**עבודת בית 3 - מעשי**

מתרגלת אחראית: מר גנאדי קוגן

תאריך פרסום: 8.12.2022

תאריך הגשה: 22.12.2022

הנחיות כלליות:

* קראו היטב את השאלות.
* **ניתן להגיש את העבודה בזוגות או ביחידים.**
* הגשת העבודה תהיה דרך אתר הקורס במודל.
* העבודה תמומש בשפת תכנות C.
* יש להגיש תכנית המכילה שני קבצים (קובץ הגדרות - קובץ עם סיומת .h, קובץ המכיל כל גופי פונקציות עם סיומת с., אין צורך בהגשת main.c).
* יש לכווץ כל קבצי התוכנית לקובץ zip.
* שם הקובץ שיוגש למערכת ההגשה יהיה מורכב מת"ז של המגיש/ים.

לדוגמה:

111111111.zip - עבור הגשה ביחיד

111111111\_22222222.zip - עבור הגשה בזוג

* לפני ההגשה יש להשתייך לקבוצת ההגשה. במקרה של הגשה בזוגות, רק אחד מבני הזוג יגיש   
   את העבודה במודל.
* חובה לתעד את הקוד.
* במידה ומצאתם טעויות בעבודה או בקשות להבהרות, אנא שלחו מייל ל-   
  [genadko@ac.sce.sc.il](mailto:genadko@ac.sce.sc.il)
* איחור במועד ההגשה יגרור הורדה של ציון, 5 נק' לכל יום איחור או חלק ממנו.  
   בכל מקרה לא יהיה ניתן להגיש מעבר ל-3 ימי איחור ממועד ההגשה המקורי.  
   במקרים חריגים בלבד יש לפנות למרצה כדי לקבל אישור על הגשה באיחור.
* שאלות לגבי העבודה יש לשאול בפורום באתר הקורס ("מודל") או בשעות קבלה של המתרגל   
   האחראי **בלבד**. ***אין לשלוח שאלות במייל המתרגלות או המרצה.***
* להזכירכם: יש לשמור על הגינות אקדמית!

**מבוא**

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 פעולת 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 ב**כללי** **הקדימויות** (precedence) של הפעולות החישוביות.

**כללי הקדימות (מהגבוה לנמוך):**

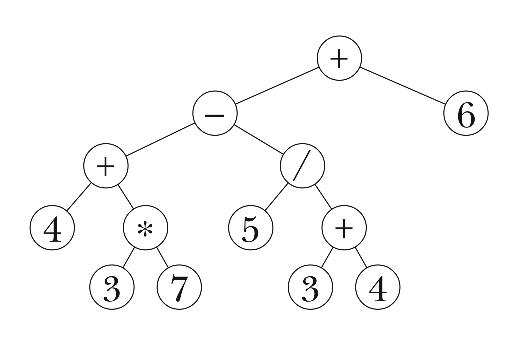
* העלאה בחזקה (נסמן אותה בסימן ^ )
* כפל, חילוק
* חיבור, חיסור
* כאשר מופיעים כמה אופרטורים מאותה דרגת קדימות, יהיה סדר הקדימויות משמאל לימין
* כאשר מופיעות כמה פעולות חזקה A ^ B ^ C, יהיה סדר הקדימויות מימין לשמאל: A ^ (B ^ C)
* שימוש בסוגריים מאפשר לכפות סדר קדימויות שונה.

במתמטיקה אנו מבצעים חישובים בייצוג תוכי )האופרטור נמצא בין האופרנדים עליהם הוא מתבצע לדוגמה: ( 5 + 3 ). ייצוג תוכי מחייב שימוש בסוגריים כדי לשמור על סדר החישוב הרצוי:

5) + 3 ) \* 2.

ניתן לתאר ביטוי אריתמטי בעזרת עץ בינארי, כאשר קודקודים פנימיים שלו הם פעולות חשבון ועלים הם מספרים. לדוגמה, העץ שבציור הבא מתאר את הביטוי:

+ 6 ) ( ( ( 4 + ( 3 \* 7 ) ) – ( 5 / ( 3 + 4 ) ) ) .

עץ בינארי שמתאר ביטוי אריתמטי נקרא expression tree.

שימו לב: אם נסרוק את עץ הביטויים בסריקת inorder, נקבל ביטוי אריתמטי בצורת infix, אם נסרוק את העץ בסריקת preorder נקבל ביטוי אריתמטי בצורת prefix, ואם נסרוק את העץ בסריקת postorder, נקבל ביטוי אריתמטי בצורת postfix.

**בתרגיל זה, בהינתן ביטוי בייצוג postfix, יש לבנות ממנו expression tree.**

**בנוסף, תשתמשו בעץ הביטויים כדי לחשב את ערך הביטוי.**

**האלגוריתם לבניית expression tree מביטוי בייצוג postfix**

1. הקלט:
   * ביטוי postfix הנתון כמחרוזת **כאשר מופיעים רווחים לפני ואחרי כל אופרנד, לפני ואחרי כל אופרטור, מלבד בהתחלה ובסוף הביטוי.** לדוגמה, "6 3 +" .
   * ביטוי הקלט מכיל אך ורק את הפעולות '^' , '/' , '\*' , '-' , '+'.
   * **ניתן להניח שקלט תקין**
2. הפלט: עץ בינארי (expression tree) המתאר את הביטוי.

האלגוריתם לבניית עץ הביטויים משתמש במחסנית עזר.

1. stack<- // create an empty stack
2. for each token in the input do
   1. read the next token
      1. if token is an operand
         1. node <- new leaf node with token
         2. stack.push(node)
      2. else if token is an operator
         1. right <- stack.pop()
         2. left <- stack.pop()
         3. node <- new node with token, left and right as its left and right children
         4. stack.push(node)
3. root <- stack.pop()

**תרגיל 1 (40 נקודות):**

* עליכם להשלים את הקוד של הפונקציה

**node\* create\_expr\_tree\_postfix(char \*postfix)**

הפונקציה מקבלת ביטוי בייצוג סופי, כאשר מופיעים רווחים לפני ואחרי כל אופרנד, ולפני ואחרי כל אופרטור, מלבד בהתחלה ובסוף הביטו. הפונקציה בונה expression tree שמייצג את הביטוי ומחזירה מצביע לשורש של העץ.

**תרגיל 2 (20 נקודות):**

* כמו כן, עליכם להשלים את הקוד של הפונקציות הרקורסיבית הבאות

**void inOrder(node\* treeNode)**

**void preOrder(node\* treeNode)**

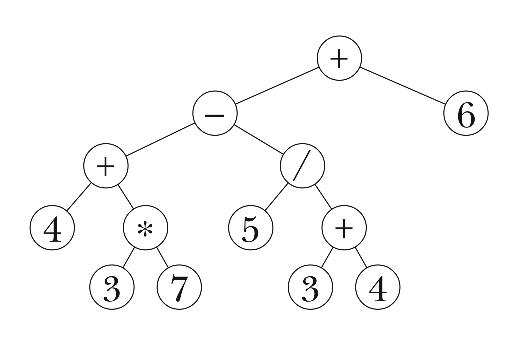
**void postOrder(node\* treeNode)**

להדפסת הביטוי בצורת prefix, infix ו-postfix

כאשר בפונקציות postOrder ו-preOrder יש להדפיס רווחים לפני ואחרי כל אופרנד, ולפני ואחרי כל אופרטור מלבד בהתחלה ובסוף הביטוי.(ראו דוגמאות למטה)

בפונקציה inOrder יש להדפיס רווחים לפני ואחרי כל אופרנד, ולפני ואחרי כל אופרטור, **ולפני ואחרי כל סוגר** מלבד בהתחלה ובסוף הביטוי. (ראו דוגמאות למטה)

**אין לשנות את החתימה של הפונקציות!**

לדוגמה, עבור עץ הביטויים הבא:

תוצאות ההדפסה:

inOrder: ( ( ( ( 4 + ( 3 \* 7 ) ) – ( 5 / ( 3 + 4 ) ) ) + 6)

preOrder: + - + 4 \* 3 7 / 5 + 3 4 6

postOrder: 4 3 7 \* + 5 3 4 + / - 6 +

**תרגיל 3 (40 נקודות):**

**הערכת (חישוב) ביטוי מתוך expression tree**

תהליך ההערכה הוא החלק הקל ביותר. בניגוד לתהליך ההמרה, אלגוריתמים להערכת הביטוי הם קצרים וקלים להבנה. הסיבה לכך היא שעכשיו אנחנו לא צריכים לדאוג לסדר העדיפות של פעולות.

את השיטה שמחשבת את ערך הביטוי יש לממש כשיטה **רקורסיבית**, שמקבלת הפנייה לצומת בעץ ומחשבת ערך הביטוי המיוצג על ידי תת עץ המושרש בצומת זה.

|  |  |  |
| --- | --- | --- |
| **Algorithm** evaluateExpression (Node *node*) | | |
|  | 1. if (*node* is an operator) |
|  | 1.1  *operand1* 🡨 evaluateExpression (*node*.left)   * 1. *operand2* 🡨 evaluateExpression (*node*.right) | | |

|  |  |  |
| --- | --- | --- |
|  | 1.3 *result* 🡨 *operand1 <operator> operand2* | |
|  | 1. else |
|  | 2.1 *result* 🡨 *node*.data // node contains a number  3. return *result* | | |

עליכם להשלים את הקוד של הפונקציה הרקורסיבית

**int evaluateExpression(node \*treeNode)**

הפונקציה מקבלת מצביע לצומת בעץ מחשבת ומחזירה את תוצאת החישוב של הביטוי.

**אין לשנות את החתימה של הפונקציה!**

|  |  |
| --- | --- |
|  |  |
|  |  |

**בהצלחה ועבודה נעימה !**