ביטויים מתמטיים
PostFix, PreFix, InFix

אחד היישומים החשובים של מדעי המחשב הינו ייצוגם של ביטויים מתמטיים.

כל אחת מהפעולות המתמטיות - חיבור, חיסור, כפל, וחילוק הינה **פעולה בינארית**, כלומר פעולה שבה יש **אופרטור** (operator) אחד - הפעולה החישובית, הפועל על שני **אופרנדים** (operand) - המשתנים או המספרים עליהם מבוצעת הפעולה החישובית.

ניתן לייצג פעולה חישובים באחד משלושת הייצוגים:

- ייצוג **תוכי infix** - האופרטור נמצא בין שני האופרנדים: A + B
- ייצוג **תחילי prefix** - האופרטור נמצא לפני האופרנדים: + A B
- ייצוג **סופי postfix** - האופרטור נמצא אחרי שני האופרנדים: A B +

כדי לחשב את הביטוי A + B * C כפי שהוא כתוב בייצוג תוכי, יש להתחשב **בכללי הקדימויות** (precedence) של הפעולות החישוביות.

כללי הקדימות (מהגבוה לנמוך):

- העלאה בחזקה (נסמן אותה בסימן ^)
- כפל, חילוק
- חיבור, חיסור
- כאשר מופיעים כמה אופרטורים מאותה דרגת קדימות, יהיה סדר הקדימויות משמאל לימין
- כאשר מופיעות כמה פעולות חזקה A ^ B ^ C, יהיה סדר הקדימויות מימין לשמאל: A ^ (B ^ C)
- שימוש בסוגריים מאפשר לכפות סדר קדימויות שונה.

ייצוג תוכי

במתמטיקה אנו מבצעים חישובים בייצוג תוכי (האופרטור נמצא בתוך / בין האופרנדים עליהם היא מתבצעת) לדוגמה: 3 + 5

ייצוג תוכי מחייב שימוש בסוגריים כדי לשמור על סדר החישוב הרצוי: 2 * (3 + 5)

הארה לייצוג סופי

בייצוג סופי, מופיע האופרטור אחרי האופרנדים.

כדי להמיר מייצוג תוכי לייצוג סופי, הרי שעל פי כללי הקדימות, עלינו להמיר תחילה את חלק הביטוי המחושב ראשון, ולאחריו את חלק הביטוי המחושב אחריו בסדר הקדימויות.

תרגיל 1: המר כל אחד מהביטויים שלהלן מייצוג תוכי לייצוג סופי.

ייצוג תוכי	ייצוג סופי
A + B	
A + B - C	
(A + B) * (C - D)	
A ^ B * C - D + E / F / (G + H)	
((A + B) * C - (D - E)) ^ (F + G)	
A - B / (C * D ^ E)	

הארה לייצוג תחילי

בייצוג תחילי, מופיע האופרטור לפני האופרנדים.

תרגיל 2: המר כל אחד מהביטויים שלהלן מייצוג תוכי לייצוג תחילי.

ייצוג תוכי	ייצוג תחילי
A + B	
A + B - C	
(A + B) * (C - D)	
A ^ B * C - D + E / F / (G + H)	
((A + B) * C - (D - E)) ^ (F + G)	
A - B / (C * D ^ E)	

שים לב לכך שהייצוג התחילי של ביטוי מורכב אינו תמונת ראי של הייצוג הסופי.

ביטויים בייצוג סופי ותחילי אינם זקוקים לסוגריים. סדר הופעת האופרטורים בביטוי בייצוגים אלו קובע את סדר הפעולות בחישוב הביטוי.

חישוב ביטוי בייצוג סופי

להלן אלגוריתם לחישוב הערך של ביטוי בייצוג סופי, תוך שימוש במחסנית.

כל אופרטור במחרוזת מתייחס לשני האופרנדים הקודמים לו במחרוזת.
שים ♥ שכל אחד מהאופרנדים יכול להיות תוצאה של הפעלת אופרטור קודם.

1 צור מחסנית-מספרים-ממשיים ריקה

2 נתונה **מחרוזת** הקלט.

3 כל עוד לא סוף המחרוזת בצע:

3.1 קרא האות הבאה.

3.2 אם **אופרנד**, (אם לא אחד האופרטורים: '^' , '/' , '*' , '-' , '+')
דחוף אותו ל**מחסנית**.

3.3 אחרת - // זהו **אופרטור** שהאופרנדים שלו הם שני הפריטים העליונים במחסנית

3.3.1 שלוף שני פריטים עליונים במחסנית

3.3.2 בצע עליהם את הפעולה המצוינת על ידי האופרטור.

3.3.3 דחוף את התוצאה חזרה למחסנית (זהו האופרנד עבור האופרטור הבא).

4 שלוף התוצאה מהמחסנית.

שים ♥: אם בעת ניסיון השליפה מתברר שהמחסנית ריקה – יש שגיאה בביטוי.

נדגים את פעולת האלגוריתם על הביטוי הבא המובא בייצוג סופי: 6 2 3 + - 3 8 2 / + * 2 ^ 3 +

x	op1	op2	result	stk
6				[6]
2				[2, 6]
3				[3, 2, 6]
+	2	3	5	[5, 6]
-	6	5	1	[1]
3	6	5	1	[3, 1]
8	6	5	1	[8, 3, 1]
2	6	5	1	[2, 8, 3, 1]
/	8	2	4	[4, 3, 1]
+	3	4	7	[7, 1]
*	1	7	7	[7]
2	1	7	7	[2, 7]
^	7	2	49	[49]
3	7	2	49	[3, 49]
+	49	3	52	[52]

התוצאה result היא 52.

תרגיל 3: חשב לפי האלגוריתם שלעיל, את הביטויים שלהלן בייצוג סופי: הנח כי: A=1 , B=2 , C=3

א. A B + C - C ^

ב. A B C + * C B A - + *

תרגיל 4: כתוב תכנית לחישוב ערך ביטוי post fix.

להלן חתימת הפעולה:

```
/*      פעולה המקבלת מחרוזת המכילה ביטוי postfix כך שכך אופרנד הינו מספר חד ספרתי,
        ומחזירה את תוצאת הביטוי
*/
public static double postfix (String expr)
```

פעולות עזר:

חתימת הפעולה	תיאור הפעולה
boolean isOperand (**char** ch)	פעולה המקבלת תו ומחזירה אמת אם הוא פעולה מהסוג: ^ , / , * , - , + ושקר אחרת
int toNumber (**char** ch)	פעולה המקבלת תו המכיל ספרה ומחזירה את ערכו המספרי
double resultOf (**double** op1, **double** op2, **char** op)	פעולה המקבלת שני מספרים ממשיים ופעולה ומחזירה את התוצאה המתקבלת: op1 op op2

תרגיל 5: שנה את הפעולה כך שתקלוט ביטוי הכתוב ב- postfix והערכים הם מספרים (לאו דווקא חד-ספרתיים)

הנחיות:

- הערכים יוקלדו כשהם מופרדים בתו רווח.
- קבע סימן מיוחד (נניח #) שיזוהה כסוף הביטוי.
- השתמש בפעולה **double** x = Double.*parseDouble*(str);
  המקבלת כפרמטר מחרוזת של ספרות ונקודה עשרונית וממירה אותה למספר ממשי.
  הנחה: המחרוזת תקינה (כלומר - ניתנת להמרה למספר ממשי).

  הערה: קיימת פעולה מקבילה במחלקה Integer

המרת ביטוי תוכי לביטוי סופי

אלגוריתם הקולט ביטוי בייצוג תוכי, תו אחר תו וממיר אותו לביטוי בייצוג סופי.
הנחה: הביטוי שנקלט מכיל סוגריים עגולים בלבד (סוגריים מקוננים)

1. צור מחרוזת ריקה ← str
2. צור מחסנית-תווים ריקה ← stk
3. קרא תו ראשון מהקלט ← ch
4. כל עוד לא נגמר הקלט, בצע:
   - 4.1 אם (ch סוגר פותח/שמאלי) דחוף את ch למחסנית
   - 4.2 אם (ch סוגר סוגר/ימני)
     שלוף את תוכן המחסנית עד (לא כולל) הסוגר השמאלי ושרשר למחרוזת str.
     בסיום - שלוף את הסוגר השמאלי.
   - 4.3 אם (ch אופרטור /פעולה-חשבונית)
     שלוף מהמחסנית את כל הפעולות בדרגת קדימויות גבוהה יותר ושרשר אותן למחרוזת str
     בסיום - דחוף את ch למחסנית
   - 4.4 אם (ch ספרה/מספר) שרשר אותו למחרוזת str
   - 4.5 קרא את התו הבא מהקלט ← ch
5. כל עוד המחסנית אינה ריקה
   שלוף ושרשר את תוכן המחסנית למחרוזת str

נדגים את פעולת האלגוריתם על הביטוי הבא המובא בייצוג תוכי: (7+3)*(8-2)

הפעולה	stk	str	ch
דחיפת)	[( ]	""	(
שרשור למחרוזת	[( ]	7	7
דחיפת +	[+ , ( ]	7	+
שרשור למחרוזת	[+ , ( ]	7 3	3
שליפה ושרשור, שליפת)	[ ]	7 3 +	)
דחיפת *	[*]	7 3 +	*
דחיפת)	[(, *]	7 3 +	(
שרשור למחרוזת	[(, *]	7 3 + 8	8
דחיפת -	[- , (, *]	7 3 + 8	-
שרשור למחרוזת	[- , (, *]	7 3 + 8 2	2
שליפה ושרשור, שליפת)	[*]	7 3 + 8 2 -	)
הסתיים הקלט, שליפה ושרשור למחרוזת	[ ]	7 3 + 8 2 - *	\r

תוצאת ההמרה מביטוי תוכי (infix) לביטוי סופי (postfix): 7 3 + 8 2 - *

תרגיל 6: הוסף לתכנית פעולה שתקלוט את הביטוי בייצוג תוכי תו אחר תו.
המחרוזת שנקלטה תשלח כפרמטר לפעולה המחשבת ומחזירה את ערך הביטוי postfix.

```
//--- פעולה הקולטת ביטוי בייצוג תוכי ממירה ומחזירה את הביטוי בייצוג סופי ---
public static String inFixToPostFix()
```

פעולות עזר:

תיאור הפעולה	חתימת הפעולה
פעולה המקבלת את התו שבראש המחסנית stkCh ואת תו הקלט strCh ומחזירה אמת אם התו שבראש המחסנית בעל קדימות גבוהה או שווה לקדימות של התו השני, ושקר אחרת.	**boolean** stronger (**char** stkCh, **char** strCh)
פעולה המקבלת את מערך הפעולות החישוביות ותו המכיל פעולה ומחזירה את מיקומו של התו במערך. (ככל שהמיקום במערך גבוה יותר, הפעילות בעלת קדימות גבוהה יותר.)	**int** find (**char** [] arr, **char** ch)

הנחיות:

כדי לחשב את הקדימות של הפעולה, ניצור את המערך הבא, המכיל את הפעולות החישוביות כך שהפעולה בעלת הקדימות הנמוכה בתחילת המערך וזו בעלת הקדימות הגבוהה בסופו:

```
char [] arr = { '(' , '+' , '-' , '*' , '/' , '^' };
```

נכניס גם את הסוגר השמאלי למערך, כשהוא בעל הקדימות הנמוכה ביותר.

כשנקלט תו שהוא פעולה, נשתמש בפעולה stronger כדי להוציא מהמחסנית את כל התווים הקודמים לו או שווים לו בסדר הקדימויות. סיום השליפה כשימצא תו בעל קדימות נמוכה או עד שנתקלנו בסוגר שמאלי (שאמור להישלף רק כשמגיע סוגר ימני).

הפעולה stronger מוצאת את מקומו של התו שבראש המחסנית במערך (שימוש בפעולה find) ואת מקומו של התו שנקלט. אם המקום של התו שבמחסנית גבוה מזה של התו שנקלט, הרי שהקדימות שלו גבוהה יותר.
אם הקדימות זהה, יש לבצע תחילה את הפעולה שבמחסנית (היא נמצאת משמאל לפעולה החדשה) ולכן גם היא תישלף ותשורשר למחרוזת.

כשתסתיים השליפה, יידחף התו למחסנית.