סדנת תכנות בשפת C ,מס׳ קורס 67316 -2018

תרגיל 3

תאריך הגשה: יום חמישי 29.11.18 עד שעה 23:55

הגשה מאוחרת (בהפחתת 10 נקודות): יום שישי 30.11.18 עד שעה 23:55

תאריך ההגשה של הבוחן: יום חמישי 29.11.18 עד שעה 23:55

הנחיות חשובות לכלל התרגילים:

- בכל התרגילים יש לעמוד בהנחיות הגשת התרגילים וסגנון כתיבת הקוד. שני המסמכים נמצאים באתר הקורס – הניקוד יכלול גם עמידה בדרישות אלו.
- בכל התרגילים עליכם לכתוב קוד ברור. בכל מקרה בו הקוד שלכם אינו ברור מספיק עליכם להוסיף הערות הסבר בגוף הקוד. יש להקפיד על תיעוד (documentation) הקוד ובפרט תיעוד של כל פונקציה.
- במידה ואתם משתמשים בעיצוב מיוחד או משהו לא שגרתי, עליכם להוסיף הערות בקוד המסבירות את העיצוב שלכם ומדוע בחרתם בו.
- עבור כל פונקציה בה אתם משתמשים, עליכם לוודא שאתם מבינים היטב מה הפונקציה עושה גם במקרי קצה (התייחסו לכך בתיעוד). ובפרט עליכם לוודא שהפונקציה הצליחה.
- בכל התרגילים במידה ויש לכם הארכה, או שאתם מגישים באיחור. **חל איסור להגיש קובץ כלשהוא בלינק הרגיל (גם אם לינק overdue טרם נפתח). מי שיגיש קבצים בשני הלינקים מסתכן בהורדת ציון משמעותית.**
- אין להגיש קבצים נוספים על אלו שתדרשו .ובפרט אין להגיש קובץ README אלא אם צוין במפורש שיש צורך בכך (לדוגמא, בתרגיל זה אין צורך להגיש).
- עליכם לקמפל עם הדגלים Wall -Wextra -Wvla -std=c99 ולוודא שהתוכנית מתקמפלת ללא אזהרות, **תכנית שמתקמפלת עם אזהרות תגרור הורדה משמעותית בציון התרגיל.**
- עליכם לוודא שהתרגילים שלכם תקינים ועומדים בכל דרישות הקימפול והריצה במחשבי בית הספר מבוססי מעבדי bit-64 (מחשבי האקווריום, לוי, השרת river). **חובה להריץ את התרגיל במחשבי בית הספר לפני ההגשה**. (ניתן לוודא שהמחשב עליו אתם עובדים הנו בתצורת bit-64 באמצעות הפקודה " uname -a" ווידוא כי הארכיטקטורה היא 64, למשל אם כתוב x86_64)
- לאחר ההגשה, בדקו את הפלט המתקבל בקובץ ה-PDF שנוצר מהpresubmission script בזמן ההגשה. באם ישנן שגיאות, תקנו אותן על מנת שלא לאבד נקודות.
שימו לב ! תרגיל שלא יעבור את ה presubmission script ציונו ירד משמעותית (הציון יתחיל מ-50, ויוכל לרדת) **ולא יהיה ניתן לערער על כך.**
- בדיקת הקוד לפני ההגשה, גם על ידי קריאתו וגם על ידי כתיבת בדיקות אוטומטיות (tests) עבורו היא אחריותכם. בדקו מקרי קצה.
במידה וסיפקנו לכם קבצי בדיקה לדוגמא, השימוש בהם יהיה על אחריותכם. **במהלך הבדיקה הקוד שלכם ייבדק מול קלטים נוספים לשם מתן הציון.**

מחשבון

לעתים קרובות אנו מתמודדים עם ביטויים אריתמטיים שנכתבו במה שמכונה סימון infix:

```
Operand1 operator Operand2
```

יש לנו חוקים לקדימות של פעולות (למשל כפל וחילוק קודמים לחיבור וחיסור), ולעתים קרובות אנו משתמשים בסוגריים כדי לעקוף את הכללים האלה.

ניתן גם לכתוב ביטויים באמצעות סימון postfix:

```
Operand1 Operand2 operator
```

לדוגמה, ביטוי בסימון infix:

```
7 * ( 2 + 6 ) - 15 / 3
```

יראה כך בסימון postfix:

```
7 2 6 + * 15 3 / -
```

בהינתן ביטוי בסימון postfix, ניתן להשתמש במחסנית כדי למצוא את הערך הכולל של הביטוי.
לכן המחשבון שלנו מצריך שני אלגוריתמים:

1. המרה מ infix ל- postfix
2. שיערוך ביטוי בסימון postfix

שני האלגוריתמים משתמשים במחסנית. לצורך כך עליכם לבנות ספריה עם קוד עבור מחסנית גנרית שראינו בשיעור ולהשתמש בה.

הנחות:

- קלט תקין, מוגבל ל 100 תווים לשורה
- כל האופרטורים הם בינאריים, כלומר פועלים על 2 מספרים (ללא אופרטורים אונריים כמו ++)
- כל האופרנדים הם מספרים מסוג int
- האופרטורים שצריך לתמוך בהם: +, -, *, /, ^ (חזקה)

1. המרה מ infix ל - postfix

נניח שנתון ביטוי אריתמטי בסימון infix. נסמן אותו ב- Q. Q הוא למעשה עיבוד של מחרוזת קלט למבנה נתונים שמפרק אותה לאופרטורים, מספרים וסוגריים. P הוא פלט, כלומר ביטוי אריתמטי בסימון postfix. נתחיל עם מחסנית ריקה ונסרוק את Q משמאל לימין לפי הפסאודו-קוד הבא:

```
While (we have not reached the end of Q)
     If (an operand is found)
        Add it to P
     End-If
     If (a left parenthesis is found)
        Push it onto the stack
```

```
        End-If
        If (a right parenthesis is found)
           While (the stack is not empty AND
                  the top item is not a left parenthesis)
              Pop the stack and add the popped value to P
           End-While
           Pop the left parenthesis from the stack and discard it
        End-If
        If (an operator is found)
           If (the stack is empty OR
               if the top element is a left parenthesis)
              Push the operator onto the stack
           Else
              While (the stack is not empty AND
                     the top of the stack is not a left parenthesis AND
                     precedence of the operator <= precedence of the top of
                     the stack)
                 Pop the stack and add the top value to P
              End-While
              Push the latest operator onto the stack
           End-If
        End-If
     End-While
     While (the stack is not empty)
        Pop the stack and add the popped value to P
     End-While
```

2. שיערוך ביטוי בסימון postfix

נניח שנתון ביטוי אריתמטי בסימון postfix. נסמן אותו ב- P. נשערך את הביטוי תוך שימוש במחסנית לשמירת אופרנדים. נתחיל עם מחסנית ריקה ונסרוק את P משמאל לימין לפי הפסאודו-קוד הבא:

```
While (we have not reached the end of P)
        If an operand is found
           push it onto the stack
        End-If
        If an operator is found
           Pop the stack and call the value A
           Pop the stack and call the value B
           Evaluate B op A using the operator just found.
           Push the resulting value onto the stack
        End-If
```

```
End-While
Pop the stack (this is the final value)
```

עליכם לממש תוכנית שקוראת מ **stdin** ביטוים אריתמטיים בסימון infix בתור מחרוזות עד EOF.

1. התוכנית שלכם תמיר כל מחרוזת לביטוי אריתמטי Q בסימון infix. עליכם לתכנן את מבנה הנתונים המתאים. ביטוי אריתמטי יכול להיות מערך של איברים שמחזיקים אינפורמציה (מהו האיבר: אופרטור, אופרנד או סוגריים ומה הערך שלו).
2. לאחר מכן אתם ממירים את הביטוי מסימון infix ל- postfix בעזרת האלגוריתם השני ומדפיסים את הביטוי
3. לבסוף עליכם לשערך את הביטוי בסימון postfix בעזרת האלגוריתם השני ולהדפיס את הערך.

לדוגמה:

```
> mycalc
77
Infix:  77
Postfix:  77
The value is 77
17^2
Infix:  17 ^ 2
Postfix:  17  2 ^
The value is 289
3+8*2
Infix:  3 + 8 * 2
Postfix:  3  8  2 *+
The value is 19
(3+8)*2
Infix: ( 3 + 8 )* 2
Postfix:  3  8 + 2 *
The value is 22
```

<u>מידע נוסף (כללי)</u>

- חל איסור להשתמש במערכים בגודל דינמי (VLA).
- אתם רשאים להשתמש בכל הספריות הסטנדרטיות של C.
- אתם רשאים (ולעתים אף נדרשים) להגדיר פונקציות נוספות לשימושכם הפנימי.
- שימו לב שאתם מכירים כל פונקציה בה אתם משתמשים ושאתם בודקים עבור כל פונקציה שהיא הצליחה.
- **עליכם לוודא שהקוד שלכם רץ באופן תקין וללא דליפות זכרון. לשם כך עליכם להשתמש בתוכנת valgrind (מדריך במודל)**

1. **<u>טיפול בשגיאות:</u>**

- הדפסות שגיאה יודפסו אל הstderr, כמו כן נדרש להבדיל בין שגיאה בתוכנה לבין קלט לא תקין וכדומה. קובץ לא קיים למשל היינה שגיאה בתוכנה שיש לטפל בה ולהחזיר הודעת שגיאה.
- עבור מספר ארגומנטים שגוי יש להדפיס מידע המסביר כיצד להריץ את הקוד (usage) ולצאת עם ערך חזרה שונה מ 0.
- מותר להשתמש ב exit, ואין צורך לשחרר זיכרון במקרה של יציאה בעקבות שגיאה

2. **בדיקת התרגיל:**

- התכניות יבדקו גם על סגנון כתיבת הקוד וגם על פונקציונאליות, באמצעות קבצי קלט שונים (תרחישים שונים להרצת התכניות). לכן עליכם להקפיד על פורמט הדפסה מדויק, כדי למנוע שגיאות מיותרות והורדת נקודות, ראו שימוש ב diff.
- אם ישנם מקרים שהוראות התרגיל לא מציינות בבירור כיצד התכנית צריכה להתנהג, הביטו בקבצי הקלט וקבצי הפלט לדוגמה שניתנים לכם ובדקו אם התשובה לשאלתכם נמצאת שם. כמו כן, היעזרו בפתרון בית הספר, הריצו עליו את הטסטים שלכם והשוו להתנהגות תוכניתכם. כמובן שניתן וכדאי להתייעץ בפורום לגבי מקרים שבהם התשובה עדיין אינה ברורה.

3. **חומר עזר:**

- קבצי בדיקה לדוגמא ניתן למצוא ב:

```
~labc/www/ex3/files/
```

- מותר ואף רצוי להשתמש ב `diff` שבמחשבי האקוריום עבור השוואת פלטים (הסבר מפורט בסוף הקובץ).

4. **הגשה:**

- עליכם להגיש קובץ tar בשם `ex3.tar` המכיל רק את הקבצים הבאים:
  - Makefile stack.h stack.c
  - כל שאר קבצי קוד שיצרתם

ניתן ליצור קובץ tar כדרוש על ידי הפקודה:

```
>tar -cvf ex3.tar file1 file2 ...
```

- לפני ההגשה, פתחו את הקובץ ex1.tar בתיקיה נפרדת וודאו שהקבצים מתקמפלים ללא שגיאות וללא אזהרות. וודאו שההגשה שלכם עוברת את ה-presubmission script ללא שגיאות או אזהרות.

```
~labc/www/ex3/presubmit_ex3
```

- אתם יכולים להריץ בעצמכם בדיקה אוטומטית עבור סגנון קידוד בעזרת הפקודה:

```
~labc/www/codingStyleCheck <code file or directory>
```

כאשר <directory or file> מוחלף בשם הקובץ אותו אתם רוצים לבדוק או תיקייה שיבדקו כל הקבצים הנמצאים בה (שימו לב שבדיקה אוטומטית זו הינה רק חלק מבדיקות ה codingStyle)

- דאגו לבדוק לאחר ההגשה את קובץ הפלט (submission.pdf) וודאו שההגשה שלכם עוברת את ה-presubmission script ללא שגיאות או אזהרות.

<u>שימוש בפקודת diff:</u>

- לרשותכם כמה קבצי קלט לדוגמה וקבצי הפלט המתאימים להם (אלו מהווים רק חלק קטן מקבצי הקלט-פלט שנשתמש בהם, כתבו לעצמכם בדיקות נוספות). עליכם לוודא שהתכנית שלכם נותנת את אותו הפלט בדיוק.
- על מנת לעשות זאת הריצו את תכניתכם עם הקלט לדוגמה על ידי ניתוב ה standard input להקרא מקובץ (באמצעות האופרטור ">" בשורת ההרצה ב terminal), ונתבו את הפלט של תכניתכם, שהוא ה standard output,לתוך קובץ (באמצעות האופרטור "<") באופן הבא:

```
prog_name < in_file > out_file
```

- השוו את קובץ הפלט שנוצר לכם עם קובץ הפלט המתאים של פתרון בית הספר, באמצעות הפקודה diff להשוואת טקסטים.

תיאור diff: בהינתן שני קבצי טקסט להשוואה (1.txt ,2.txt) הפקודה הבאה תדפיס את השורות אשר אינן זהות בשני הקבצים:

```
diff 1.txt 2.txt
```

במידה והקבצים זהים לחלוטין, לא יודפס דבר.

קראו על אפשרויות נוספות של diff בעזרת הפקודה man diff. לחלופין אתם יכולים גם להשתמש בתוכנה tkdiff אשר מראה גם את השינויים ויזואלית.

כמו כן, אתם יכולים גם להשוות ישירות באופן הבא:

```
prog_name < in_file | diff expected.out
```

בהצלחה!