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

סמסטר א' תשס"ב
1.3.2002


פרופ' אורי אברהם
פרופ' דניאל ברנד
ד"ר שמואל ספרוני
פרופ' מיכאל קודיש
ד"ר חן קיסר


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

במבחן זה 7 שאלות, בניקוד המסתכם ב- 110 נקודות. ענו על כל השאלות.

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.**

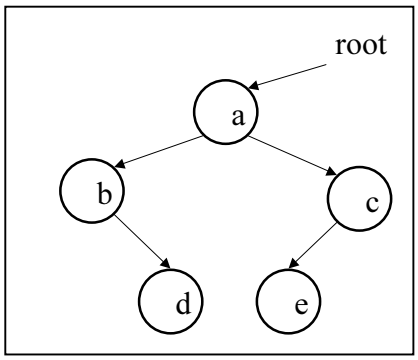
**בהצלחה !**

**שאלה 1 (20 נקודות)**

נתונה התוכנית הבאה אשר מקבלת עץ בינארי ומחזירה תור (המחלקה QueueAsList מממשת את ממשק התור).

```
public static Queue preInPost(BinaryTree t){
    Queue q;
    if (t==null) q = null;
    else{q = new QueueAsList( );
         preInPost(t.root,q);
    }
    return q;
}
private static void preInPost(BinaryNode n, Queue q){
   if ( n != null ){
      q.enqueue(n.get_data( ));
      preInPost(n.get_left( ),q);
      q.enqueue(n.get_data( ));
      preInPost(n.get_right( ),q);
      q.enqueue(n.get_data( ));
   }
}
```

לדוגמה, אם `tree` עץ בינארי כמתואר בציור הבא:




**אזי הקריאה preInPost(tree) תיצור את התור Q שמכיל את האיברים הבאים (משמאל לימין)**

a b b d d d b a c e e e c c a

(הכוונה היא ש- `tree.root.data = "a"` ,`tree.root.left` פונה לתת-העץ ששורשו מסומן ב-b, `tree.root.right` פונה לתת-העץ ששורשו מסומן ב-c וכדומה).

בשאלה זו עליכם להשלים את השיטה buildTree(Queue q), אשר מקבלת תור q ומחזירה עץ בינארי.

מטרת השיטה היא זו: עבור כל עץ בינארי $tree_1$, אם נפעיל את preInPost($tree_1$) שתוארה לעיל ונקבל את התור q, אזי הקריאה buildTree(q) תחזיר עץ $tree_2$ השווה (equals) ל-$tree_1$. למשל בדוגמה שתוארה לעיל הפעלת buildTree על התור **Q** תחזיר עץ השווה לזה המתואר בציור.

על מנת לפשט את המשימה תוכלו להניח שכל קדקוד בעץ tree מכיל data שאינו null ושונה משאר הקדקודים.

רמז: השיטה buildTree תסמן את סוף התור במחרוזת "$" ואחר כך תקרא לשיטה הרקורסיבית build_until(Queue q, Object until) אשר מורידה עצמים מהתור על מנת לבנות את העץ עד שהיא נתקלת בעצם until. בקריאה הראשונה היא תפעל עד שתגיע לסימן "$". בפעמים הבאות היא תפעל עד שהיא מוצאת פעם נוספת את שורש העץ שהיא בונה. תוכלו להניח שהעצם "$" לא מופיע בעץ.

```
public static BinaryTree build(Queue q){
   BinaryTree t;
   if (q==null) t = null;
   else{
      t = new BinaryTree( );
      q.enqueue("$");
      t.root = build_until(q, "$");
   }
   return t;
}
```

```
private static BinaryNode build_until(Queue q, Object until){

                    השלם

}
```

להזכירכם, הבונה במחלקה BinaryTree נראה כך:

```
BinaryTree(){root = null;}
```

והבונה במחלקה BinaryNode נראה כך:

```
BinaryNode(Object data, BinaryNode left, BinaryNode right){
      this.data = data;
      this.left = left;
      this.right = right;
}
```

## שאלה 2 (20 נקודות)

שיטה מקובלת לכתיבת ביטויים חשבוניים היא infix שבה יש צורך בכללי קדימויות וסוגריים.
בכתה למדנו לכתוב ולחשב ביטויים הכתובים בשיטת postfix וראינו שכתיבה בשיטה זו מאפשרת כתיבת ביטויים חשבוניים ללא צורך בכללי קדימויות וסוגריים. שיטה שלישית לכתיבת ביטויים חשבוניים היא prefix שגם היא אינה דורשת סוגריים וכללי קדימות. בביטוי prefix סימן הפעולה (האופרטור) מופיע לפני הביטויים עליהם הוא פועל (אופרנדים). סימני הפעולה מופיעים בביטוי בסדר שבו יבוצעו.

ניתן לחשב את הערך של ביטוי prefix באופן רקורסיבי:

1. ביטויים המכילים אך ורק מספר בודד, ערכם כערך המספר. לדוגמה הביטוי "8" ערכו שמונה.
2. ביטויים אחרים חייבים להתחיל בסימן פעולה ואחריו שני תתי-ביטוי. מחשבים תחילה את ערכי תתי-הביטוי ואחר כך מבצעים את הפעולה המתאימה

דוגמאות:

א. "3"
זהו ביטוי שערכו שלוש
(הגרשיים אינם חלק מן הביטוי ומופיעים רק כדי להפריד בין הביטוי לטקסט המתאר אותו.)

ב. "+ 1 * 2 3"
זהו ביטוי חיבור של שני תת-הביטויים "1" ו- "* 2 3" . ראשית נחשב את ערכי תתי-הביטוי (אחד ושש) ואחר כך נבצע את פעולת החבור.

ג. "* + 4 1 + 2 3"
זהו ביטוי כפל של שני תתי-הביטוי "+ 4 1" ו- "+ 2 3" . ראשית נחשב את ערכי תתי-הביטוי (במקרה זה שניהם חמש) ואחר כך נבצע את פעולת הכפל.

המחלקה PrefixTerm מיצגת ביטויי prefix, הערך של ביטוי מחושב על ידי השיטה evaluate אשר מקבלת StringTokenizer העובר על אברי הביטוי. לשם הפשטות נניח שהביטויים מכילים רק מספרים שלמים וסימני חיבור וכפל. המספרים והסימנים מופרדים על ידי רווחים.
לדוגמה, השיטה main הבאה במחלקה PrefixTerm

```
public static void main(String[] args) {
     PrefixTerm term = new PrefixTerm();
     System.out.println(term.evaluate(new StringTokenizer("* + 1 2 3")));
     System.out.println(term.evaluate(new StringTokenizer("+ 1 * 2 3")));
     System.out.println(term.evaluate(new StringTokenizer("+ 1 * 2 3 3")));
     System.out.println(term.evaluate(new StringTokenizer("+ + 1 * 2 3")));
}
```

מדפיסה את השורות הבאות:

```
9
7
null
null
```

(הערה: `null` מציין ביטוי בלתי תקין. למשל בעל סימן או מספר מיותרים.)

**תזכורת:**

א. במחלקה **StringTokenizer**

```
StringTokenizer(String str)   // Constructs a string tokenizer for the specified string.
boolean hasMoreTokens()       // Tests if there are more tokens available from this tokenizer's
                                 string.
String nextToken()            // Returns the next token from this string tokenizer.
```

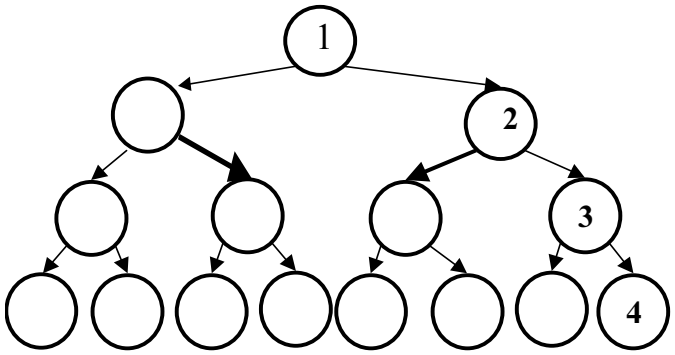
ב. במחלקה **Integer**

```
Integer(int value)            // Constructs a newly allocated Integer object that represents
                              //  the primitive int argument.
Integer(String s)             //  Constructs a newly allocated Integer object that represents
                              //  the value represented by the string.
int intValue()                //  Returns the value of this Integer as an int.
```

```
import java.util.StringTokenizer;
public class PrefixTerm {

// מחלקה המייצגת ביטוי פרפיקס כלשהו
    public PrefixTerm() {} // בונה ריק (בררת מחדל)
    public Integer evaluate(StringTokenizer tokens) {
        Integer out;
        if (! tokens.hasMoreTokens()) return null;
        out = (string2term(tokens.nextToken())).evaluate(tokens);
        if (tokens.hasMoreTokens()) return null;
        return out;
    }
    public PrefixTerm string2term (String s) {
        if ("+".equals(s)) return new PrefixPlus();
        if ("*".equals(s)) return new PrefixTimes ();
        א. השלימו ( נקודה 1)
    }
}

// מחלקה המייצגת ביטוי פרפיקס שבו מחרוזת אחת שהיא מספר
class PrefixInt  extends PrefixTerm {
    Integer Value;
    ב. השלימו בונה כך שהתוכנית תעבוד (2 נקודות)
     public Integer evaluate(StringTokenizer tokens) {
       return value;
     }
}

// מחלקה המייצגת ביטוי פרפיקס המתחיל בסימן פעולה כלשהו
abstract class PrefixOperator extends PrefixTerm {
      ג. השלימו בונה כך שהתוכנית תעבוד (2 נקודות)
     public Integer evaluate(StringTokenizer tokens) {
                        ד. השלימו (13 נקודות)
     }
     abstract Integer calculate(Integer a, Integer b);
}

// מחלקה המיצגת ביטוי פרפיקס המתחיל בסימן הפעולה "+"
class  PrefixPlus extends  PrefixOperator{
    Integer calculate(Integer a, Integer b) {
        ה. השלימו (נקודה 1)
    }
}

// מחלקה המיצגת ביטוי פרפיקס המתחיל בסימן הפעולה "*"
class  PrefixTimes extends  PrefixOperator{
    Integer calculate(Integer a, Integer b) {
      ו. השלימו (נקודה 1)
    }
}
```

**<u>שאלה 3</u>** (15 נקודות)

בהינתן רשימת מספרים ללא חזרות, ניתן לצייר עץ חיפוש בינארי כך שהרשימה תתקבל ממנו על ידי מעבר **preorder.**
לדוגמה: בהינתן הרשימה 1,2,3,4 נוכל לצייר את העץ:




בהינתן הרשימה 20,10,5,15,12,25,22,30,27 (משמאל לימין) שהתקבלה על ידי מעבר preorder על עץ חיפוש בינארי.

א. ציירו עץ חיפוש בינארי מתאים (3 נקודות).
ב. מספר העצים האפשריים כתשובה לסעיף א' הוא (4 נקודות):
   a. אחד
   b. יותר מאחד
ג. אילו הייתה הרשימה מתקבלת על ידי מעבר inorder, מספר העצים האפשריים היה (4 נקודות):
   a. אין עץ אפשרי
   b. אחד
   c. יותר מאחד
ד. אילו הייתה הרשימה מתקבלת על ידי מעבר postorder, מספר העצים האפשריים היה (4 נקודות):
   a. אין עץ אפשרי
   b. אחד
   c. יותר מאחד

**<u>שאלה 4 (20 נקודות)</u>**

נתאר כאן הצעה למימוש אלגוריתמי מיון.
בהינתן סדרה של מספרים באורך n :

$$a_0, a_1, \ldots, a_{n-1}$$

נסתכל באוסף הזוגות

$$P_n = \{ (i,j): 0 \le i < j < n \}$$

ונסדר אותו בסדר קבוע כלשהוא A.
למשל, עבור $n = 3$ סדור אפשרי של $P_3$ יהיה

$$A = (0,1), (0,2), (1,2)$$

האלגוריתם SORT התלוי ב- A עובר על פני כל זוגות A בסדרם, ולכל זוג $(i,j)$ מחליף את $a_i$ ו- $a_j$ אם $a_j < a_i$. בהנחה שסדרת המספרים ניתנת על ידי מערך a אזי פעולה זו של השוואה נעשית על ידי קריאה לפונקציה compare_and_swap(a, i, j) המתוארת להלן:

```
static void compare_and_swap(int[] a, int i, int j){
  int tmp;
  if ( 0<=i & i<j & j<a.length && a[i]>a[j])
        {tmp=a[i]; a[i]=a[j]; a[j]=tmp;}
} //compare_and_swap
```

בשאלה זו נתייחס לשתי השיטות הבאות שהן גרסאות של שיטות המיון המכונות selection sort ו- insersion sort

```
// At the j'th step selects the j'th largest
// element, and puts it in its correct position  in a.
    static void selection_sort(int[] array){
        for (int j= array.length-1; j > 0; j = j-1)
            for (int i=0; i<j; i=i+1)
                compare_and_swap(array,i,j);
    }// selection_sort

// At the j'th step insert a[j] to its correct
// position among a[0]. . .a[j-1] (which are already
// sorted).
    static void insertion_sort(int [] array){
        for (int j = 1; j < array.length; j = j + 1)
            for (int i = j; i>0; i = i - 1)
                compare_and_swap(array,i-1,i);
    }// insertion_sort
```

ענו על שתי השאלות הבאות:

**סעיף א.** לכל $n \ge 0$ קיים סידור A של $P_n$ כך שהאלגוריתם SORT התלוי ב- A מבצע בדיוק את אותן הפעולות כמו הגרסה של selection sort המתוארת לעיל .

a. לא, אין זה כך.
b. אכן כן. והנה הסדור A המתאים עבור המקרה $n = 4$ .

**סעיף ב.** לכל $n \ge 0$ קיים סידור A של $P_n$ כך שהאלגוריתם SORT התלוי ב- A מבצע בדיוק את אותן הפעולות כמו הגרסה של insersion sort המתוארת לעיל .

a. לא, אין זה כך.
b. אכן כן. והנה הסדור A המתאים עבור המקרה $n = 4$ .

**<u>שאלה 5 (10 נקודות)</u>**

שאלה זו מתייחסת לאלגוריתמי המיון SORT שתוארו בשאלה הקודמת. נאמר שאלגוריתם SORT התלוי ב- A פועל כהלכה אם הוא פועל נכון עבור כל קלט.

מה נכון באפשרויות הבאות

1. עבור כל n טבעי ועבור כל סידור A של $P_n$ האלגוריתם SORT התלוי ב A פועל כהלכה.

2. עבור כל n טבעי האלגוריתם merge sort הוא מימוש (או ניתן לראותו ככזה) של SORT עבור סידור מסוים A של $P_n$ . (כמובן שעקב הרקורסיה, מימוש זה הוא עקיף, אך לאחר שנבדוק את ההשוואות, נגלה שאלו תואמות בדיוק את פעולות של SORT עבור ה- A.)

3. לכל מספר טבעי $n\geq 3$ גדול או שווה 3 יש סדרה $a_0,a_1, \ldots ,a_{n-1}$ של מספרים וסידור A של $P_n$, עבורם האלגוריתם SORT התלוי ב- A נכשל.

4. בין אם האלגוריתם SORT התלוי ב- A פועל כהלכה ובין אם לאו, כל הפעלה שלו על קלט נתון מסדרת לפחות את האבר הגדול ביותר ואת האבר הקטן ביותר במקומם (בסוף ובהתחלה).

5. בין אם האלגוריתם SORT התלוי ב- A פועל כהלכה ובין אם לאו, כל הפעלה שלו על קלט נתון מגדילה את מספר הזוגות העומדים בסדרם הנכון (אלא אם כן הסדרה מסודרת לגמרי), ועל כן אם נפעיל את האלגוריתם מספיק פעמים נקבל תמיד מיון של סדרת הקלט.

6. אף לא אחת מהתשובות לעיל אינה נכונה.

**שאלה 6 (10 נקודות)**

בחן את שתי השיטות הבאות

```
static void randomPermutation1(int n)
{
   int location, x;
   for (location = 0; location < n; location++)
   {
      x = (int)( n * Math.random() ) + 1;
      System.out.print( x + " ");
   }
} //method randomPermutation1

static void randomPermutation2(int n)
{
   int location, number;
   boolean success;
   int[] permutation = new int[n];
   for (location = 0; location < n; location++)
      permutation[location] = 0;
   for (number = 1; number <= n; number++)
   {
      success = false;

      while (!success)
      {
         location = (int) (n * Math.random());
         if (permutation[location] == 0)
         {
            permutation[location] = number;
            success = true;
         }
      }
   }
   for (location = 0; location < n; location++)
      System.out.print(permutation[location] + " ");
} //method randomPermutation2
```

הערה: " " מיצג מחרוזת שבה תו רווח יחיד.

מטרת שיטות אלו (**randomPermutation1** ו- **randomPermutation2**) היא להדפיס תמורה אקראית ( random permutation) של המספרים 1,2, ... , n (וכן רווחים ביניהם). תמורה היא סידור ללא חזרות של כל המספרים מ-1 עד n.

**לדוגמה**: עבור n = 5 פלט **אפשרי** יכול להיות 2 5 3 1 4 .
נאמר שהשיטה עובדת כהלכה אם לכל n טבעי גדול מאפס היא אכן מדפיסה תמורה של המספרים 1,2, ... ,n ולכל אחת מ- n! התמורות האפשריות אותו סיכוי להתקבל.

כזכור, השיטה Math.random() מחזירה ערך אקראי בין 0 (כולל) ל-1 (לא כולל).

מה נכון מהטענות הבאות?

א. שתי השיטות פועלות כהלכה עבור כל ערך קלט n כנדרש. עם זאת השיטה **randomPermutation1** עדיפה משום שפעולתה מהירה יותר.

ב. השיטה **randomPermutation1** אינה פועלת כהלכה משום שהחישוב עלול להביא לחזרה על ערכי המשתנה x, ואז הפלט יכיל חזרות. אם נתעלם מאפשרות זו (שהיא נדירה מאוד) נקבל פעולה נכונה של השיטה. כמו כן השיטה **randomPermutation2** פועלת כשורה.

ג. עבור כל אחת משתי השיטות, בהינתן n, מספר צעדי החישוב הוא קבוע.

ד. השיטה **randomPermutation1** יכולה להדפיס לפעמים פלט נכון, אך לפעמים תדפיס פלט שגוי. השיטה **randomPermutation2** פועלת כהלכה תמיד, אך מספר הצעדים הנדרש לפעולתה אינו נקבע באופן חד-ערכי על ידי n והוא תלוי במספרים שהוחזרו על ידי Math.random() .

ה. הבעיה בשיטה **randomPermutation1** היא בכך שככל שנתקדם בבניית הסדרה כך יגבר הסיכוי לחזרות.

ו. נניח כעת ש- n גדול מאד. השיטה **randomPermutation2** מאיטה את קצבה ככל שהיא מתקדמת בפעולתה. זאת אומרת שעבור מציאת האבר הראשון בפלט יש צורך בקריאה אחת בלבד ל- Math.random() . עבור האבר השני הפלט יצריך בדרך כלל קריאה אחת בלבד. אך עבור ערכים מאוחרים, למשל עבור הערך האחרון, יש קרוב לודאי צורך במספר רב של קריאות ל- Math.random() עד למציאתו.

ז. אף לא אחת מהתשובות לעיל אינה נכונה.

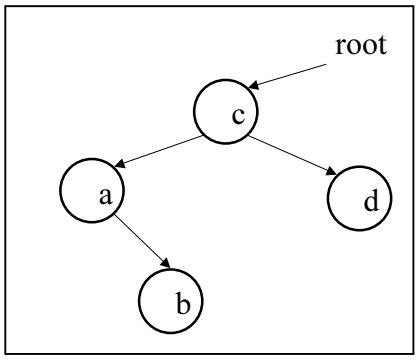
**<u>שאלה 7 (15 נקודות)</u>**

למחלקה BinaryNode הוספו השיטות numberOfNodes() וכן what(int coeff). השיטה numberOfNodes() מחזירה את מספר הקדקודים אשר בעץ ששורשו הוא הקודקוד הנתון. השיטה what(int coeff) נתונה להלן:

```
int what(int coeff)
{
   int result = 0;
   if (left != null)
   {
      result = result +left.what(coeff);
      coeff  = coeff + left.numberOfNodes();
   }
   result = result + coeff * ((Integer) data).intValue();
   if (right != null)
      result = result + right.what(coeff+1);
   return result;
} //method what
```

ענו על שתי השאלות הבאות

א. ( 5 נקודות) יהי T עץ החיפוש הבינארי המתואר בציור. כאשר a<b<c<d מספרים שלמים.




מה תוצאת ההדפסה הבאה עבור קלט x מטפוס int?

```
System.out.println( T.root.what(x) );
```

יש לרשום נוסחה התלויה ב- x, a, b, c, d.

ב. (10 נקודות) **מה תדפיס השורה הבאה?**

```
System.out. println(T.root.what(1));
```

כאשר T הוא עץ חיפוש בינארי כלשהו המורכב מקדקדים במחלקה BinaryNode ואשר מכיל בדיוק n קדקודים בעלי ערכים שלמים מאחד עד n .