובבים) מסתובבת סביב צירה ב-90 מעלות נגד כיוון השעון, השניה לא משתנה.

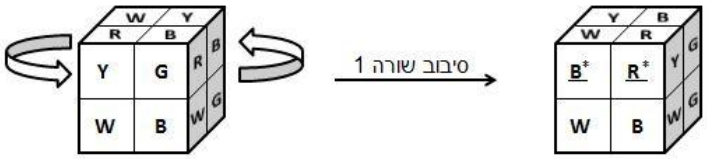
לדוגמא:

(* עמודה זו התקבלה מהפאה האחורית)

  - <u>סיבוב שורה (i)</u> - בפעולה זו מתרחשים השינויים הבאים בקובייה:

1. כל פאה מעבירה את השורה ה-i לפאה הבאה אחריה בסדר הבא:
   קדמית ← ימנית← אחורית ← שמאלית← קדמית
2. אחת הפאות,העליונה או התחתונה (בהתאם לשורה אותה מסובבים) מסתובבת סביב צירה ב-90 מעלות נגד כיוון השעון. השניה לא משתנה.

לדוגמא:




(* שורה זו התקבלה מהפאה השמאלית)

את הקובייה נייצג בצורה הבאה:

- כל פאה תיוצג על ידי רשימה של 2 רשימות באורך 2 כל אחת - המייצגת את רשימת השורות בפאה.
  למשל, את הפאה הקדמית בדוגמא הנ"ל נייצג באמצעות הרשימה:[[yellow,green],[white,blue]].
- כל קובייה תיוצג על ידי רשימה של 6 פאות (לפי הייצוג שהוגדר לעיל), לפי הסדר הבא:
  [Front,Right,Back,Left,Top,Bottom].

לכל אורך השאלה, ניתן להניח את ההנחות הבאות:

- הקלט תמיד תקין, כלומר הרשימות תמיד מכילות איברים מהקבוצה :
  {red, green, blue, yellow, white, orange}
- הקובייה תמיד חוקית - כלומר ישנם בדיוק ארבעה תאים מכל צבע בקובייה.
- המצב ההתחלתי של הקובייה הוא כזה שיש ממנו סדרת פעולות **סופית** שתביא לפתרון.
- בכל סעיף ניתן להשתמש ביחסים שנלמדו בתרגול או ביחסים מסעיפים קודמים, גם אם לא מימשתם אותם.

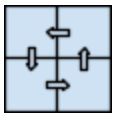
א. (1 נק') כתבו את היחס: solvedFace(F) המתקיים אם ורק אם F הינה רשימה המייצגת פאה **פתורה**.

```
solvedFace([[X,X],[X,X]]).
```

ב. (1 נק') כתבו את היחס: solvedCube(C) המתקיים אם ורק אם C היינה רשימה המייצגת קובייה **פתורה**.

```
solvedCube([Front,Right,Back,Left,Top,Bottom]) :-
solvedFace(Front), solvedFace(Right), solvedFace(Back), solvedFace(Left), solvedFace(Top),
solvedFace(Bottom).
```

ג. (2 נק') כתבו את היחס: rotateFace(F1,F2) המתקיים אם ורק אם F2 היא הפאה המתקבלת מסיבוב הפאה F1 ב-90 מעלות נגד כיוון השעון, כמתואר בדיאגרמה הבאה:



```
rotateFace([[C,A],[D,B]],[[A,B],[C,D]]).
```

ד. (2 נק') כתבו את היחס: shiftRow(N,F1,F2,F3) המתקיים אם ורק אם F3 היינה הפאה F2 אשר הוחלפה בה השורה ה-N'ית עם השורה ה-N'ית מהפאה F1.

```
shiftRow(1,[[A,B],[_,_]],[[_,_],[C,D]],[[A,B],[C,D]]).
shiftRow(2,[[_,_],[C,D]],[[A,B],[_,_]],[[A,B],[C,D]]).
```

ה. (2 נק') כתבו את היחס: shiftColumn(N,F1,F2,F3) המתקיים אם ורק אם F3 היינה הפאה F2 אשר הוחלפה בה העמודה ה-N'ית עם העמודה ה-N'ית מהפאה F1.

```
shiftColumn(1,[[A,_],[C,_]],[[_,B],[_,D]],[[A,B],[C,D]]).
shiftColumn(2,[[_,B],[_,D]],[[A,_],[C,_]],[[A,B],[C,D]]).
```

ו. (4 נק') כתבו את היחס: rotateRow(N,C1,C2) המתקיים אם ורק אם C2 היינה הקובייה המתקבלת מהקובייה C1 ע"י סיבוב השורה ה-N'ית ב-90 מעלות נגד כיוון השעון - כלומר, השורה ה-N'ית בכל פאה מוחלפת בשורה ה-N'ית מהפאה הקודמת לה.

```
rotateRow(1,[Front1,Right1,Back1,Left1,Top1,Bottom],
          [Front2,Right2,Back2,Left2,Top2,Bottom]) :-
    shiftRow(1,Front1,Right1,Right2),
    shiftRow(1,Right1,Back1,Back2),
    shiftRow(1,Back1,Left1,Left2),
    shiftRow(1,Left1,Front1,Front2),
    rotateFace(Top1,Top2).

rotateRow(2,[Front1,Right1,Back1,Left1,Top,Bottom1],
          [Front2,Right2,Back2,Left2,Top,Bottom2]) :-
    shiftRow(2,Front1,Right1,Right2),
    shiftRow(2,Right1,Back1,Back2),
    shiftRow(2,Back1,Left1,Left2),
    shiftRow(2,Left1,Front1,Front2),
    rotateFace(Bottom2, Bottom1).
```

<u>הערה</u>: התקבלה גם תשובה שמכילה את השורה הבאה במקום השורה האחרונה:

```
    rotateFace(Bottom1,Bottom2).
```

ז. (4 נק') כתבו את היחס: rotateColumn(N,C1,C2) המתקיים אם ורק אם C2 הינה הקוביה המתקבלת מהקוביה C1 ע"י סיבוב העמודה ה-N'ית ב-90 מעלות, נגד כיוון השעון - כלומר, העמודה ה-N'ית בכל פאה מוחלפת בעמודה ה-N'ית מהפאה הקודמת לה.

```
rotateColumn(1,[Front1,Right,Back1,Left1,Top1,Bottom1],
                [Front2,Right,Back2,Left2,Top2,Bottom2]) :-
    shiftColumn(1,Front1,Bottom1,Bottom2),
    shiftColumn(1,Bottom1,Back1,Back2),
    shiftColumn(1,Back1,Top1,Top2),
    shiftColumn(1,Top1,Front1,Front2).
    rotateFace(Left2,Left1).
```

<u>הערה</u>: התקבלה גם תשובה שמכילה את השורה הבאה במקום השורה האחרונה:

```
    rotateFace(Left1,Left2).
```

```
rotateColumn(2,[Front1,Right1,Back1,Left,Top1,Bottom1],
                [Front2,Right2,Back2,Left,Top2,Bottom2]) :-
    shiftColumn(2,Front1,Bottom1,Bottom2),
    shiftColumn(2,Bottom1,Back1,Back2),
    shiftColumn(2,Back1,Top1,Top2),
    shiftColumn(2,Top1,Front1,Front2).
    rotateFace(Right1,Right2).
```

ח. (1 נק') כתבו את היחס applyMove(C1,C2) המתקיים אם ורק אם C2 היינה הקובייה המתקבלת ע"י הפעלת פעולה חוקית בודדת על הקובייה C1.

```
applyMove(C1,C2) :- rotateRow(1,C1,C2).
applyMove(C1,C2) :- rotateRow(2,C1,C2).
applyMove(C1,C2) :- rotateColumn(1,C1,C2).
applyMove(C1,C2) :- rotateColumn(2,C1,C2).
```

ט. (3 נק') כתבו את היחס solve(Cube,MovesLeft,Moves) המתקיים אם ורק אם:

- Cube - מצב התחלתי של הקובייה.
- MovesLeft - מס' הצעדים שנותרו לפתרון הבעיה.
- Moves - רשימה של מצבי ביניים חוקיים בדרך לפתרון הבעיה, כך שניתן להגיע ממצב למצב העוקב ברשימה ע"י הפעלת פעולה בודדת.
  אורך הרשימה Moves הוא לכל היותר ערכו הנוכחי של MovesLeft.
  המצב האחרון ב-Moves היינו מצב בו הקובייה פתורה.

הניחו כי Cube ו-MovesLeft היינם אובייקטים קונקרטיים, ואילו Moves היינו משתנה.

```
solve(Cube,MovesLeft,[]) :- solvedCube(Cube), MovesLeft >= 0, !.
```

```
solve(Cube,MovesLeft, [Next|T]) :-
        MovesLeft > 0,
        M2 is MovesLeft - 1,
        applyMove(Cube,Next),
        solve(Next,M2,T).
```

## שאלה 2

בשאלה זו אין להשתמש בפונקציות עזר או בפונקציות שנלמדו בכיתה (למעט האופרטורים על רשימות :: ו @). ניתן להשתמש בפונקציות הנתונות (ראה פירוט בהמשך) ובפונקציות מסעיפים קודמים (גם אם לא פתרתם אותם). בנוסף ניתן להניח שהקלטים חוקיים ואין צורך לבדוק אותם.

הגדרות:

**<u>עץ חיפוש בינארי</u>** הוא עץ בו לכל צומת יש לכל היותר 2 בנים (ימני ושמאלי). בבן הימני ובכל צאצאיו ערכים גדולים מהערך בצומת. בבן השמאלי ובכל צאצאיו ערכים קטנים מהערך בצומת.

**<u>גובה</u>** של עלה הוא 0. גובה של צומת הוא המקסימום של גבהי בניו ועוד 1.

**<u>עץ AVL</u>** הוא עץ חיפוש בינארי מאוזן שבו הפרש גובהם של תת-העצים הבנים של כל צומת (בערך מוחלט) הוא לכל היותר 1.

נגדיר את הטיפוס הבא:

```
datatype avltree =      Leaf
                      | Node of int * int * avltree * avltree;
```

אשר ישמש אותנו למימוש עץ AVL.

ערך מטיפוס זה יכול להיות עלה או צומת פנימי שמאחסן את הגובה שלו, הערך שלו ושני עצי AVL בנים (ע"פ סדר זה).

פונקציות נתונות:

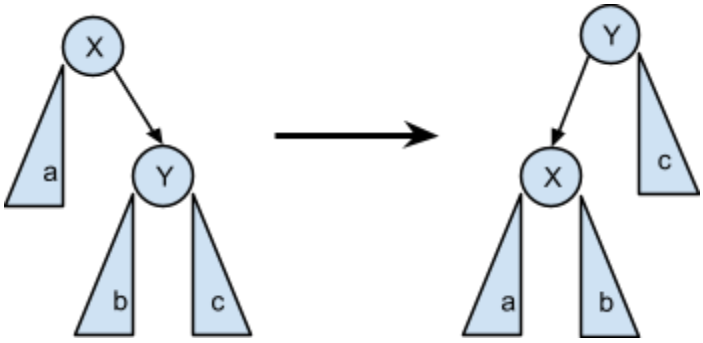
- נתונה הפונקציה **height: avltree -> int** אשר מקבלת עץ AVL ומחזירה את גובהו.
- נתונה הפונקציה **balance_factor: avltree -> int** אשר מקבלת עץ AVL ומחזירה את הפרש הגבהים של בניו. במידה והתקבל עלה היא מחזירה 0.
- נתונה הפונקציה**max: int * int -> int** אשר מקבלת 2 מספרים ומחזיר את הגדול ביניהם.

א. (2 נק') ממשו את הפונקציה createNode: int * avltree * avltree -> avltree אשר מקבלת ערך, תת-עץ שמאלי (עץ avl חוקי), תת-עץ ימני (עץ avl חוקי) ומחזירה צומת עם <u>גובה נכון</u> (ע"פ הגדרה), הערך והבנים שהתקבלו. אין צורך לדאוג לשמירת תכונת החיפוש או איזון העץ.

```
fun createNode(v: int, l: avltree, r: avltree) = Node(1+max(height l, height r), v, l, r);
```

ב. (3 נק') ממשו את הפונקציה rotate_left: avltree -> avltree אשר מקבלת עץ AVL ומבצעת גלגול

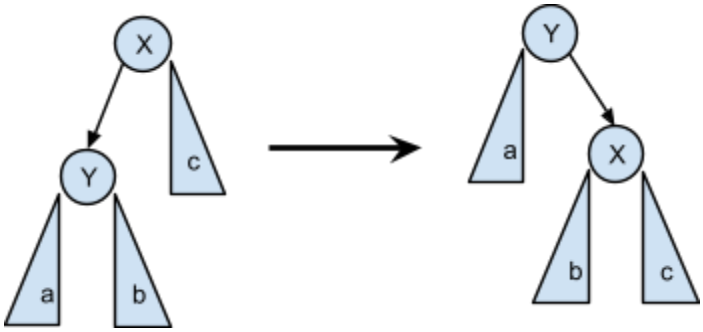
שמאלה באופן הבא:




בשרטוט: X,Y הם הערכים בצמתים ו a,b,c הם תתי עצים.
ז"א הפונקציה מקבלת עץ ואם הוא בעל המבנה המתאים מבצעת גלגול כפי שמתואר בשרטוט.
במידה ולא ניתן לבצע גלגול (בגלל מבנה העץ שהתקבל) היא מחזירה את העץ שהתקבל.

```
fun rotate_left(Node(_, x, a, Node(_, y, b, c))) =
_createNode(y, createNode(x, a, b), c)_____________________________________
_____________________________________________________________________________
_____________________________________________________________________________
| rotate_left(_____t_____) = ____t_____________________________________________
```

ג. (3 נק') ממשו את הפונקציה rotate_right: avltree -> avltree אשר מקבלת עץ AVL ומבצעת גלגול ימינה באופן הבא:




בשרטוט: X,Y הם הערכים בצמתים ו a,b,c הם תתי עצים.
ז"א הפונקציה מקבלת עץ ואם הוא בעל המבנה המתאים מבצעת גלגול כפי שמתואר בשרטוט.
במידה ולא ניתן לבצע גלגול (בגלל מבנה העץ שהתקבל) היא מחזירה את העץ שהתקבל.

```
fun rotate_right(_____Node(_, x, Node(_, y, a, b), c)_____________________) =
____createNode(y, a, createNode(x, b, c)) ___________________________________
_____________________________________________________________________________
_
| rotate_right(_____t_____) = _____t___________________________________________
```

ד. (4 נק') ממשו את הפונקציה balance: avltree -> avltree אשר מקבלת צומת ומאזנת אותו אם יש צורך בכך.
איזון צומת מתבצע לפי החוקיות הבאה:

- אם מקדם האיזון (balance factor) של הצומת הוא 2 ומקדם האיזון של בנו השמאלי הוא 1- מבצעים גלגול שמאלי לבן השמאלי וגלגול ימני לצומת (ע"פ סדר זה).
- אם מקדם האיזון (balance factor) של הצומת הוא 2 ומקדם האיזון של בנו השמאלי אינו 1- מבצעים גלגול ימני לצומת.
- אם מקדם האיזון (balance factor) של הצומת הוא 2- ומקדם האיזון של בנו הימני הוא 1 מבצעים גלגול ימני לבן הימני וגלגול שמאלי לצומת (ע"פ סדר זה).
- אם מקדם האיזון (balance factor) של הצומת הוא 2- ומקדם האיזון של בנו הימני אינו 1 מבצעים גלגול שמאלי לצומת.

הערה: n as Node מאפשר לנו להתייחס לצומת שהתקבל באמצעות הפרמטר n וגם מבצע pattern matching ומאפשר לנו להשתמש בערכיו באמצעות הפרמטרים h,v,l,r

```
fun balance(n as Node(h, v, l, r)) =
  case (balance_factor n , balance_factor l, balance_factor r)
    of (2, ~1,_) => ___rotate_right(createNode(v, rotate_left l, r))___
     | (2,_,_)  => _____rotate_right(n)_____________________________
     | (~2,_,1)  => ___rotate_left (createNode(v, l, rotate_right r))___
     | (~2,_,_)  => ___rotate_left (n)_____________________________
     | _ => _________n_____________________________________________________
| balance(________n_______) =_____n_____________________________________________________
```

ה. (4 נק') ממשו את הפונקציה insert: avltree * int -> avltree אשר מקבלת עץ AVL ומספר ומכניסה את המספר לעץ תוך שמירת תכונת החיפוש והאיזון. אם האיבר כבר נמצא בעץ יש להחזיר את העץ ללא שינוי.
את המספר מכניסים לעלה (תוך שמירת תכונת החיפוש). על מנת לשמור על תכונת האיזון יש לאזן כל צומת במסלול מהעלה עד לשורש העץ (ע"פ סדר זה).

```
fun insert (Leaf, newVal) = _____createNode(newVal, Leaf, Leaf)__________
| insert (n as Node(h, v, l, r), newVal) = ____________________________________________
        if(newValue<v) then
                balance(createNode(v, insert(l, newValue), r))
         else  if(v<newValue) then
                balance(createNode(v, l, insert(r, newValue)))
         else
                n
```

ו. (4 נק') ממשו את הפונקציה sortList: int list -> int list המקבלת רשימת מספרים ללא חזרות וממיינת את הרשימה תוך שימוש בעץ AVL. בסעיף זה מותר לממש פונקציות עזר.

```
fun listToAvltree ([],t)  = t
|         listToAvltree (x::xs,t)  = listToAvltree(xs,insert(t, x));

fun avltreeToList (Leaf)  = []
|         avltreeToList (Node(_, v, l, r))  =  avltreeToList(l)@[v]@avltreeToList(r);

fun sortList(lst) = avltreeToList(listToAvltree(lst, Leaf));
```

## שאלה 3

בהינתן ההגדרה הבאה:

```
exception Error of int;
```

ובהינתן הפונקציה הבאה אשר מדפיסה למסך:

```
fun myPrint (Error i) = let val tmp=print ("Error "^(Int.toString i)^"\n"); in () end;
```

דוגמת הרצה של הפונקציה:

```
- myPrint (Error 100)
Error 100
val it = () : unit
```

מה יודפס במקרים הבאים, כאשר הם הוכנסו אחד אחרי השני למפרש:
במידה ויש שגיאה- יש לציין זאת ולהסביר את סיבת השגיאה. אין צורך לנסח את תגובת המפרש במדויק.
במידה ונזרקת חריגה- יש לציין זאת. לדוגמא: נזרקת חריגה Error 100.

א. (2 נק')

```
fun f 1 = raise (Error 1)
|     f 2 = raise (Error 2)
|     f n = (if (n<4) then raise (Error 3)
            else  (f (n div 2))) handle t => t;
val f = fn : int -> exn
```

---

ב. (2 נק')

```
local
val x = f 100;
val y = f 200;
in
val z = (myPrint x);
end;
Error 3
val z = () : unit
```

---

ג. (3 נק')

```
 ([raise Error 1, raise Error 2, raise Error 3]) handle t => (myPrint t);
_stdIn:25.2-25.73 Error: expression and handler don't agree [tycon mismatch]
  body:         'Z list
  handler range:        unit
  in expression:
    (raise (Error 1)) :: (raise (Error <exp>)) :: (raise <exp>) :: nil
    handle
       t => myPrint t
    | exn => raise exn
```

טיפוס הערך המוחזר ע"י הhandler חייב להתאים לטיפוס הביטוי ממנו יכולה להיזרק החריגה.
טיפוס הביטוי ממנו נזרקת החריגה הוא `a list ולכן ערך החזרה של הhandler חייב להיות רשימה כלשהי (יכול להיות תת טיפוס), לדוגמה: **[myPrint t]**
תתקבל גם התשובה

```
Error 1
val it = [()] : unit list
```

ד. (2.5 נק')

```
val exponent = fn t => if(t<=0) then 1 else (t* exponent(t-1));
stdIn:34.49-34.57 Error: unbound variable or constructor: exponent
```

לקריאה הרקורסיבית חייבים להשתמש בval rec.

ה. (2.5 נק')

```
fun f () = let
datatype err = Error;
in raise Error end;
stdIn:41.10-41.15 Error: argument of raise is not an exception [tycon mismatch]
  raised: ?.err
  in expression:
    raise Error
```

אסור לזרוק ערך שאינו מטיפוס exn.

בסעיפים הבאים נשתמש בפונקציות map, filter, tl, null אשר נלמדו בכתה ובהגדרות:

```
fun upto m n = if (m>n) then [] else m::(upto (m+1) n);
infix o;
fun f o g = fn x=> f(g(x));
```

ו. (3 נק')

```
val a = map (upto 2) (upto 2 5);
val a = [[2],[2,3],[2,3,4],[2,3,4,5]] : int list list
```

ז. (3 נק')

```
map (   ( fn f=> null (f()) )   o   ( fn t=> fn ()=> tl t )   ) a;
val it = [true,false,false,false] : bool list
```

---

ח. (2 נק')

```
map (List.filter  (fn t=>(t mod 2 = 0)) )   a;
val it = [[2],[2],[2,4],[2,4]] : int list list
```

---

## שאלה 4

א. (5 נק') כתבו פונקציה בשם map-read בשפת LISP (כפי שהוצגה בכיתה) שתקבל שני פרמטרים:
"מפה", שהיא רשימה באורך זוגי של ערכים, ומפתח. אל המפה נתייחס כרשימה של מפתח-ערך,
מפתח-ערך. הפונקציה תחזיר את הערך המתאים למפתח זה.

למשל, הביטוי:

```
(map-read '(quote to-be-or-not-to-be
            title hamlet
            author william-shakespeare
            year 1599) title)
```

ישוערך לערך hamlet.
אם המפתח שהועבר כפרמטר השני ל-map-read לא קיים במפה (הפרמטר הראשון), הפונקציה תחזיר ().

```
(defun map-read (map key)
  (cond ((atom map) ())
        ((eq key (car map)) (car (cdr map)))
        ('t (map-read (cdr (cdr map))))))
```

ב. (4 נק') הסבירו מה היה קורה אם היינו כותבים את הדוגמא מהסעיף הקודם ללא הגרש, כך:

```
(map-read (quote to-be-or-not-to-be
           title hamlet
           author william-shakespeare
           year 1599) title)
```

אם לא "מצטטים" את הרשימה, אז זהו ביטוי לשיערוך. האיבר הראשון ברשימהי (האופרטור) הוא quote. כלומר, הביטוי שהתקבל הוא נסיון להפעיל את quote עם יותר מפרמטר אחד, וזו שגיאה.

ג. (3 נק') הפונקציה eval שנדונה בכיתה מקבלת פרמטר e, שהוא הביטוי לשערוך, ופרמטר a, שהוא מפה בין שמות לערכים. הסבירו לשם מה משתמשת eval בפרמטר a.

כדי לשערך ביטוי, אנו זקוקים לערכי המשתנים השונים בזמן השערוך. זהו ההקשר (context) בו משוערך הביטוי. כלומר, a משמש כהקשר עבור e, או במילים אחרות, a מכיל את ה-binding למזהים המופיעים ב-e. אם מופיע בביטוי e מזהה (של משתנה, פונקציה, וכו') שאינו אופרטור פנימי בשפה, ערכו יישלף מתוך a.

## שאלה 5

הטיפוס Address(σ) הוא קבוצת הכתובת בזיכרון של ערך מטיפוס σ.
הטיפוס Pointer(σ) הוא קבוצת הכתובת בזיכרון של ערך מטיפוס σ או nil.
הטיפוס byte מייצג תת-תחום של השלמים, בין 0 ל-255 (כולל).

א. (2 נק') במחשב עם מרחב זיכרון של $2^{32}$ כתובות, מה מספר הערכים האפשריים של Address(byte)?
$2^{32}$ ערכים.

ב. (2 נק') מה מספר הערכים האפשריים של Pointer(byte) באותו מחשב?
$2^{32}+1$ ערכים.

ג. (2 נק') כיצד ניתן לכתוב את הטיפוס Pointer(σ) במונחים של Address(σ), עם אופרטורים על טיפוסים כפי שהוגדרו בכיתה?

Pointer(σ) = Address(σ) + Unit

או:

Pointer(σ) = Address(σ) + {nil}

(חשוב השימוש באופרטור "+" ולא "∪" - מדובר ב-disjoint union בין טיפוסים ולא באיחוד בין קבוצות).

ד. (4 נק') נניח כי זיכרון המחשב הוא strongly typed, כלומר, אם בכתובת מסוימת יש ערך מטיפוס σ, לא ניתן להתייחס אליה כאל כתובת של ערך מטיפוס אחר.
מהם הערכים (אם בכלל) ב-polytype הבא: List(Pointer(σ))?
כתובת לטיפוס אחד אינה כתובת לטיפוס אחר, אך הערך nil אינו כתובת והוא משותף לכל הטיפוסים. לכן

ב-polytype קיימים הערכים:

[], [nil], [nil, nil], [nil, nil, nil], ...

כלומר, רשימה של nil-ים (מכל אורך שהוא) יכולה להיות רשימה של Pointers מכל טיפוס שהוא.

## שאלה 6

שאלה זו באה להדגים את ההבדל בין פולימורפיזם ad-hoc לפולימורפיזם אוניברסלי.

דני רוצה לממש פונקציה בשם StrCat לשרשור מחרוזות בשפת התכנות X, הדומה מאוד לשפת C++.
הפונקציה StrCat מקבלת שני ערכים ומחזירה ערך מסוג string.

ידוע כי בשפת X:

- הטיפוסים int, float, ו-string הם טיפוסים פרימיטיביים.
- אין coercion בין טיפוסים פרימיטיביים.
- ישנן פונקציות המרה בין טיפוסים פרימיטיביים: הפונקציה i2s מקבלת int ומחזירה string, והפונקציה f2s מקבלת float ומחזירה string.
- האופרטור "+" מוגדר בין מחרוזות (מבצע שרשור), בין שלמים (מבצע חיבור שלמים) ובין ערכים מסוג float (מבצע חיבור). צירופים אחרים של האופרטור "+" לא מוגדרים (למשל, מחרוזת ועוד int).

א. (3 נק') על פי הנאמר עד-כה, האם יש overloading כלשהו בשפת X? (כן/לא/לא ניתן לדעת; נמקו!)
כן - האופרטור "+" הוא overloaded עבור שלושה טיפוסים שונים.

דני מעוניין כי StrCat תעבוד עם פרמטרים מכל אחד משלושת הטיפוסים הפרימיטיביים. כלומר, כל הקריאות הבאות אמורות לעבוד:

```
string ss = StrCat("Hello", "World");   // = "HelloWorld"
string si = StrCat("WhenI'm", 64);      // = "WhenI'm64"
string is = StrCat(4, "ScoreAnd7");     // = "4ScoreAnd7"
string fi = StrCat(3.14, 15);           // = "3.1415"
```

וכו' - כל הצירופים האפשריים.

ב. (3 נק') נניח כי מתכנת בשפת X יכול להגדיר overloaded functions. הסבירו כיצד יכול דני להשתמש ביכולת זו כדי להגדיר את StrCat כך שתעבוד באופן הרצוי.
דני צריך להגדיר מספר פונקציות שונות עם אותו השם - אחת עבור כל צירוף אפשרי של טיפוסי פרמטרים.
יש 9 צירופים כאלה (3 טיפוסים עבור הפרמטר הראשון, 3 עבור השני). הקריאות שבדוגמא משתמשות ב-4 מתוך 9 הפונקציות הנדרשות.

דני משתמש בספריית התוכנה Lib1, המגדירה מחלקה חדשה: A. יש בספריה זו גם פונקציה בשם A2s הממירה ערך מסוג A למחרוזת. כעת, הוא רוצה ש-StrCat תוכל לקבל גם ערכים מסוג A כפרמטרים (אחד הפרמטרים, או שניהם; כל צירוף אפשרי עם הטיפוסים הקיימים).

ג. (2 נק') כמה פונקציות חדשות צריך דני להגדיר? ____7_____ (נא לציין מספר, ללא נימוק)
הסבר: A כטיפוס לפרמטר הראשון ושלושה צירופים אפשריים לפרמטר שני; A כטיפוס לפרמטר השני ושלושה צירופים אפשריים לראשון; או ששני הפרמטרים הם מטיפוס A.

כעת החל דני להשתמש גם בספריה Lib2, המגדירה מחלקה בשם B ופונקציה B2s. הוא מעוניין ש-StrCat תוכל לקבל גם ערכים מסוג B (בנוסף לערכים מסוג A); בפרט, הפונקציה אמורה לעבוד עם פרמטר אחד מסוג A ואחר מסוג B, או להיפך, או כל צירוף אפשרי אחר עם הטיפוסים הקודמים.

ד. (2 נק') כמה פונקציות חדשות נוספות צריך דני להגדיר? ____9_____ (נא לציין מספר, ללא נימוק)
גם "8" התקבל, משום שלא צוין בפירוש שיש לתמוך גם בשרשור בין B ל-B (אבל זה צריך להיות מובן מאליו).

דני מתכנן להשתמש גם ב-Lib3, Lib4, וכו', עד Lib42, והבין שהוא בבעיה. הוא גילה שיש יותר מפתרון אחד, ובחר להשתמש בפתרון מבוסס templates. פתרון זה מאפשר לו להגדיר את StrCat פעם אחת (ללא כל overloading), ועבור כל טיפוס נתמך (כלומר int, string, float, A, B, וכו') להגדיר פונקציית עזר יחידה.

ה. (8 נק') ממשו את הפונקציה StrCat תוך שימוש ב-templates:

```
template < typename S, typename T >
string StrCat( S s, T t ) {
  return MakeStr(s) + MakeStr(t);
}
```

ממשו את פונקציית העזר עבור הטיפוס A:

```
string MakeStr(A a) {
  return A2s(a);
}
```

סטודנטים רבים הגדירו מתודה במקום פונקציה, כאיבר חדש במחלקה A. כלומר, הקוד בפונקציה StrCat נראה כך:

```
return s.MakeStr() + t.MakeStr();
```

הבעיה בגישה זו היא שלא ניתן להגדיר מתודה עבור טיפוסים פרימיטיביים כמו int, float וכו' ולכן פתרון זה שגוי וקיבל ניקוד חלקי בלבד.