הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל
מבוא למערכות תוכנה

תרגיל בית 3 - ADT

מועד ההגשה:	יום חמישי, 27/12/18 בשעה 23:50	
האחראי על התרגיל:	איתי לוי	itaylevi@campus.technion.ac.il

המטרה:

מימוש ושימוש ב –ADT.

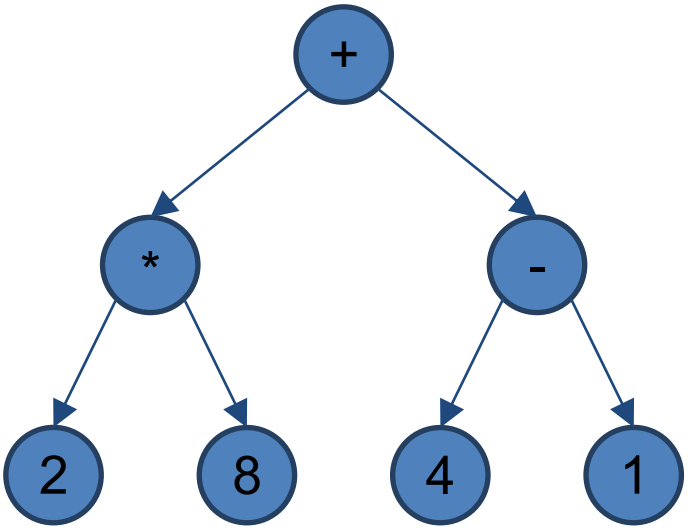
התרגיל:

מימוש ADT של עץ ביטוי בינארי והשימוש בו.

הסבר כללי:

עץ ביטוי בינארי הינו מבנה נתונים בו לכל צומת יש אב יחיד ולכל היותר 2 בנים, כאשר עלי העץ מייצגים אופרנדים ושאר צמתי העץ מייצגים אופרטורים.

לדוגמא, ניתן להסתכל על העץ הבא המייצג ביטוי אריתמטי:



כאשר האופרנדים הם: "2", "8", "4" ו "1", והאופרטורים הם: "+", "*" ו "-".

ניתן לראות כי העץ הנ"ל מייצג את הביטוי: "(2 * 8) + (4 – 1)".

שימו לב כי עץ ביטוי אינו מוגבל לביטויים אריתמטיים וניתן להשתמש במבנה זה גם לצורך פעולות לוגיות או ייצוג אופרטורים על קבוצות.

הצמתים בעץ יהיו מטיפוס כללי (void*) כאשר על האופרטורים מוגדרת פעולת "Operate" אשר פועלת על שני אופרנדים.

כמו כן, לכל צומת בעץ, קיים מפתח מסוג void* ופונקציית השוואת מפתחות המאפשרת חיפוש של איבר בעץ.

הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל
מבוא למערכות תוכנה

חלק א':

בחלק זה נממש ADT בשם ExpTree שייצג עץ ביטוי לפי התיאור הנ"ל.על מנת שהמבנה יוכל לעבוד עם צמתים כלליים, עליו לקבל מהמשתמש מצביעים לפונקציות מהטיפוסים הבאים (ראו expTree.h):

```
typedef void* pElement;
typedef void* pKey;

typedef pElement     (*CloneFunction)(pElement e);
typedef void         (*DelFunction)(pElement e);
typedef pElement     (*OperateFunction)(pElement op,
                                        pElement left,
                                        pElement right);
typedef pKey         (*GetKeyFunction)(pElement elem);
typedef Bool         (*CompareKeyFunction)(const pKey key1,
                                           const pKey key2);
```

כאשר:

- $CloneFunction$: מקבלת מצביע לאלמנט ומחזירה מצביע לעותק חדש שלו.
- $DelFunction$: מקבלת מצביע לאלמנט ומשחררת את הזיכרון שהוא תופס.
- $OperateFunction$: מקבלת מצביעים לשלושה אלמנטים ומפעילה את האלמנט הראשון עלהשניים הבאים. הפונקציה תחזיר את התוצאה באלמנט חדש שהיא תיצור בעצמה.
- $GetKeyFunction$: מקבלת אלמנט ומחזירת את המפתח שלו.
- $CompareKeyFunction$: מקבלת שני מפתחות ומחזירה האם הם זהים או לא.

שימו לב כי לא כל הפונקציות מוגדרות עבור כל סוגי האלמנטים ולכן יש לבצע בדיקת תקינות לגבי טיפוס המשתנים בתוך הפונקציות.

להלן פירוט פונקציות הממשק של העץ הכללי:

- $TreeCreate$: מקבלת פונקציות מהטיפוסים שהוגדרו לעיל ומחזירה מצביע למבנה ריק.
- $TreeDestroy$: מקבלת כפרמטר מצביע לעץ ומשחררת את כל הזיכרון שהמבנה תופס. הפונקציה אינה מחזירה דבר.
- $TreeAddRoot$: מקבלת כפרמטר מצביע לעץ ומצביע לאלמנט חדש, ומוסיפה צומת חדש בתור שורש העץ המכיל את האלמנט החדש.
  הפונקציה תחזיר מצביע לשורש העץ שנוסף.
  הפונקציה תחזיר NULL במקרה של כישלון.
- $TreeAddLeftChild$: מקבלת כפרמטר מצביע לעץ, מצביע לצומת בעץ ומצביע לאלמנט חדש, ומוסיפה צומת חדש בתור בן שמאלי של הצומת שהועבר.
  הפונקציה תחזיר מצביע לצומת החדש שנוסף.
  הפונקציה תחזיר NULL במקרה של כישלון.

- $TreeAddRightChild$: מקבלת כפרמטר מצביע לעץ, מצביע לצומת בעץ ומצביע לאלמנט חדש, ומוסיפה צומת חדש בתור בן ימני של הצומת שהועבר.
  הפונקציה תחזיר מצביע לצומת החדש שנוסף.
  הפונקציה תחזיר NULL במקרה של כישלון.
- $TreeFindElement$: מקבלת כפרמטר מצביע לעץ, ומפתח של צומת בעץ. הפונקציה תחפש בעץ אלמנט המתאים למפתח ותחזיר מצביע לאלמנט.
  אם האלמנט לא נמצא, הפונקציה תחזיר NULL.
- $TreeEvaluate$: מקבלת כפרמטר מצביע לעץ. הפונקציה תעבור על העץ באופן רקורסיבי ותחשב את ערך הביטוי שהעץ מייצג. הפונקציה תחזיר אלמנט חדש המייצג את ערך הביטוי.
  הפונקציה תחזיר NULL במקרה של כישלון.

הערות:

1. בהוספת צומת חדש, יש ליצור **עותק חדש** של הצומת ולהכניסו למבנה. שימו לב גם לטיפוס של הפונקציה המתאימה $CloneFunction$.
2. פונקציות המבצעות הקצאה של זיכרון כדוגמת $TreeAddRoot$, צריכות לוודא את הצלחת ההקצאות, ובמקרה של כישלון, על הפונקציה להחזיר NULL.
3. הפונקציה $TreeFindElement$ תחזיר מצביע לאלמנט הקיים בעץ ולא תיצור אלמנט חדש באמצעות $CloneFunction$.
4. טיפוסי חזרה אשר תדרשו להם מוגדרים בקובץ ***defs.h*** שמסופק לכם.

<u>מטלות חלק א:</u>

נתונים לכם הקבצים הבאים:

- *defs.h* – קובץ הגדרות טיפוסים חיוניים. **<u>אין לשנות קובץ זה.</u>**
- *expTree.h* – קובץ ממשק של העץ.
- *expTree.c* – קובץ מימוש של העץ.

עליכם להשלים את המימוש של העץ בקבצים *expTree.h* ו-*expTree.c* על פי הדרישות הנ"ל.

<u>חלק ב':</u>

בחלק זה נעשה שימוש בעץ שמימשנו בחלק א' על מנת לממש מחשבון סימבולי.

על מנת להקל על שלב ניתוח הקלט, אנו נשתמש בייצוג מיוחד לביטויים אלגבריים הנקרא "prefix notation" או "polish notation".
(ניתן לקרוא על הייצוג כאן: http://en.wikipedia.org/wiki/Polish_notation)

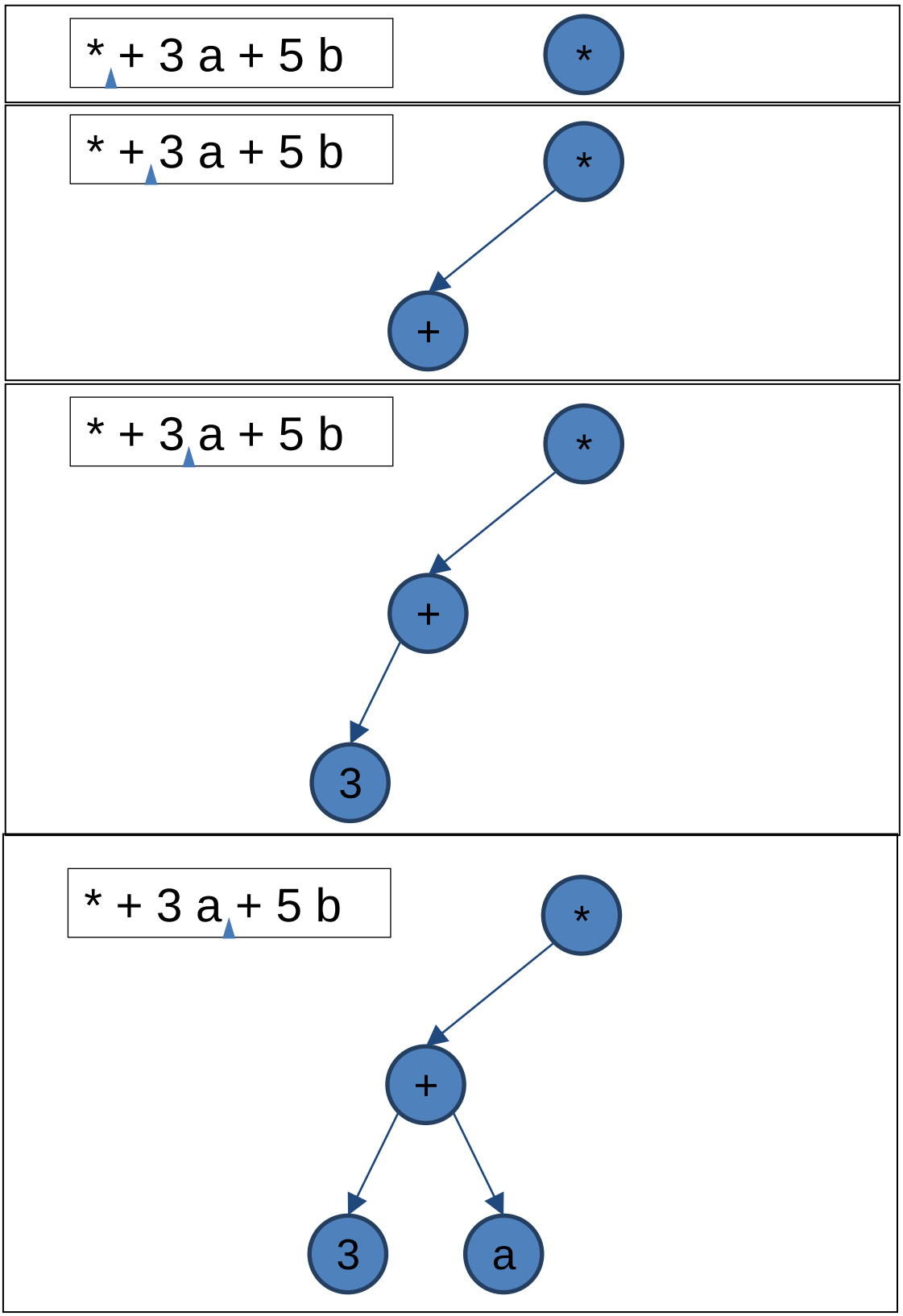
בייצוג זה האופרטורים מופיעים לפני האופרנדים עליהם הם עובדים, לדוגמא:

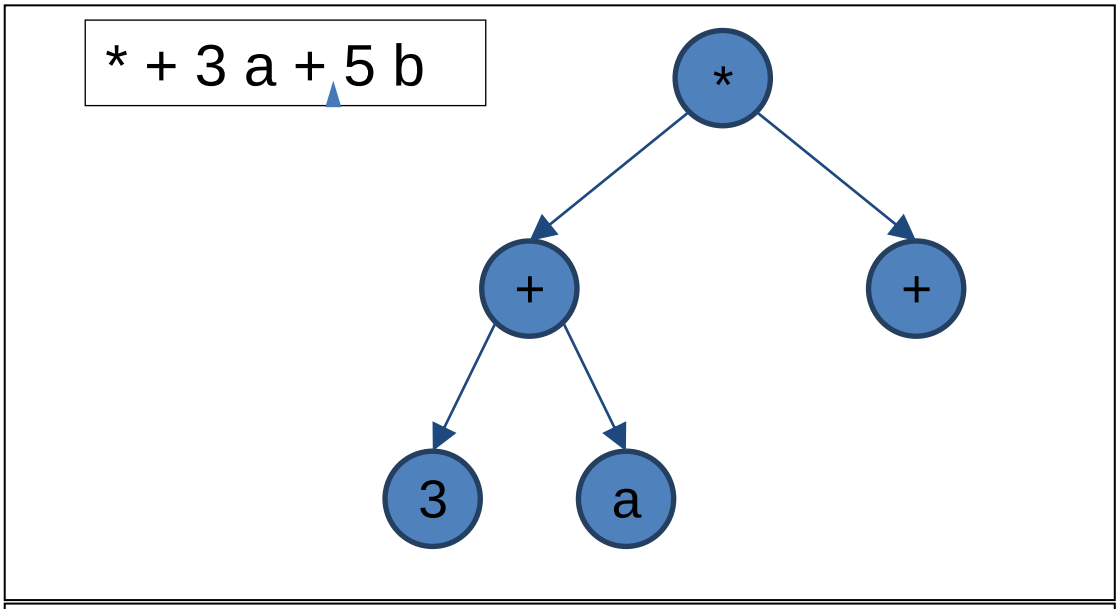
ייצוג "רגיל"	Polish notation
3 + a	+ 3 a
(3 + a) * 5	* + 3 a 5
(3 + a) * (5 + b)	* + 3 a + 5 b

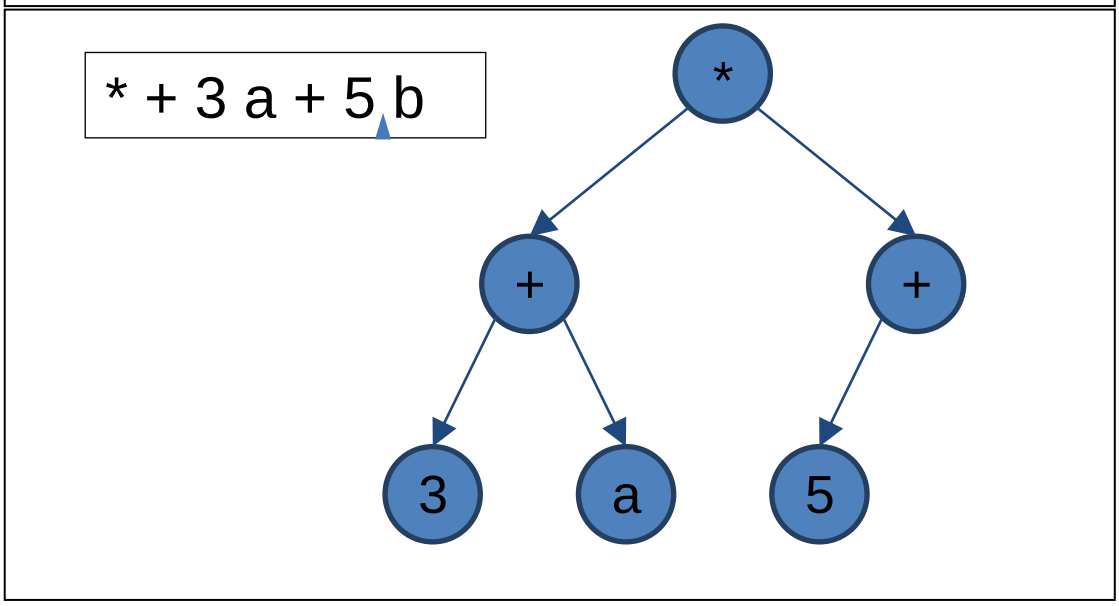
כפי שניתן לראות, בייצוג זה אין התייחסות לסדר פעולות חשבון, ואין צורך בשימוש בסוגריים, דבר המקל מאוד על הניתוח של הביטוי.

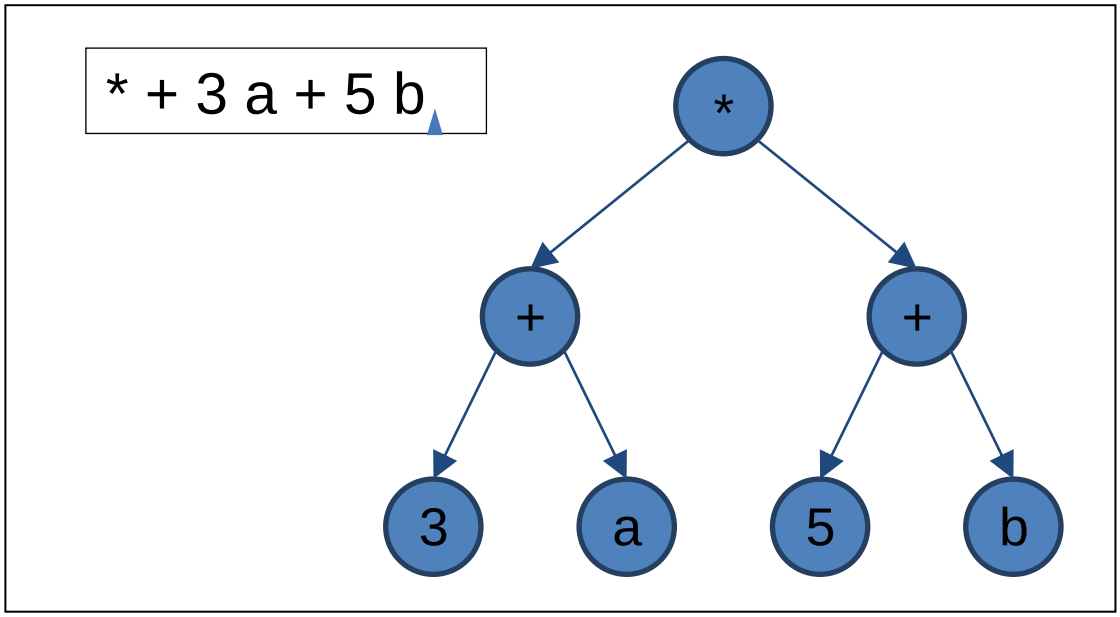
יתרון נוסף של השימוש בייצוג זה בהקשר של עצי ביטוי הוא העובדה שסדר האלמנטים בקלט, תואם את סדר היצירה של עץ הביטוי.
האלמנט הראשון שנקרא מהקלט תמיד יהיה שורש העץ, האלמנט הבא יהיה הבן הימני שלו (ראו דוגמא) וניתן להמשיך בתהליך זה באופן רקורסיבי לשם יצירת עץ הביטוי.

נדגים זאת על הביטוי האחרון בטבלה:




* + 3 a + 5 b
*
+
+
3
a


* + 3 a + 5 b
*
+
+
3
a
5


* + 3 a + 5 b
*
+
+
3
a
5
b

להלן פונקציות הממשק של החבילה polishCalc אותה נממש:

- $InitExpression$: מקבלת מחרוזת המייצגת ביטוי חשבוני כאשר האלמנטים מופרדים בתו רווח בודד, ומאתחלת עץ ביטוי מתאים.
- $SetSymbolVal$: מקבלת שם של סימבול וערך חדש ומעדכנת את האלמנט המתאים בעץ הביטוי.
- $EvaluateExpresion$: מחשבת את הביטוי בעץ הביטוי ומחזירה את התוצאה.
- $DeleteExpresion$: מוחקת את עץ הביטוי ומשחררת את הזיכרון שהוקצה.

הערות:

1. אם בזמן הקריאה ל $InitExpression$ כבר קיים עץ ביטוי, הפונקציה תשחרר אותו ע"י קריאה ל $DeleteExpresion$.
2. משתנים סימבולים יאותחלו ל – 0.
3. במקרה של ביטוי לא חוקי מהקלט, יש לבדוק זאת בזמן הקריאה ל $InitExpression$.

<u>מטלות חלק א:</u>

נתונים לכם הקבצים הבאים:

- *main.c* – קובץ התכנית הראשית שמשתמשת ב polishCalc. **<u>אין לשנות קובץ זה.</u>**
- *polishCalc.h* – קובץ הממשק של החבילה PolishCalc. **<u>אין לשנות קובץ זה.</u>**
- polishCalcTypes.h – קובץ עזר המכיל הגדרות טיפוסים עבור החבילה PolishCalc. **<u>אין לשנות קובץ זה.</u>**

השלימו את המימוש של החבילה בקובץ polishCalc.c, כך שהתכנית הראשית תתקמפל ותעבוד כשורה. במימוש יש להשתמש ב-ADT של עץ הביטוי שכתבתם בחלק א'. בחלק זה מותר להגדיר מצביע גלובלי לעץ שיכיל את הביטוי.

על מנת לבנות את התכנית, עליכם לכתוב קובץ makefile מתאים. **שם תכנית ההרצה שתיווצר חייב להיות: calc**.

הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל
מבוא למערכות תוכנה

הנחיות הגשה:

1. קבצי קוד חלקיים, וכן קבצי קלט ופלט לדוגמה, נמצאים בתיקייה:
~eesoft/hmw/hmw3
לפני תחילת העבודה, הורידו את הקבצים לחשבונכם באמצעות הפקודה:
cp ~eesoft/hmw/hmw3/*.
2. עברו היטב על הוראות ההגשה של תרגילי הבית המופיעים באתר טרם ההגשה! ודאו כי התכנית שלכם עומדת בדרישות הבאות:
   - התכנית קריאה וברורה
   - התכנית מתועדת היטב לפי דרישות התיעוד המופיעות באתר
   - התכנית מתקמפלת ללא שגיאות וללא warnings כלל
   - התכנית רצה **ללא דליפות זיכרון** וגישות לא חוקיות לזיכרון כלל (בדיקה באמצעות valgrind)
   - התכנית נותנת פלט **זהה לחלוטין** לפלט הצפוי על כל קבצי הקלט שסופקו (בדיקה באמצעות פקודת diff על קבצי הפלט)
   - קובץ ה-makefile יוצר קובץ הרצה בשם הנדרש
3. יש להגיש קובץ tar יחיד המכיל את **כל הקבצים** שאתם נדרשים להגיש **ואותם בלבד** – ללא תתי-תיקיות. ודאו כי לא שכחתם את קובץ readme המכיל את פרטי הסטודנטים, וכן את ה-makefile במידה ונדרשתם.
4. שאלות בנוגע לתרגיל יש להפנות לפורום התרגיל ב-moodle בלבד – ניתן לשלוח שאלות במייל **למתרגל האחראי על התרגיל בלבד**, ורק במידה והשאלה מכילה פתרון חלקי.
5. סיכום מפרט התרגיל:

סעיף	**תיאור**
נושא התרגיל	ADT
תאריך ההגשה	יום חמישי, 27/12/18 בשעה 23:50
המתרגל האחראי על התרגיל	איתי לוי itaylevi@campus.technion.ac.il
תיקייה המכילה קבצים לשימוש הסטודנטים	~eesoft/hmw/hmw3
קבצי הקוד הנתונים	defs.h expTree.h expTree.c polishCalc.h polishCalcTypes.h main.c
קבצי הקלט והפלט הנתונים	input1.txt output1.txt input2.txt output2.txt
הקבצים שיש להגיש	readme makefile expTree.h expTree.c polishCalc.c
שם תכנית ההרצה הדרושה (הנוצרת ע"י makefile)	calc

בהצלחה!