

תוכנה 1 – אביב תשע"ג

תרגיל מספר 4

(2 נקודות)

הנחיות כלליות:

קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.

- הגשת התרגיל תיעשה במערכת ה-moodle בלבד (<http://moodle.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש האוניברסיטאי שלכם ומספר התרגיל (לדוגמא, עבור שם המשתמש aviv יקרא הקובץ aviv_hw3.zip). קובץ ה-zip יכיל:
 - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
 - ב. קבצי ה-java של התוכניות אותם התבקשתם לממש.
 - ג. קובץ טקסט answers.txt עם התשובות לשאלות (אפשר קובץ טקסט או קובץ Word).

שימו לב,

- לכל אורך התרגיל ניתן להוסיף שירותי עזר בתנאי שאינם פוגעים בנדרש בסעיפים אחרים.
 - כדי להקל עליכם, סיפקנו מחלקות הבודקות חלק מהדוגמאות המופיעות בגוף התרגיל. אם הקוד שמימשתם עונה נכונה על הדוגמאות, תתקבל הודעה "QX pass" כאשר X הוא מספר השאלה. אחרת, תתקבלנה הודעות שגיאה מתאימות.
- חשוב:** הודעת הצלחה אינה מעידה על נכונות הקוד אלא על התאמה לדוגמאות הספציפיות אותן סיפקנו. לעומת זאת, הודעות שגיאה מעידות על כך שהמימוש שגוי.
- חשוב מאוד:** קוד שאינו מתקמפל עם תוכניות הבדיקה לא ייבדק.
- בהצלחה

1. מתחשבנים (60 נקודות)

בשאלה זו נגדיר שפת תכנות (פשוטה מאוד) לחישובים מתמטיים שתקרא CALC. כמו כן, נבנה מפרש ומהדר (compiler) לשפה. בשפה זו מבצעים חישובים בעזרת מחסנית, ומשתמשים באחת משתי פעולות בסיסיות:

- א. הכנסת מספרים ממשיים (כלומר מסוג double) למחסנית.
- ב. הפעלת פקודה. רוב הפקודות שולפות איברים מהמחסנית, מחשבות בעזרתן ערך חדש ומכניסות ערך זה חזרה למחסנית. (לדוגמא פקודת חיבור מכניסה למחסנית את סכום שני האיברים שהיו בראשה).

להלן רשימת הפקודות המוגדרות ב-CALC (בטבלה ראש המחסנית הוא מימין).

הפקודה	מחסנית לפני החישוב	מחסנית אחרי החישוב	הסבר
+	[a,b,c]	[a,b+c]	חיבור
-	[a,b,c]	[a,c-b]	חיסור
*	[a,b,c]	[a,b*c]	כפל
/	[a,b,c]	[a,c/b]	חילוק
**	[a,b,c]	[a,c ^b]	העלאה בחזקה
PI	[a,b,c]	[a,b,c,3.14159....]	הוספת π למחסנית
E	[a,b,c]	[a,b,c,2.71828....]	הוספת e למחסנית
swap	[a,b,c]	[a,c,b]	החלפת האיברים שבראש המחסנית
רמז: השתמשו במחלקה Math של Java כדי לממש העלאה בחזקה וכדי למצוא את ערכי הקבועים π ו-e.			

לתוכנית ב-CALC יש את המבנה הבא:

- השורה הראשונה בכל תוכנית מורכבת מהמילה args ולאחריה מספר הפרמטרים שהתוכנית מצפה להם. התוכנית מניחה שפרמטרים אלו מצויים במחסנית בתחילת הריצה.
- שורות חישוב. החל מהשורה השנייה, כל שורה מכילה בדיוק פעולה אחת (פקודה מהטבלה או מספר שיוכנס לראש המחסנית).

דוגמא, תוכנית המחברת את שני הפרמטרים שלה:

```
args 2
+
```

דוגמא, תוכנית המוסיפה מע"מ בסך 16% לפרמטר המועבר לה:

```
args 1
1.16
*
```

דוגמאות נוספות מצויות בקבצי CALC המסופקים עם התרגיל (בספריית resources).

חלק א' – מה זאת אומרת? (המפרש, Interpreter)

בחלק זה נממש את המחלקה Interpreter אשר מריצה תכניות ב-CALC תחת ההנחות הבאות:

- גודל המחסנית המקסימלי הדרוש להרצת תוכנית הוא 512.
- תכנית ה-CALC נתונה בזרם ב-System.in (היעזרו במחלקה Scanner כפי שראיתם בתרגול).
- המפרש מריץ תוכנית על ידי קריאת שורות בודדות וביצוע האמור בכל שורה בנפרד (הפעלת פקודה או הכנסת מספר למחסנית) עד סוף התוכנית.

- כאשר מגיעים לסוף התכנית (אין עוד שורות בזרם System.in), המפרש מדפיס את הערך הנמצא בראש המחסנית (תוצאת החישוב).

א. ממשו את פונקציות העזר הבאות אשר ישמשו ניהול המחסנית (בכל הדוגמאות מצב המחסנית לאחר ביצוע הפקודה נתון בהערה, ראש המחסנית הוא מימין).

i. הפונקציה push מוסיפה את הערך value אל ראש המחסנית.

```
public static void push(double value)
```

דוגמאות:

```
// initial stack: []
push(1.1); // stack: [1.1]
push(3); // stack: [1.1, 3.0]
push(5.4); // stack: [1.1, 3.0, 5.4]
```

ii. הפונקציה pop מוציאה (ומחזירה) את הערך שנמצא בראש המחסנית.

```
public static double pop()
```

דוגמאות:

```
// initial stack: [1.1, 3.0, 5.4]
double a = pop(); // stack: [1.1, 3.0]; a == 5.4
double b = pop(); // stack: [1.1]; b == 3.0
double c = pop(); // stack: []; c == 1.1
```

iii. הפונציה top מחזירה את הערך שנמצא בראש המחסנית מבלי להסירו.

```
public static double top()
```

דוגמאות:

```
// initial stack: []
push(1.1); // stack: [1.1]
double a = top(); // a == 1.1
push(3); // stack: [1.1, 3.0]
double b = top(); // b == 3.0
push(5.4); // stack: [1.1, 3.0, 5.4]
double c = top(); // c == 5.4
```

iv. הפונקציה size() מחזירה את מספר האיברים במחסנית.

```
public static int size()
```

דוגמאות:

```
// initial stack: []
push(1.1); // stack: [1.1]
int a = size(); // a == 1
push(3); // stack: [1.1, 3.0]
int b = size(); // b == 2
push(5.4); // stack: [1.1, 3.0, 5.4]
int c = size(); // c == 3
```

ב. ממשו את הפונקציה runCommand אשר מריצה שורה אחת של תוכנית CALC ומחזירה את הערך המצוי בראש המחסנית לאחר ההרצה.

```
public static double runCommand(String cmd)
```

דוגמאות:

```
// initial stack: []
runCommand("2.2"); // stack: [2.2]
runCommand("2"); // stack: [2.2, 2.0]
runCommand("+"); // stack: [4.2]
runCommand("0.5"); // stack: [4.2, 0.5]
runCommand("*"); // stack: [2.1]
runCommand("2"); // stack: [2.1, 2.0]
runCommand("swap"); // stack: [2.0, 2.1]
runCommand("**"); // stack: [4.41]
runCommand("PI"); // stack: [4.41, 3.14159]
runCommand("E"); // stack: [4.41, 3.14159, 2.71828]
```

ג. ממשו את הפונקציה `interpret` המריצה תוכנית `CALC` שלמה הנתונה ב-`System.in`. הפונקציה מחזירה `true` אם הביצוע הצליח ו-`false` במקרים הבאים:

i. התוכנית לא מתחילה בשורת `args`

ii. מספר האיברים במחסנית קטן מזה שהתוכנית מצפה לו.

ניתן להניח כי שאר הקלט תקין (כלומר ששורות התוכנית הן מספרים או פקודות ידועות).

```
public static boolean interpret ()
```

דוגמאות:

```
// initial stack: []
// CALC script:
// args 0
// 2.2
// 2
// +
boolean b = interpret(); // stack: [4.2]; b == true

// initial stack: [10]
// CALC script:
// args 1
// 2
// PI
// *
// *
// 1
// swap
boolean b = interpret(); // stack: [1.0, 62.8318]; b == true

// CALC script:
// 2
boolean b = interpret(); // b == false ("args" line is missing)

// initial stack: []
// CALC script:
// args 1
// PI
boolean b = interpret(); // b == false (expecting 1 argument)

// initial stack: [1.0]
// CALC script:
// args 2
// PI
boolean b = interpret(); // b == false (expecting 2 arguments, got 1)
```

ד. ממשו את הפונקציה main המריצה תוכנית CALC ומדפיסה את תוצאת החישוב. הפרמטרים המועברים בשורת הפקודה מיועדים לתוכנית ה-CALC, יש להכניסם למחסנית לפני הרצתה.

דוגמאות (הרצה משורת הפקודה בספרייה bin):

```
$ java Interpreter 10 < ../resources/Circumference.calc  
62.83185307179586
```

```
$ java Interpreter 10 < ../resources/HalfCircleArea.calc  
157.07963267948966
```

```
$ java Interpreter 10 < ../resources/Vat.calc  
11.6
```

הערה: ציון שם קובץ בשורת הפקודה לאחר הסימן "<" מורה למערכת ההפעלה לרשום את תוכן הקובץ לזרם System.in.

חלק ב' – ובכל זאת Java (המהדר)

בחלק זה נממש את המחלקה Compiler המתרגמת תכניות CALC למחרוזת המכילה קוד Java. לאחר מכן ניצור קובץ class מהמחרוזת ע"י שימוש בקוד מוכן.

הדרכה:

- נעזר במחסנית של מחרוזות כדי לייצר ביטוי מתמטי ב-Java. לדוגמא, את הפקודות "2", "2", ו-"**" ב-CALC נייצג כ-"(2*2)" ב-Java, בצורה הבאה:

פקודה	מצב המחסנית
2	["2"]
2	["2", "2"]
*	["(2*2)"]

הערה: היזהרו במימוש פעולות בהם סדר הארגומטים ב-Java אינו בהכרח תואם את זה שב-CALC (לדוגמא: חילוק, חזקה).

- א. ממשו פונקציות עזר לניהול מחסנית מחרוזות, בדומה לפונקציות מהחלק הקודם.

חתימות הפונקציות:

```
private static void push(String i)  
private static String pop()  
private static String top()  
public static int size()
```

ב. ממשו את הפונקציה generateHeader המקבלת שם מחלקה (מחרוזת) ואת מספר הפרמטרים הנדרשים לתוכנית ה-CALC ומחזירה מחרוזת המכילה קוד Java המגדיר מחלקה בשם שסופק, בודק את מספר הארגומנטים, ומציב אותם במשתנים.

- את הפרמטר ה-X בשורת הפקודה מציבים במשתנה argX.

- הקפידו על הזחה (תווי TAB), מעברי שורה ורווחים בקוד הנוצר כדי להקל בבדיקה.

הערה: הפונקציה גם מכניסה את שמות המשתנים למחסנית לשימוש מאוחר יותר.

```
public static String generateHeader(String className, int expectedArgs)
```

דוגמא 1 (ללא פרמטרים):

```
String test = Compiler.generateHeader("Test", 0);
```

המשתנה test יכיל את המחרוזת הבאה:

```
public class Test {
    public static void main(String[] args) {
        if(args.length < 0) {
            System.out.println("Missing arguments");
            return;
        }
    }
}
```

דוגמא 2 (שני פרמטרים):

```
String test = Compiler.generateHeader("WithArguments", 2);
```

המשתנה test יכיל את המחרוזת הבאה:

```
public class WithArguments {
    public static void main(String[] args) {
        if(args.length < 2) {
            System.out.println("Missing arguments");
            return;
        }
        double arg0 = Double.parseDouble(args[0]);
        double arg1 = Double.parseDouble(args[1]);
    }
}
```

ג. ממשו את הפונקציה processCommand המקבלת שורה אחת של תוכנית CALC ומעדכנת את המחסנית (כך שבסיום, הביטוי ב-Java לתוצאת החישוב ימצא בראש המחסנית). הפונקציה מחזירה את הביטוי הנמצא בראש המחסנית.

```
public static String processCommand(String cmd)
```

דוגמאות:

```
// initial stack: []
processCommand("2.2"); // stack: ["2.2"]
// ("2.2" may be written as "2.20", "2.20000" etc.)
processCommand("2"); // stack: ["2.2", "2.0"]
// ("2.0" may be written as "2.000" etc.)
processCommand("+"); // stack: ["(2.0 + 2.2)"]
// (may also be "(2.2 + 2.0)", "(2.2+2.0)", etc.)
processCommand("0.5"); // stack: ["(2.0 + 2.2)", "0.5"]
processCommand("*"); // stack: ["(0.5 * (2.0 + 2.2))"]
// (may also be: "(2.0 + 2.2) * 0.5" etc.)
```

הערה: ייתכנו מספר רב של ייצוגים שונים (ונוכחים) עבור אותה תוכנית CALC ולכן תוכנית הבדיקה מבצעת בדיקות מוגבלות בלבד עבור סעיף זה.

ד. ממשו את הפונקציה generateFooter המחזירה מחרוזת ובה קוד Java משלים ל-header שנוצר בסעיף ב', אשר מדפיס את תוצאת החישוב למסך.

```
public static String generateFooter()
```

דוגמאות:

דוגמא 1: ללא ארגומנטים

```
// initial stack: ["1.0"]
String test = Compiler.generateFooter();
```

המשתנה test יכיל את המחזורת הבאה:

```
        double result = 1.0;
        System.out.println(result);
    }
}
```

דוגמא 2: שני ארגומנטים

```
// initial stack: ["(1.0 * 20.1)"]
String test2 = Compiler.generateFooter();
```

המשתנה test2 יכיל את המחזורת הבאה:

```
        double result = (1.0 * 20.1);
        System.out.println(result);
    }
}
```

ה. ממשו את הפונקציה generateSourceCode המקבלת מסלול לקובץ ובו תוכנית CALC ומחזירה את קוד ה-Java השקול במחלקה ששמה זהה לשם קובץ הקלט (ללא הסיימת).

עבור תוכנית לא חוקית (כלומר שלא מתחילה בשורת args) תוחזר מחזורת ריקה.

```
public static String generateSourceCode(Path path) throws IOException
```

(עבור קבצים המסופקים עם תוכנית הבדיקה בספרייה resources)

```
Compiler.generateSourceCode(Paths.get("resources", "Vat.calc")) -->
```

```
"public class Vat {
    public static void main(String[] args) {
        if(args.length < 1) {
            System.out.println("Missing arguments");
            return;
        }
        double arg0 = Double.parseDouble(args[0]);
        double result = (1.16 * arg0);
        System.out.println(result);
    }
}"
```

```
Compiler.generateSourceCode(Paths.get("resources", "HalfCircleArea.calc"))
-->
```

```
"public class HalfCircleArea {
    public static void main(String[] args) {
        if(args.length < 1) {
            System.out.println("Missing arguments");
            return;
        }
        double arg0 = Double.parseDouble(args[0]);
```

```

    double result = ((Math.PI * Math.pow(arg0, 2.0)) / 2.0);
    System.out.println(result);
}
}"

```

1. ממשו את main() אשר מייצרת קוד Java עבור קובץ המכיל תוכנית CALC המועבר בשורת הפקודה. לאחר מכן יש להשתמש בשירות generateFromSource של המחלקה Bytecode (המסופקת עם התרגיל) כדי לקמפל את קוד ה-Java לקובץ class.

חשוב מאוד להקפיד להריץ את המחלקה בעזרת java מה-JDK ולא מה-JRE, אחרת המחלקה Bytecode תכשל.

דוגמאות (הרצה משורת הפקודה בספרייה bin):

```

$ java Compiler ../resources/Circumference.calc
$ java Circumference 10
62.83185307179586

```

```

$ java Compiler ../resources/HalfCircleArea.calc
$ java HalfCircleArea 10
157.07963267948966

```

```

$ java Compiler ../resources/Vat.calc
$ java Vat 10
11.6

```

הערה: תוכנית הבדיקה איננה בודקת סעיף זה.

2. מנו שלושה הבדלים בין שימוש ב-Interpreter לבין שימוש ב-Compiler.

רמזים: כיצד משפיע גודל מחסנית? כיצד רצה התוכנית Silly.calc הנתונה עם התרגיל?

2. שחור ופתור (40 נקודות)

שחור ופתור הוא חידה שבה יש לגלות ציור חבוי בטבלת משבצות. בראש כל שורה ועמודה מופיעה סדרה של מספרים המייצגים רצפים של משבצות שחורות, בסדר נתון. לדוגמה, הסדרה [2,3] מייצגת רצף של שתי משבצות המופיע לפני רצף של שלוש משבצות. המרחק של הרצף הראשון מקצה התמונה אינו ידוע. המרחק בין הרצפים גדול מ-1 (לפחות תא לבן אחד ביניהם).

דוגמא לחידת שחור ופתור (ופתרונה):

				2	1	3	1	2
				2	3	1	3	2
			5					
1	1	1						
			3					
		2	2					
			5					

נייצג את הקלט כשני מערכים דו-מימדיים, האחד עבור השורות והשני עבור העמודות, המכילים את סדרות הרצפים. תמונת הפתרון, תיוצג במטריצה דו-מימדית, בה ערכו של תא לבן הוא 0 וערכו של תא שחור הוא 1.

נשתמש באלגוריתם הנאיבי הבא לפתרון החידה:

- א. נבחר צביעה חוקית עבור השורה ה- i (נתעלם מהמידע על העמודות).
- ב. אם הצביעה סותרת את חוקיות העמודות, נחליף אותה, במידת האפשר, בצביעה חוקית חדשה. אחרת, נעבור לבחירת צביעה חוקית (שלב א') לשורה ה- $i+1$.
- ג. אם הצלחנו למצוא צביעה לכל השורות, מצאנו פתרון חוקי ולכן נסיים את האלגוריתם.

סעיף א' – שורות

בסעיף זה נממש שתי פונקציות עזר למציאת צביעות חוקיות של שורה. נייצג צביעה של שורה כמערך בו המקום ה- i מכיל את האינדקס בשורה בו מתחיל הרצף ה- i .

- א. ממשו את הפונקציה `firstPattern` המקבלת מערך של אורכי רצפים, ומחזירה צביעה (מערך של אינדקסים) הממקמת את הרצפים בתחילת השורה אחד אחרי השני (במיקום השמאלי ביותר האפשרי לכל רצף).

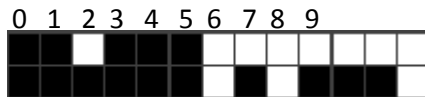
```
public static int[] firstPattern(int[] lengths)
```

דוגמאות:

```
firstPattern(new int[] { 2, 3 }); --> [0,3]
```

```
firstPattern(new int[] { 6, 1, 3 }); --> [0,7,9]
```

ייצוג גרפי



- ב. ממשו את הפונקציה `nextPattern` המקבלת צביעה (מערך של אינדקסים), מערך של אורכי רצפים, ואת רוחב התמונה, ומעדכנת את הצביעה ע"י הזזת רצפים ימינה. אם לא ניתן לעדכן את הצביעה (כל הרצפים נמצאים במקום הימני ביותר), הפונקציה תחזיר `false`, אחרת היא תחזיר `true`.

העדכון נעשה עפ"י האלגוריתם הבא:

- i. קבע את הרצף הימני ביותר כרצף הנוכחי.
- ii. הזז את הרצף הנוכחי משבצת אחת ימינה.
- iii. אם ניתן לעשות כן, מקם את הרצפים הנמצאים מימין לרצף הנוכחי במיקום השמאלי ביותר האפשרי להם והחזר `true`. אחרת המשך.
- iv. חזור על (ii) עם הרצף הקודם (משמאל) כרצף הנוכחי. אם הרצף הנוכחי הוא השמאלי ביותר החזר `false`.

```
public static boolean nextPattern(int[] starts, int[] lengths, int width)
```

דוגמאות (ערך החזרה ותוכן המערך `firstPattern` מופיעים בהערה):

```
int[] lengths = new int[] { 2, 3 };
int[] firstPattern = firstPattern(lengths);

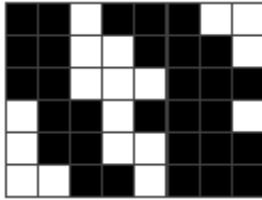
nextPattern(firstPattern, lengths, 8); // true, [0,4]
nextPattern(firstPattern, lengths, 8); // true, [0,5]
nextPattern(firstPattern, lengths, 8); // true, [1,4]
```

```

nextPattern(firstPattern, lengths, 8); // true, [1,5]
nextPattern(firstPattern, lengths, 8); // true [2,5]
nextPattern(firstPattern, lengths, 8); // false

```

ייצוג גרפי



ג. ממשו את המתודה `renderPatternInRow`, המקבלת צביעה (מערך אינדקסים), מערך אורכים ומערך המייצג שורה בתמונת הפלט, ומ"ציירת" את השורה בהתאם לאמור במערכים אלו.

```

public static void renderPatternInRow(int[] pattern, int[] lengths,
                                     int[] row)

```

דוגמאות:

```

int[] row = new int[8];
int[] lengths = new int[] { 2, 3 };
renderPatternInRow(new int[]{0, 3}, lengths, row); // row: [1,1,0,1,1,1,0,0]
renderPatternInRow(new int[]{1, 5}, lengths, row); // row: [0,1,1,0,0,1,1,1]

```

סעיף ב' – עמודות

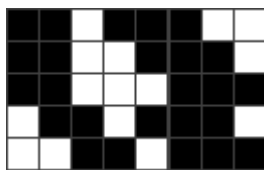
המתודה `anyColumnViolated` (המסופקת עם התרגיל) בודקת האם צביעת n השורות הראשונות בתמונה סותרת את אורכי הרצפים בעמודה כלשהיא ע"י הפעלת המתודה `isColumnViolated` לכל עמודה. השלימו את המימוש של `isColumnViolated` המוודאת כי צביעת n השורות הראשונות בעמודה נתונה אינה סותרת את אורכי הרצפים הנתונים עבור עמודה זו.

```

private static boolean isColumnViolated(int[][] picture, int[] column_def,
                                       int n, int column_index)

```

דוגמאות:



```

int[][] picture = { {1,1,0,1,1,1,0,0}, {1,1,0,0,1,1,1,0},
                   {1,1,0,0,1,1,1,0}, {0,1,1,0,1,1,1,0},
                   {0,0,1,1,0,1,1,1} };

```

```

// the first two rows of the first column are filled, compatible with {2}
isColumnViolated(picture, new int[]{2}, 2, 0) --> false

```

```

// the first three rows of the first column are filled, violates {2}
isColumnViolated(picture, new int[]{2}, 3, 0) --> true

```

```

// the first five rows of the second column are filled, compatible with {4}
isColumnViolated(picture, new int[]{4}, 5, 1) --> false

```

```

// the last two rows of the third column are filled, compatible with {4}

```

```
isColumnViolated(picture, new int[]{4}, 5, 2) --> false
```

סעיף ג' – מחוברים

נטר לחבר את הפונקציות שמימשנו כדי לפתור את החידה. המתודה solve() מאתחלת את תמונת הפלט וקוראת לפונקציה solveRow() הפותרת את החידה באמצעות האלגוריתם הנאיבי שהוצג לעיל.

בחירת הצביעות בשלבים (א) ו-(ג) באלגוריתם תמומש בעזרת פונקציות העזר מסעיף א' ואילו שלב (ב) באלגוריתם ימומש באמצעות פונקציות העזר מסעיף ב'.

```
public static boolean solveRow(int[][] picture, int[][] columns,
                               int[][] rows, int current_row)
```

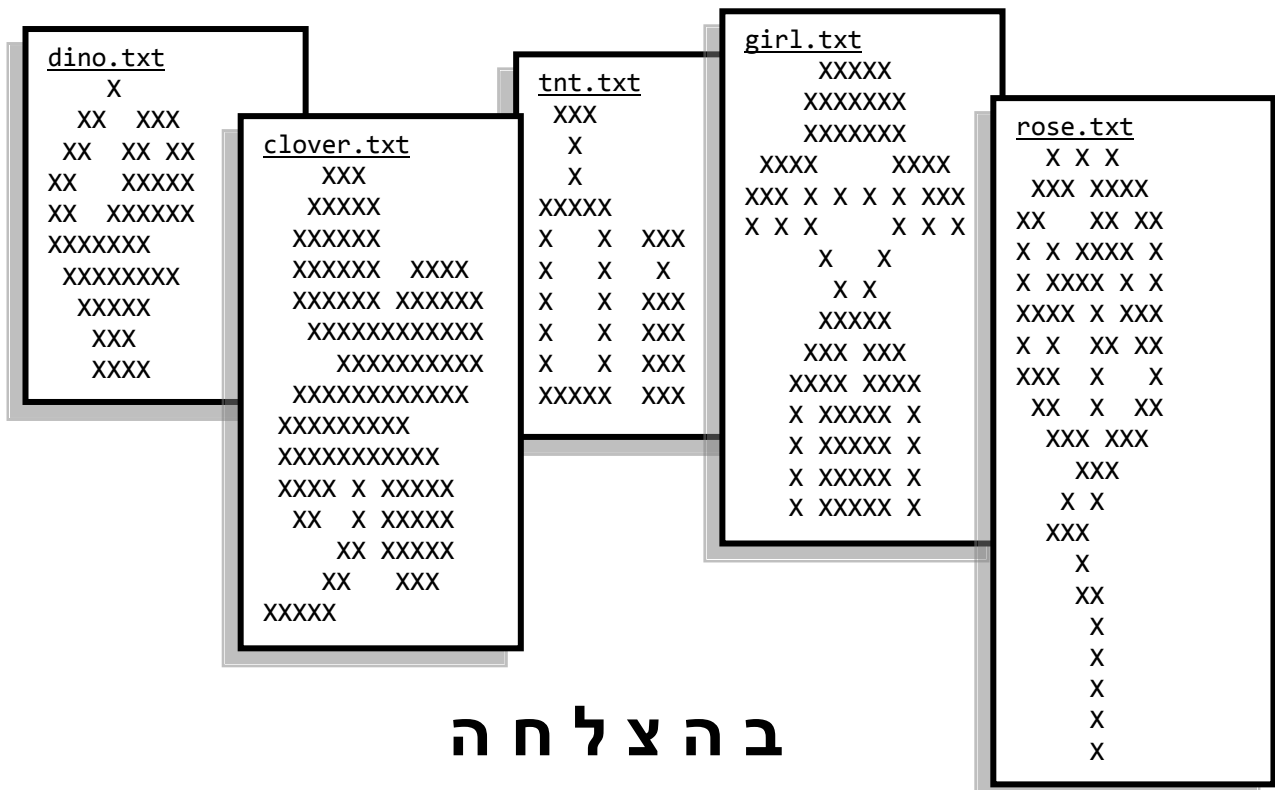
דוגמאות:

```
int[][] solve_columns = {{3}, {4}, {2}, {1, 1}, {2, 1}, {5}, {4}, {1, 1}};
int[][] solve_rows = {{2,3}, {2,3}, {2,3}, {2,3}, {2,3}};
int[][] result = Griddler.solve(solve_columns, solve_rows);
result == { {1,1,0,1,1,1,0,0}, {1,1,0,0,1,1,1,0}, {1,1,0,0,1,1,1,0},
            {0,1,1,0,1,1,1,0}, {0,0,1,1,0,1,1,1}};
```

```
printPicture(result); // provided with the assignment, outputs the image to
                      // the console
```

```
XX XXX
XX XXX
XX XXX
XX XXX
XX XXX
```

דוגמאות נוספות: אם תוכנית הבדיקה לא מזהה שגיאות, יודפסו פתרונות החידות הבאות למסך לפני הכיתוב Q2 Passed (כל החידות מצויות בקבצים בספרייה resources).



קבצי החידה והדוגמאות הגרפיות בשאלה 2 מבוססים על חומר שהופק מהאתר: <http://www.gridlers.net>