מעבדה 6. נושא: עץ ביטוי

תאריך הגשה: 14.12.2019 בשעה 23:00 (בזוגות)

יש לקרוא היטב לפני תחילת העבודה !

מבוא:

במעבדה הנוכחית נלמד "שיטה פולנית" לכתיבת ביטויים אריתמטיים ונממש אותה בעזרת עץ בינארי הנקרא עץ ביטוי.

תיאור:

הגדרה 1: שיטת infix

בדרך כלל, נהוג לכתוב ביטוי אריתמטי בצורה הנקראת **infix form:**

num op num

כאשר

num– מספר

op– פעולה (+,-,*,/)

למשל, הביטוים הבאים הם ביטויי infix :

((31 * 1) + (21 / 2))

((1 + 3) * (6 - 4))

הגדרה 2: שיטת prefix

כתיב פולני (שיטת prefix) הוא שיטה שפותחה על ידי הלוגיקן הפולני יאן לוקשביץ בשנת 1920. שיטה זו באה לפצות על מספר חסרונות של שיטת הכתיב הנפוצה infix:

- הצורך להגדיר כללי קדימות אופרטורים.
- הצורך בשימוש בסוגריים.

העיקרון המנחה של הכתיב הפולני הוא כתיבת הפעולה (האופרטור) לפני האופרנדים (המספרים) שעליהם הוא פועל. לדוגמה

+ 21 2

במקום

(21 + 2)

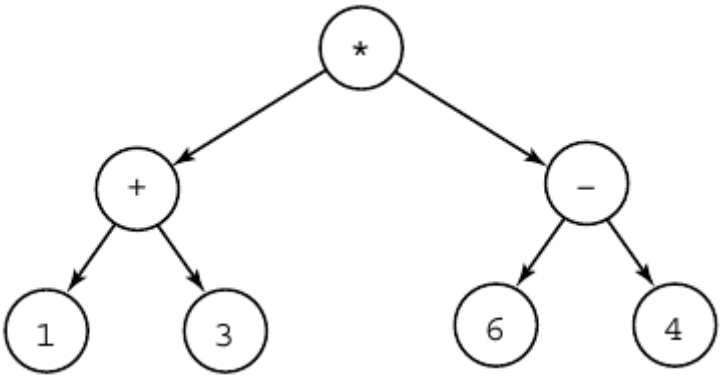
שיטה זו איננה בשימוש בלוגיקה כיום, אך בשל העובדה שקל לנתח ביטוי בכתיב פולני באמצעות מחשב, נעשה בה שימוש במספר שפות תכנות, בעיקר בשפות תכנות מבוססות מחסנית כמו פוסטסקריפט.

דוגמאות:

Infix expression	Prefix expression
(2 + 3)	+ 2 3
(4 * 6)	* 4 6
((2 + 3) * (4 * 6))	* + 2 3 * 4 6
((1 + 3) * (6 - 4))	* + 1 3 – 6 4
(((1 + 3) * (6 - 4)) + ((9 - 7) * 8))	+ * + 1 3 – 6 4 * – 9 7 8

הגדרה 3: עץ ביטוי

נוח להציג את הביטויים האריתמטיים בעזרת עצים בינאריים. עצים כאלה נקראים עצי ביטוי.
למשל, עבור לביטוי ((4 - 6) * (3 + 1)) מתאים העץ:



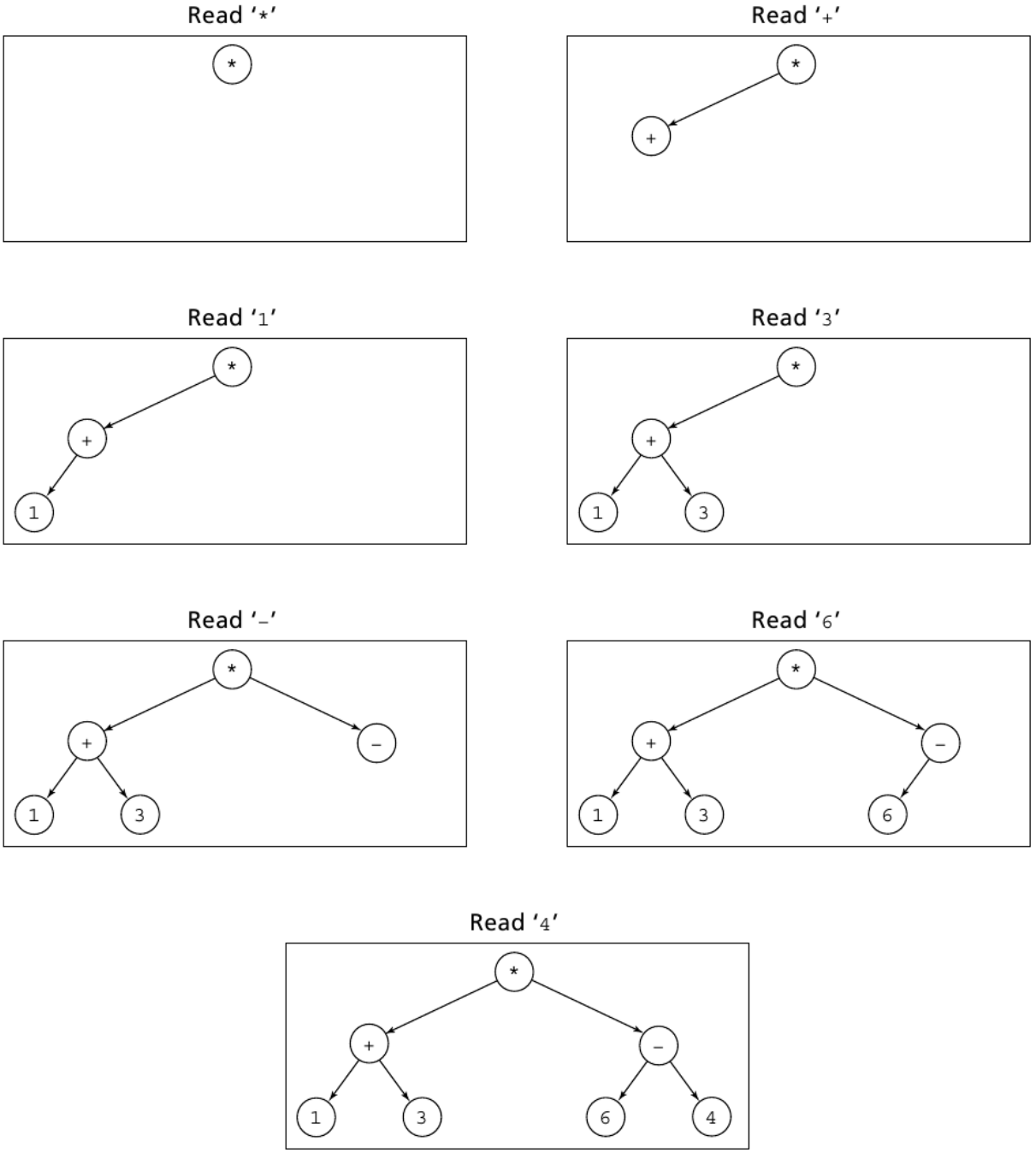
אלגוריתם ליצירת עץ ביטוי:

נניח כי נתון ביטוי אריתמטי בצורת **prefix**. נעבור על הביטוי **משמאל לימין** וניצור עץ ביטוי באופן הבא:

```
Read the next arithmetic operator or numeric value.
Create a node containing the operator or numeric value.
if the node contains an operator
    then Recursively build the subtrees that correspond to the
         operator's operands.
    else The node is a leaf node.
```

דוגמא:

נדגים איך האלגוריתם המתואר לעיל עובד על הביטוי: * + 1 3 - 6 4



1) כתבו מחלקה גנרית בשם BinaryTree הממשת עץ בינארי.
המחלקה צריכה לממש את המנשק הנתון BinaryTreel<T>.(המנשק נתון בגיטהב, ראה הוראות להלן).
הבנאים צריכים לאפשר יצירת עלה (בנאי עם פרמטר אחד מטיפוס T), ויצירת צומת פנימי (בנאי עם שלשה פרמטרים לפי הסדר הבא: פרמטר מטיפוס T, רפרנס לתת-עץ של בן שמאלי, ורפרנס לתת-עץ של בן ימני).
המחלקה צריכה להיות בחבילה il.ac.telhai.ds.trees;

2) כתבו מחלקה גנרית המרחיבה את הנ"ל וממשת עץ בינארי מלא. (בעץ בינארי מלא לכל צומת שהוא לא עלה שני ילדים בדיוק).
שם המחלקה הוא FullBinaryTree<T>, והיא נמצאת באותה החבילה.

```
package il.ac.telhai.ds.trees;
public class FullBinaryTree<T> extends BinaryTree<T> {
...
}
```

הוסיפו שני בנאים בדומה לאלו שכתבתם ב 1).

3) כתבו מחלקה המרחיבה את FullBinaryTree<T>, וממשת עץ ביטוי. (הוסיפו בנאים לתבנית הבאה).

```
package il.ac.telhai.ds.trees;
import java.io.IOException;
import java.io.StreamTokenizer;
public class ExpressionTree extends FullBinaryTree<String>{
        /*
         * Read the stream tokenizer until EOF assuming and construct
         *  the expression tree corresponding to it.
         * The input contains a prefix expression.
         */
        public static ExpressionTree createTree(StreamTokenizer
                tokenizer) throws IOException {
        }

        /*
         * Returns the infix expression corresponding to the current tree (*)
         */
        public String infix() {
        }

        /*
         * Returns the prefix expression corresponding to the current tree (*)
         */
        public String prefix() {
        }

        /*
         * Evaluates the expression corresponding to the current tree
         * and returns its value
         */
        public double evaluate() {
        }
}
```

(*) פורמט הפלט צריך להיות כך:
עבור infix: הרווחים הקיימים הם רק לפני אופרטור ולאחריו (רווח אחד לפני ורווח אחד אחרי).
למשל: "(3 * (1 + 2))"
עבור prefix: נקבל את המחרוזת עם שני רווחים בין כל זוג איברים ברשימה, וכן רווח אחד בהתחלה ורווח אחד בסוף. למשל: " 3 1 2 + * "
יש להשתמש במתודות inorder ו-preOrder למימוש המתודות infix ו-prefix, בהתאמה.

4) הריצו את התוכנית PrePolishToInfix הנתונה לכם, ובדקו את התוצאות. שימו לב כי הקלט ל-PrePolishToInfix הוא בצורת preOrder.

סדר העבודה ופרטים טכניים

- שליפת הפרויקט DS-Lab06-ExpressionTree מתוך GITHUB בקישור:
  https://github.com/michalHorovitz/DSLab2021-2022Public
  - אם אין לכם גישה לפרויקט שהורדתם מ GITHUB במעבדות הקודמות יש לבצע שליפה מחדש.
  - אם יש לכם גישה לפרויקט שהורדתם מ GITHUB במעבדה הראשונה אז בצעו:
    - קליק על שם הפרויקט.
    - עכבר ימני
    - Team-->Pull
    - File-->Import->Git->Projects From Git->Existing Local Repository

פורמט קובץ ההגשה ובדיקתו:

פורמט: יש להגיש קובץ ZIP בשם

37_Lab06_123456789_987654321.zip

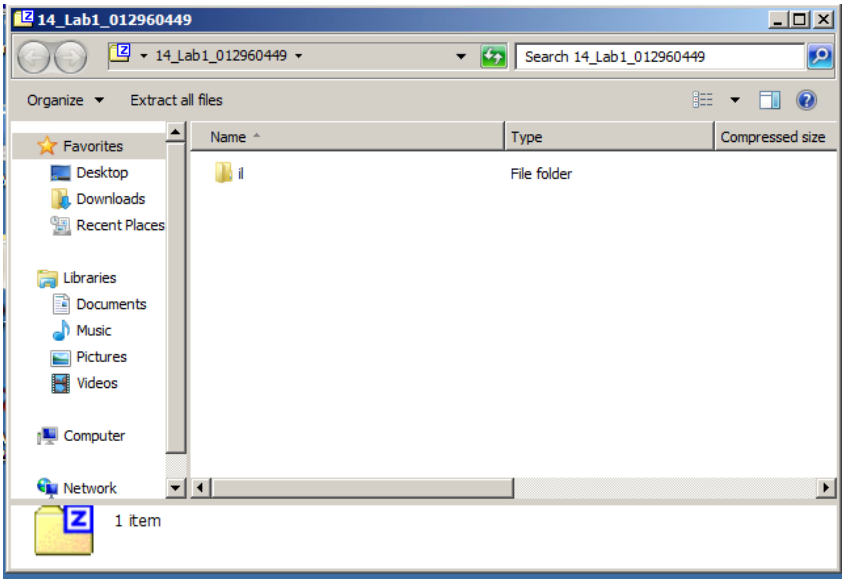
(כמובן, יש להחליף את המספרים עם מספרי ת.ז. של המגישים).

על הקובץ להכיל את כל קבצי ה JAVA שכתבתם כאשר הם נמצאים בתיקייה

il/ac/telhai/ds/trees

כלומר, השורש של קובץ ההגשה יכיל רק תיקייה בשם il.

ומכיל את כל קבצי - java . להמחשה תמונה של קובץ כזה שנפתח ב - WindowsExplorer



בדיקת קובץ ההגשה: בדקו את הקובץ שיצרתם בתוכנת הבדיקה בקישור:

https://cs.telhai.ac.il/homework/

ראו סרטון הדגמה של השימוש בתוכנת הבדיקה.

חשוב !!!

בדיקת ההגשות תבוצע ברובה ע"י תוכנית הבדיקה האוטומטית הנ"ל. תוצאת הבדיקה תהייה בעיקרון זהה לתוצאת הבדיקה הנ"ל שאתם אמורים לערוך בעצמכם . כלומר, אם ביצעתם את הבדיקה באתר החוג, לא תקבלו הפתעות בדיעבד. אחרת, ייתכן שתרגיל שעבדתם עליו קשה

ייפסל בגלל פורמט הגשה שגוי וכו'. דבר שהיה ניתן לתקנו בקלות אם הייתם מבצעים את הבדיקה.
היות ואין הפתעות בדיעבד, לא תינתן אפשרות של תיקונים, הגשות חוזרות וכד'.
הגשה שלא מגיעה לשלב הקומפילציה תקבל ציון 0.
הגשה שלא מתקמפלת תקבל ציון נמוך מ- 40 לפי סוג הבעיה.
הגשה שמתקמפלת תקבל ציון 40 ומעלה בהתאם לתוצאות הריצה, ותוצאת הבדיקה הידנית של הקוד (חוץ ממקרה של העתקה).
תכנית הבדיקה האוטומטית מכילה תוכנה חכמה המגלה העתקות. מקרים של העתקות יטופלו בחומרה