מבחן מסכם מועד א' ב-"מבוא למדעי המחשב" 202-1-101-1

סמסטר א' תשס"ג
23.1.2003

מר בועז בן-משה
פרופ' מיכאל קודיש
ד"ר חן קיסר
ד"ר יצחק רוזן

משך הבחינה שלוש שעות.
חומר עזר אסור.
אין להשתמש במחשבון.

במבחן זה 4 שאלות.

אנא רשמו את תשובותיכם בדף התשובות בלבד. המחברת שקיבלתם היא **מחברת טיוטה והיא לא תימסר כלל לבדיקה.** בסיום הבחינה נשמור אך ורק את דף התשובות. כל שאר החומר יועבר לגריסה. הקפידו לרשום בדף התשובות גם את מספר הנבחן ומספר החדר שבו אתם נבחנים.

בשאלות התכנות, מספר השורות העומדות לרשותכם בדף התשובות רומז על אורך הקוד הנדרש. הקפידו על כתב יד ברור. תשובות מסורבלות או ארוכות מדי לא יזכו בניקוד מלא. **אין צורך להעתיק את שורות הקוד הנתונות בשאלון לדף התשובות.**

בהצלחה !

<u>שאלה 1 (30 נקודות)</u>

נתונים ממשק המחסנית

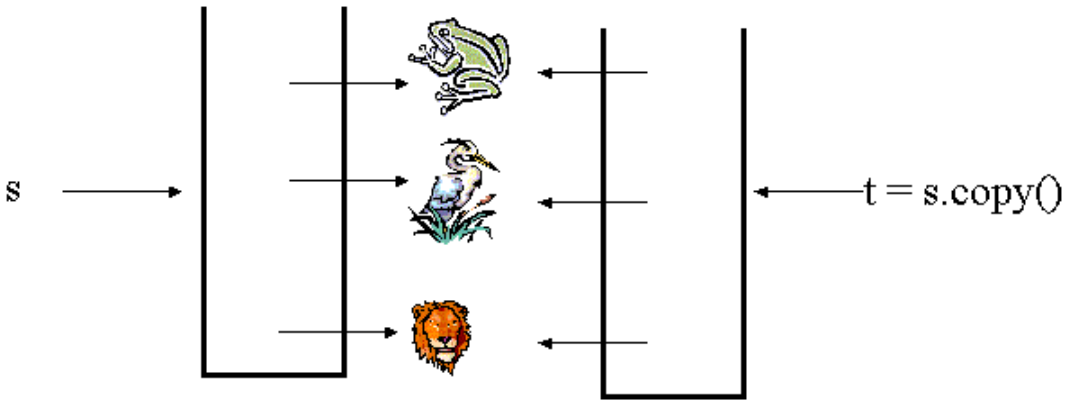
```
public interface Stack{
        void push(Object a);
        Object pop();
        boolean isEmpty();
        Stack copy();
}
```

ומחלקה MyStack המיישמת את השיטות push, pop ו- isEmpty.

<u>סעיף א' (10 נקודות):</u>

השלימו את השיטה Stack copy() במחלקה MyStack תוך שימוש בשיטות האחרות המוגדרות בממשק. על השיטה להחזיר העתק של המחסנית המכיל את אותם העצמים כמו עצם המפתח ובאותו סדר. שימו לב שהשיטה צריכה להחזיר מחסנית <u>שונה מעצם המפתח</u>. כלומר <u>this != copy()</u>.

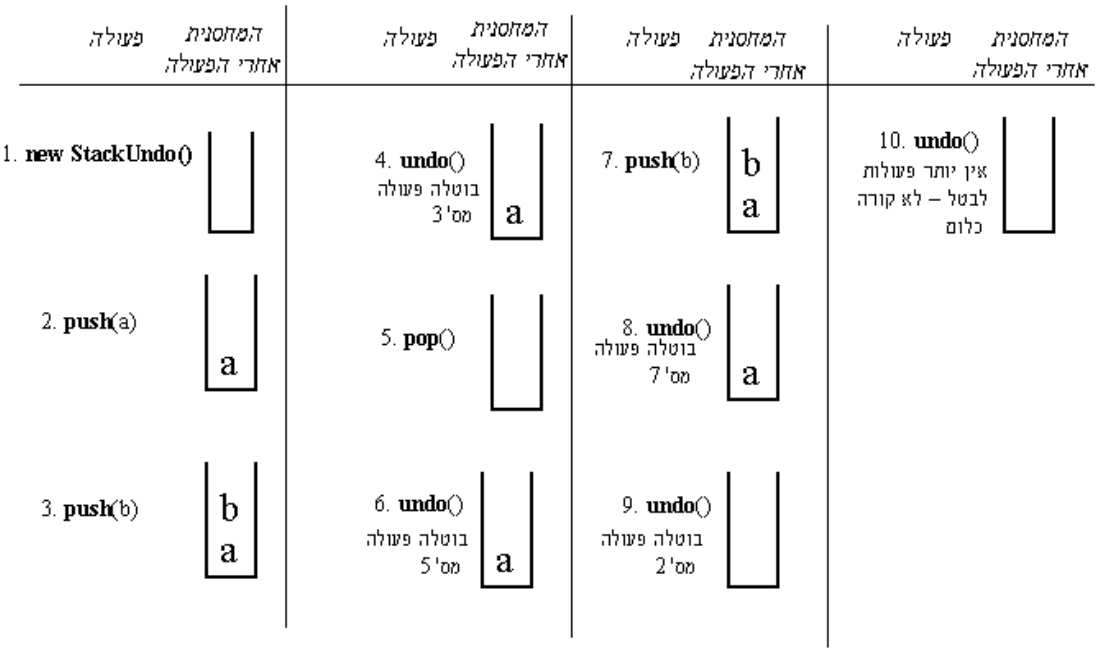
```
public Stack copy() {
        //      א. השלימו בדף התשובות
}
```



<u>סעיפים ב'-ה' (20 נקודות):</u>

מטרתנו בסעיפים הבאים היא ליצור מחלקה חדשה StackUndo המיישמת גם היא את הממשק ובנוסף לכך מיישמת שיטת undo(). שיטה זו מבטלת את הפעולה האחרונה שנעשתה במחסנית (שאינה פעולת undo). שימו לב: מספר הפעמים שניתן לבצע פעולת undo אינה מוגבלת. אם אין יותר שינויים לבטל אזי השיטה undo לא תעשה דבר!

דוגמה:



השלימו את החסר במחלקה

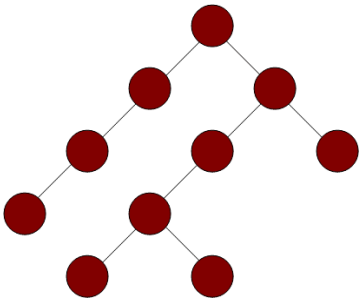
```
class StackUndo implements Stack {
   Stack real, undo;
   public StackUndo() {
         //      ב. השלימו בדף התשובות  (5 נקודות)
   }

   public void push(Object x) {
         //      ג. השלימו בדף התשובות (5 נקודות)
   }

   public Object pop() {
         //      ד. השלימו בדף התשובות (5 נקודות)
   }

   public void undo() {
         //      ה. השלימו בדף התשובות (5 נקודות)
   }
}
```

רמז: השתמשו במחסנית של מחסניות.



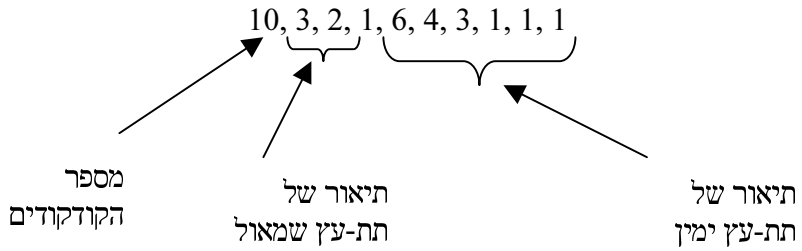
<u>שאלה 2 (25 נקודות)</u>

עץ בינארי מוטה שמאלה (LBT) הוא עץ בינארי שבו אם לצומת יש בן יחיד, אזי הוא בן שמאלי. למשל, העץ שמשמאל הוא עץ מוטה שמאלה. כמו כן, כל עץ בינארי מלא הנו מוטה שמאלה שכן לכל קודקוד בו שאינו עלה, שני בנים.

יהי x עץ מוטה שמאלה. המבנה של x ניתן לתיאור ע"י סדרה של מספרים באופן הבא:

- אם x עץ ריק, המבנה של x מתואר ע"י הסדרה הריקה.
- אחרת, המבנה של x מתואר ע"י הסדרה שבראשה מספר הקדקודים הכולל של x ובזנבה בזה אחר זה הסדרה המתארת את המבנה של תת-העץ השמאלי של x והסדרה המתארת את המבנה של תת-העץ הימני של x, בסדר הזה.

דוגמה - המבנה של העץ המוטה שמאלה, שלמעלה מתואר ע"י סדרת המספרים הבאה (משמאל לימין):



<u>סעיפים א' ו- ב':</u> בתוכנית שלמטה

- המערך a מכיל תיאור של העץ המוטה שמאלה המוצג בדוגמה שלמעלה.
- השיטה numberOfLeaves מקבלת כפרמטר מערך של מספרים המתאר עץ מוטה שמאלה (כדוגמת המערך a) ומחשבת את מספר העלים ב- LBT המתואר על ידיו (מותר להניח קלט תקין).
- השיטה numberOfFullNodes מקבלת אף היא כפרמטר מערך של מספרים המתאר עץ מוטה שמאלה (כדוגמת המערך a) ומחשבת את מספר הקדקודים המלאים (שלהם שני בנים) ב-LBT המתואר על ידיו (מותר להניח קלט תקין).
- בסעיפים אלו **<u>אין</u>** לבנות את העץ המתואר על-ידי המערך.

```
public class LBT {
        public static void main (String [ ] args) {
                int [ ] a = {10, 3, 2, 1, 6, 4, 3, 1, 1, 1};
                System.out.println ("# of leaves = " + numberOfLeaves(a));
                System.out.println ("# of full nodes = " + numberOfFullNodes(a));
        }
        static int numberOfLeaves (int [ ] a) {
                int ans = 0;
                  //   א. השלימו בדף התשובות (5 נקודות)
                return ans;
         }
        static int numberOfFullNodes (int [ ] a) {
                int ans = 0;
                  //   ב. השלימו בדף התשובות (5 נקודות)
                return ans;
        }
}
```

למשל, הרצת התוכנית הנתונה תדפיס על המסך:

```
# of leaves = 4
# of full nodes = 3
```

<u>סעיפים ג'-ט'</u>:

המחלקות שלמטה מגדירות עץ בינארי מוטה שמאלה. במחלקה LBT יש בנאי <u>יחיד</u> המקבל כפרמטר מערך המייצג עץ מוטה שמאלה ובונה את העץ הבינארי שהוא מתאר (אפשר להניח שהמערך אכן מתאר עץ בינארי מוטה שמאלה). השיטה build מסייעת לבנאי זה. היא מקבלת כפרמטרים את המערך a המייצג את ה- LBT, מקום (אינדקס) p על המערך, וקדקוד n קיים בעץ. שיטה זו בונה את הבנים של הקדקוד n לפי המתואר במערך a החל מאינדקס p. השלימו את החסר במחלקה LBT.

```
class TreeNode{
     private Object data;
     private TreeNode left,right;

    public TreeNode(){
      data = null;
      left = null;
      right = null;
    }

    private TreeNode left() {
    //         מחזיר את הבן השמאלי, אין צורך להשלים
    }

    private TreeNode right(){
    //           מחזיר את הבן הימני, אין צורך להשלים
    }

    private void setLeft(TreeNode l) {
    //          קובע את הבן השמאלי, אין צורך להשלים
    }

    private void setRight()(TreeNode r){
    //            קובע את הבן הימני, אין צורך להשלים
    }

   // שיטות נוספות
  }
}
```

```
class LBT{

    private TreeNode root;
    //         משתנים ושיטות נוספות
    //

    public LBT(int[] a) {
     // Assume a != null
      if (a.length == 0) {
       //(1 נק') השלימו בדף התשובות ג.
      }
      else {
            root = new TreeNode();
            build(a,0,root);
      }
    }

    private void build(int[] a,
                       int p,
                        TreeNode n);
     if (/*(3 נק') השלימו בדף התשובות ד.*/) {
          // טיפול במקרה שאין בן שמאלי
          // (1 נק') השלימו בדף התשובות ה.
      }
      else {
          // בניית תת העץ השמאלי
          // (3 נק') השלימו בדף התשובות ו.

      }
     if (/*(3 נק') השלימו בדף התשובות ז.*/) {
          // טיפול במקרה שאין בן הימני
          // (1 נק') השלימו בדף התשובות ח.
      }
      else {
          // בניית תת העץ הימני
          // (3 נק') השלימו בדף התשובות ט.

      }

}
```

שאלה 3 (25 נקודות)

התבוננו במחלקה IntegerList הנתונה שמגדירה רשימה של חוליות המכילות עצמים מסוג Integer.
המחלקה Link אשר מגדירה חוליה נתונה בהמשך.

```
public class IntegerList{

  protected Link first;

  public IntegerList(){
        first = new Link("stam", null);
  }
  public IntegerList(Link sentinel){
        first = sentinel;
  }
  public IntegerList(Integer data, IntegerList list){
    first = new Link("stam", new Link(data,list.first.getNext()));
  }

  public boolean isEmpty(){
        // checks if the list is empty
        return first.getNext() == null;
  }
  public Integer head(){
        // returns the data in the first (real) element of the
        // list. "this" cannot be isEmpty()
        return (Integer) (first.getNext().getData());
  }
  public IntegerList tail() {
        // returns the tail of the list
        // "this" cannot be (isEmpty())
        return new IntegerList(first.getNext());
  }
}// class IntegerList
```

חשוב להבחין שבהעדר שיטה במחלקה הנתונה להוספת איבר לרשימה, הדרך לבצע זאת היא בעזרת הבונה המודגש. לדוגמה, הקוד הבא (בתוספת שיטת toString מתאימה)

```
IntegerList list1 = new IntegerList();
IntegerList list2 = new IntegerList(new Integer(7),list1);
IntegerList list3 = new IntegerList(new Integer(8),list2);
IntegerList list4 = new IntegerList(new Integer(9),list3);

System.out.println("list1  " + list1);    System.out.println("list2  " + list2);
System.out.println("list3  " + list3);    System.out.println("list4  " + list4);
```

ידפיס את השורות הבאות:

```
list1
list2  7
list3  8, 7
list4  9, 8, 7
```

```
public class Link{
    private Object data;
    private Link next;

    Link(Object data, Link next){
         setData(data);  setNext(next);
    }

    public Object getData(){return data; }
    public Link   getNext(){return next; }
    public void   setData(Object a){data = a; }
    public void   setNext(Link a){next = a; }

    public void addNext(Object addMe) {
        this.setNext(new Link(addMe,this.getNext()));
    }
 }// Link
```

<u>סעיפים א-ב:</u>

השלימו במחלקה IntegerList שיטה סטטית

```
public static IntegerList[] partition (Integer pivot, IntegerList list){
    IntegerList[] pair;
    if (list.isEmpty()){
      pair = new IntegerList[2];
          //                (6 נקודות) א. השלימו בדף התשובות
          //
    }
    else{
          //
          //              (6 נקודות) ב. השלימו בדף התשובות
          //
    }
    return pair;
}
```

אשר מקבלת "ציר" pivot שהנו מספר שלם ורשימה list של מספרים שלמים ומחלקת את הרשימה לשתי רשימות אותן היא מחזירה כמערך מסוג IntegerList[ ]. הרשימות המוחזרות מכילות הראשונה את איברי הרשימה list שקטנים מ-pivot והשניה את אלו שאינם קטנים מ-pivot. שיטה זו לא משנה את הרשימה list. אפשר להניח שהפרמטרים אינם null.

בהמשך לדוגמה הקודמת, הקוד הבא

```
IntegerList[] pair = partition(new Integer(8),list4);
System.out.println("less  " + pair[0]);
System.out.println("greater equal  " + pair[1]);
System.out.println("list4  " + list4);
```

ידפיס את השורות הבאות

```
less  7
greater equal  9, 8
list4  9, 8, 7
```

<u>סעיף ג (6 נקודות)</u>:
השלימו במחלקה IntegerList את **ה**שיטה

```
public IntegerList append(IntegerList list){
        if (isEmpty())
            return list;
        else{
            return      //... ג. השלימו בדף התשובות
        }
}
```

אשר מקבלת רשימה list ומחזירה רשימה שמכילה קודם את איברי עצם המפתח ואחר כך את איברי הרשימה list. שיטה זו לא משנה את עצם המפתח או את הרשימה list. אפשר להניח שהפרמטר אינו null.

בהמשך לדוגמה הקודמת, הקוד הבא

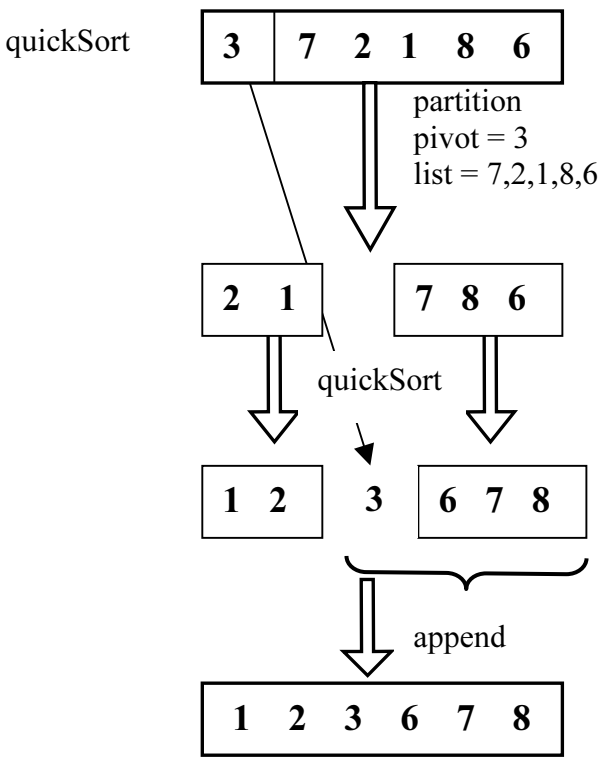
```
System.out.println("list3+list3   " + list3.append(list3));
System.out.println("list3   " + list3);
```

ידפיס את השורות הבאות

```
list3+list3   8, 7, 8, 7
list3 8, 7
```

<u>סעיף ד (7 נקודות):</u>

התרשים משמאל מדגים אלגוריתם למיון שנקרא "מיון מהיר" (quicksort).
בהנתן רשימה לא ריקה list משתמשים באיבר הראשון בה כציר (pivot) ומפצלים את שאר איברי הרשימה לאלו שקטנים מ-pivot ולאלו שאינם. לאחר שממיינים את הרשימות שמתקבלות (ברקורסיה) ניתן לחבר את התוצאות לרשימה ממוינת אחת אותה מחזירה הקריאה ל- quickSort. אפשר להניח שהפרמטרים אינם null.



השלימו את השיטה הסטטית הרקורסיבית הבאה:

```
public static IntegerList quickSort(IntegerList list){

    if (list.isEmpty())
       return list;
    else{
          //
          //   ד. השלימו בדף התשובות
          //
    }
}
```

<u>שאלה 4 (20 נקודות)</u>

<u>סעיף א (10 נק'):</u>

השיטה String[] gray(int n) מקבלת מספר שלם n גדול או שווה ל- 0 (מותר להניח קלט תקין) ומחזירה מערך של כל 2^n המחרוזות באורך n המורכבות מאפסים ואחדים. סדר המחרוזות הוא כזה שכל שתי מחרוזות סמוכות (מעגלית) נבדלות בדיוק בספרה אחת. סידור כזה נקרא *"קוד gray n ביטים"*.

לדוגמה, המערכים הבאים מהווים קוד gray n ביטים.

n = 0 {""}
n = 1 {"0","1"}
n = 2 {"00","01","11","10"}
n = 3 {"000","001","011","010","110","111","101","100"}

למשל עבור n = 3, שתי המחרוזות הראשונות ("000" ו- "001") ושתי המחרוזות האחרונות ("101" ו- "100") נבדלות רק בספרה השלישית. המחרוזות השניה והשלישית מההתחלה ("001" ו- "011") ומהסוף ("101" ו- "111") נבדלות רק בספרה השניה. שימו לב שגם המחרוזת הראשונה והאחרונה נבדלות רק בספרה אחת, שכן התכונה מתקיימת "מעגלית".

השלימו את השיטה.

```
Public static String[] gray(int n) {
      String[] answer;
      if (n==0) {
            answer = new String[1];
            answer[0] = "";
      }
      else {
      //
      //                  השלימו בדף התשובות
      //
      }
      return answer;
}
```

<u>סעיף ב (10 נק'):</u>

השלימו את השיטה boolean one(String s1, String s2) אשר מקבלת שתי מחרוזות ומחזירה ערך true אם ורק אם הן שוות בתוויהם פרט למיקום אחד בדיוק (שם התווים שבהן שונים).

למשל הקריאות הבאות בצד שמאל יחזירו את הערכים בצד ימין

one("001","011")	true
one("011","011")	false
one("000","011")	false
one("011","0110")	false

```
public static boolean one(String s1, String s2) {
      if ((s1 != null & s2 != null) &&
          (s1.length() > 0 & s2.length() > 0)){
      //
      //                  השלימו בדף התשובות
      //
      }
      else return false;
}
```

על תשובה רקורסיבית תוכלו לקבל עד 10 נק'.
על תשובה שאינה רקורסיבית תוכלו לקבל עד 5 נק'.

להזכירכם, במחלקה String קיימות השיטות:

- char charAt(int i) - מחזירה את התו במקום ה-i במחרוזת (עצם המפתח).
- String substring(int i) - מחזירה את הסיפא המתחילה במקום ה-i במחרוזת (עצם המפתח).