תורת הקומפילציה
תרגיל 5

ההגשה בזוגות בלבד

שאלות במייל בלבד אל omerkatz@cs

לתרגיל יפתח דף FAQ באתר הקורס. יש להתעדכן בו. הנחיות והבהרות שיופיעו בדף ה-FAQ עד יומיים לפני הגשת התרגיל מחייבות. שאלות המופיעות בדף ה-FAQ לא יענו.

התרגיל ייבדק בבדיקה אוטומטית. **הקפידו למלא אחר ההוראות במדויק.**

1. כללי

בתרגיל זה עליכן לממש תרגום לאסמבלי לשפת FanC שהכרתם בתרגיל בית 3. תחביר השפה הוא כפי שהכרתן ומימשתן בתרגיל בית 3. בתרגיל תממשנה את רשומת ההפעלה של הפונקציות, ואת מבני הבקרה תממשנה בשיטת **backpatching**.

2. MIPS

בתרגיל תשתמשו בשפת האסמבלי MIPS עבור הסימולטור Spim. השפה מכילה רגיסטרים, פעולות אריתמטיות בין רגיסטרים, קפיצות מותנות ולא מותנות, וקריאות מערכת. תמצאו מפרט מלא של השפה כאן: http://www.egr.unlv.edu/~ed/MIPStextSMv11.pdf

תוכלו לדבג את קוד האסמבלי שאתם מייצרים על ידי הדיבאגר של QtSpim. גרסאות ווינדוס, לינוקס ומאק שלו מצורפות לתרגיל, יחד עם גרסת Spim.

א. פקודות אפשריות

בשפת MIPS יש מספר גדול מאוד של פקודות. בתרגיל תרצו להשתמש בפקודות המדמות את שפת הרביעיות שנלמדה בתרגול. להלן הפקודות שתרצו להשתמש בהן.

1. טעינה לרגיסטר: lw, li ו-la
2. שמירת תוכן רגיסטר: sw
3. פעולות חשבוניות: add, sub, mul, div, move
4. קפיצות מותנות: beq, blt, ble, bgt, bge, bne
5. קפיצה לא מותנית: j
6. קפיצה לא מותנית המעדכנת את $ra: jal
7. פקודה ריקה: nop

תוכלו למצוא תיעוד של כולן במדריך MIPS לעיל.

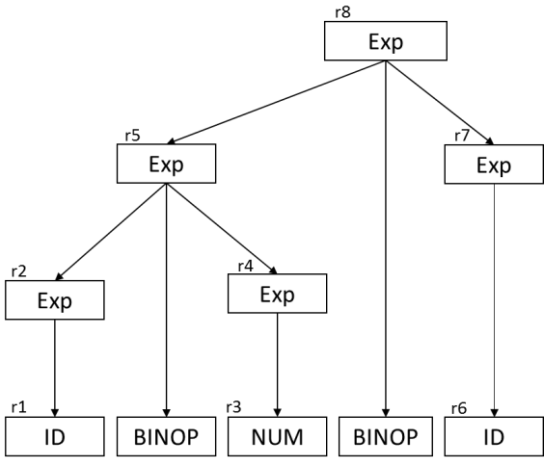
במידה ותרצו להשתמש בפקודות נוספות - מותר להשתמש בכל מה ש-Spim מסוגל להריץ (יש לבדוק את עצמכם עם הגרסה המצורפת לתרגיל, 9.1.18, ולא באף גרסה אחרת).

ב. רגיסטרים זמניים

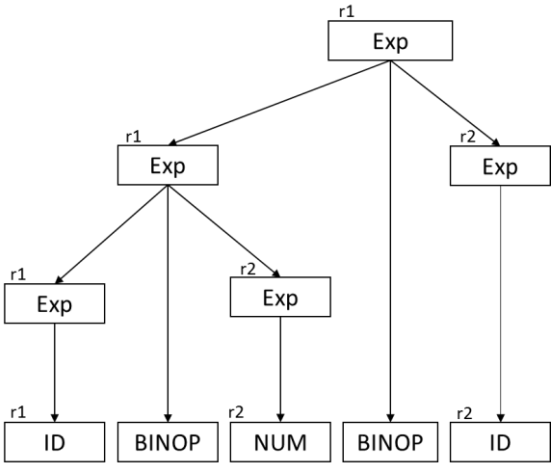
שימו לב כי ב-MIPS אין משתנים זמניים ולכן העבודה עם ביטויים מספריים בתרגיל תהיה ברגיסטרים. בתור רגיסטרים זמניים לפעולות ניתן להשתמש בכל הרגיסטרים שאינם שמורים

למטרות אחרות - $t0-$t7, $s0-$s7, ו-$t8-t9 (שהם שמות נוספים לרגיסטרים $8-$25). אין צורך לבצע אלגוריתמים מתקדמים של הקצאת רגיסטרים, אבל עליכן להיות חסכוניות.

מספר הרגיסטרים המקסימלי שאפשר להקצות לעץ ביטוי הוא כמספר הצמתים בעץ, כך:



לשם המימוש החסכוני עליכן למחזר רגיסטרים ברגע שאינם נחוצים יותר. במימוש החסכוני יש להשתמש ברגיסטר של אחד הבנים בתור רגיסטר המטרה, וברגע שהחישוב בוצע יש למחזר את כל הרגיסטרים שהחזיקו ביטויים זמניים פרט לזה שכעת מחזיק את התוצאה. בשיטה זו מספר הרגיסטרים יהיה חסום ב-$h(k-1)+1$, כאשר h הוא גובה העץ ו-k הוא מספר תתי-הביטויים המקסימלי של צומת כך העץ הקודם יראה כעת כך:



הביטויים שתידרשו לטפל בהם גדולים מדי בשביל שיטת ההקצאה הנאיבית. תוכלו להניח שלא תקבלו ביטוי שידרוש יותר מ-15 רגיסטרים בהקצאה חסכונית, אבל עליכן לטפל **בכל** ביטוי עד גודל זה.

אין להשתמש ברגיסטרים לאחסון ערכי ביטויים בוליאניים בזמן שערוך הביטוי (דרישה זו תובהר בפרק הסמנטיקה).

ג. לייבלים (תוויות קפיצה)

ב-MIPS יעדים של קפיצות מסומנים על ידי לייבלים אלפאנומריים (+ קו תחתון) כך:

```
label_42:
lw $t6,0($sp)
```

כאשר קפיצה אל label_42 תקפוץ אל הפקודה שנמצאת שורה אחריה – במקרה הזה, lw. בעת הטעינה לסימולטור SPIM הקפיצות יתורגמו לקפיצות לכתובות של פקודות, כפי שראינו בשיעור. את

הלייבלים בפקודות קפיצה של מבני בקרה יש לייצר ולהשלים בשיטת backpatching כפי שנלמד בשיעור. נתונה לכן המחלקה bp בקובץ bp.hpp עם מימוש של באפר ופונקציה המבצעת backpatching. נמצאת בה הפונקציה ()next שתכתוב לייבל חדש לבאפר ותחזיר אותו. אין חובה להשתמש בה, ניתן לנהל את הלייבלים שלכן בעצמכן. אפשר להשתמש בפונקציית bp המבצעת backpatching עם כל לייבל שתבחרו. כמו כן, ניתן לכתוב לבאפר גם לייבלים אחרים.

ג. 1. גוף הפונקציה כיעד קפיצה

שימו לב שבעוד שניתן להשלים את הקפיצות לפונקציות נקראות על ידי backpatching, כאן מדובר על מקרה בו יעד הקפיצה ידוע מראש: אם התכנית תקינה אזי הפונקציה קיימת ולכן יהיה לייבל עבור הפונקציה, ומכיוון ששם הפונקציה הוא ייחודי, אפשר לייצר לייבל ששמו ידוע מראש. כלומר, אפשר לקבוע את יעד הקפיצה בקפיצה לפונקציה ללא שימוש ב-backpatching.

ג.2. עבודה עם המחלקה CodeBuffer

לצורך העבודה עם באפר הקוד נתונה לכם מחלקה CodeBuffer בקובץ bp.hpp. במחלקה תוכלו למצוא מתודות לעבודה עם באפר קוד וביצוע backpatching, ומתודה שתדפיס את באפר הקוד ל-stdout.

המחלקה מממשת את הפונקציות emit, makelist ו-merge בהן ניתן להשתמש כפי שראיתן בתרגולים 7 ו-8.

הפונקציות מטפלות ברשימת כתובות בבאפר הקוד. ניתן להשתמש בכתובות אלה לצורך דיבוג הבאפר.
שימו לב, ערך החזרה של הפונקציה emit הוא הכתובת אליה כתבתם. זו הכתובת בה עליכם להשתמש לצורך backpatching. כלומר, פקודת הקוד הבאה הינה חוקית:

```
CodeBuffer::makelist(CodeBuffer::instance().emit("j "));
```

ד. דאטה

תכנית MIPS נחלקת לאזור text ואזור data. בעוד שקוד האסמבלי שתייצרו שייך לאזור ה-text, כמו גם המימוש של הפונקציות print ו-printi, אזור ה-data מיועד לשמירת ערכים שיש לטעון לזיכרון כדי להשתמש בהם אחר כך. באזור זה ניתן לשמור ליטרל מחרוזת (ראו דוגמאות במדריך MIPS).

המחלקה CodeBuffer מכילה מתודה להדפסת באפר הקוד בתוספת הצהרה על אזור text, ומכילה בנוסף לכך גם שתי מתודות לטיפול באיזור ה-data: המתודה emitData שתכתוב שורות לבאפר נפרד, והמתודה printDataBuffer שתדפיס את תוכן הבאפר הנפרד בתוספת הצהרה על אזור data. מומלץ להשתמש בהן.

3. מחסנית

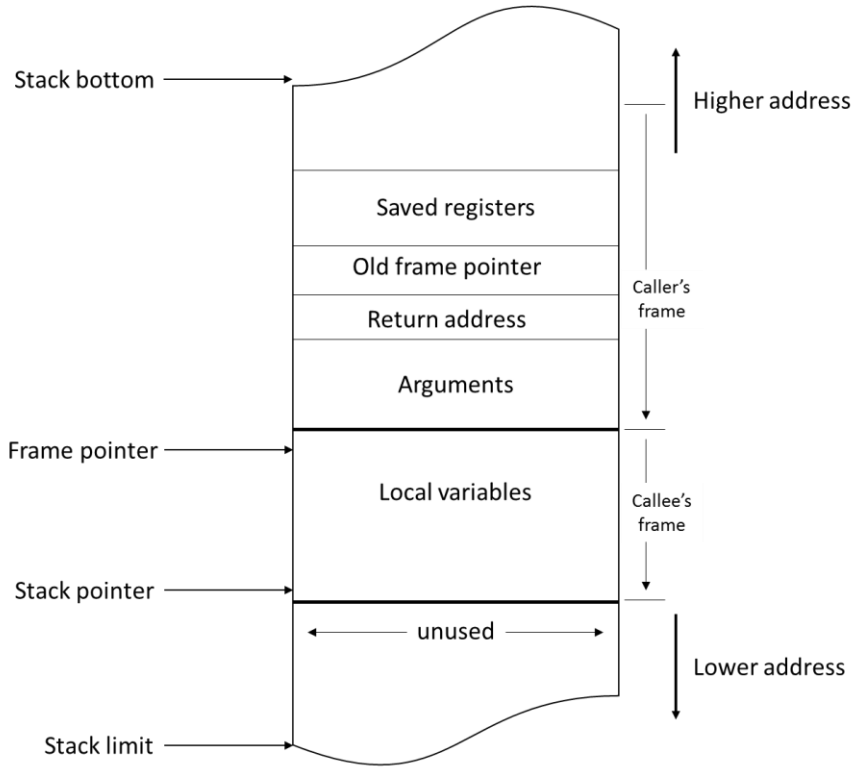
בתרגיל תידרשו לנהל את המחסנית עם רשומות ההפעלה של הפונקציות הנקראות. לשם כך יש להשתמש ברגיסטר $fp להצביע לראש הפריים הנוכחי וברגיסטר $sp כדי להצביע לראש המחסנית (הרגיטסר $sp יאותחל אוטומטית להצביע לראש המחסנית בתחילת התכנית).

התרשים המופיע מטה הוא בגדר המלצה, ניתן לסדר את הרכיבים ברשומת ההפעלה בכל סדר שנוח לכן כל עוד אתן מצליחות לשחזר אותם בצורה נכונה.

האחסון של $fp הישן ו-$ra (כתובת החזרה, return address) נועדו לאפשר למחסנית לקפוץ בחזרה אל הקוד וראש רשומת ההפעלה של הפונקציה הקודמת, גם אחרי שתיקרא פונקציה נוספת – כפי שנלמד בשיעור. האחסון של כל הרגיסטרים שהיו בשימוש בזמן הקריאה לפונקציה נועד לאפשר לחישוב להמשיך מאותה הנקודה ברגע שקריאת הפונקציה תסתיים.

את המשתנים והפרמטרים לפונקציות יש לאחסן במחסנית, לפי ה-offsets שחושבו בתרגיל 3. מומלץ לשמור כל משתנה, ללא תלות בטיפוסו, ב-4 בתים במחסנית (כגודל רגיסטר). ניתן להיעזר בדוגמאות לניהול המחסנית במדריך MIPS.

מומלץ להתייחס לכל הרגיסטרים כרגיסטרים שהאחריות לגבותם היא בידי הcaller. שימו לב כי הדבר אומר שיגובו רק הרגיסטרים שנמצאים בשימוש.



רשומת הפעלה: הצעת הגשה

4. סמנטיקה

יש לממש את ביצוע כל ה-statements בפונקציה ברצף בסדר בו הוגדרו. הסמנטיקה של ביטויים אריתמטיים ושל קריאות לפונקציות מוגדרת כמו הסמנטיקה שלהם בשפת C. ההרצה תתחיל בפונקציה main, ותסתיים כשהפונקציה main חוזרת. עבור מבני הבקרה יש להשתמש ב-backpatching. ניתן להיעזר בדוגמאות מהתרגולים.

א. אתחול משתנים

יש לאתחל את כל המשתנים בתכנית כך שיכילו ערך ברירת מחדל במידה ולא הוצב לתוכם ערך.

הטיפוסים המספריים יאותחלו ל0.
הטיפוס הבוליאני יאותחל לfalse.

ב. ביטויים חשבוניים

יש לממש פעולות חשבוניות לפי הסמנטיקה של שפת C.

הטיפוס המספרי int הינו signed, כלומר מחזיק מספרים חיוביים ושליליים.
הטיפוס המספרי byte הינו unsigned, כלומר מחזיק מספרים אי-שליליים בלבד.

חילוק יהיה חילוק שלמים.

השוואות רלציוניות בין שני טיפוסים מספריים שונים יתייחסו לערכים המספריים עצמם (כלומר, כאילו הערך הנמצא ב-byte מוחזק על ידי int). לכן, למשל, הביטוי

```
8b == 8
```

יחזיר אמת.

יש לממש שגיאת חלוקה באפס. במידה ועומדת להתבצע חלוקה באפס, תדפיס התכנית

```
"Error division by zero\n"
```

ותסיים את ריצתה.

ב.1. גלישה נומרית

יש לדאוג שתתבצע גלישה מסודרת של ערכים נומריים במידה ופעולה חשבונית חורגת (מלמעלה או מלמטה) מהערכים המותרים לטיפוס.

טווח הערכים המותר ל-int הוא 0-0xffffffff (כך שה-0-0x7fffffff חיוביים ו-0x80000000-0xffffffff שליליים). גלישה נומרית עבור int אמורה לעבוד באופן אוטומטי במידה ומימשתן את התרגיל לפי ההנחיות (כלומר, תתקבל תמיד תוצאה בטווח הערכים המותר, ללא שגיאה).

טווח הערכים המותר ל-byte הוא 0-255. יש לוודא כי גם תוצאת פעולה חשבונית מסוג byte תניב תמיד ערך בטווח הערכים המותר, על ידי truncation של התוצאה (איפוס הביטים הגבוהים בתוצאה).

ג. ביטויים בוליאניים

יש לממש עבור ביטויים בוליאניים short-circuit evaluation, באופן הזהה לשפת C: במידה וניתן לקבוע בשלב מסוים בביטוי בוליאני את תוצאתו, אין להמשיך לחשב חלקים נוספים שלו. כך למשל בהנתן הפונקציה printfoo:

```
bool printfoo() {
      printi(1);
      return true;
}
```

והביטוי הבוליאני:

```
true or printfoo()
```

לא יודפס דבר בעת שערוך הביטוי.

בנוסף, אין להשתמש ברגיסטרים לתוצאות או תוצאות ביניים של ביטויים בוליאניים. יש לתרגם אותם לסדרת קפיצות כפי שנלמד בתרגול. במידה והביטוי הבוליאני הוא ה-Exp במשפט השמה למשתנה או ברשימת פרמטרים לפונקציה, יש להשתמש רק ברגיסטר אחד לתוצאה הסופית כמשתנה ביניים לצורך ביצוע sw (שמירה לזיכרון).

ד. קריאה לפונקציה

בעת קריאה ל-Call, ישוערכו קודם כל הארגומנטים של הפונקציה לפי הסדר (משמאל לימין) ויועברו לפונקציה הנקראת דרך המחסנית. קוד הפונקציה יקרא על ידי קפיצה ובסוף ביצוע הפונקציה תבוצע קפיצה בחזרה לקוד הקורא. (יש להשתמש ב-jal כדי לקפוץ לקוד הפונקציה, מה שישמור את הכתובת אליה יש לחזור ב-$ra).

במידה והפונקציה מחזירה ערך, מומלץ להחזיר אותו ב-$v0 אך ניתן גם להחזירו במקום מוסכם מראש ברשומת ההפעלה, והקוד הקורא יוכל לקרוא אותו משם.

ה. משפט if

בראשית ביצוע משפט if משוערך התנאי הבוליאני Exp. במידה וערכו true, יבוצע ה-Statement בענף הראשון, ואחריו ה-Statement שנמצא בקוד אחרי ה-if. במידה וערכו false, יבוצע ה-Statement בענף השני, ואחריו ה-Statement שנמצא בקוד אחרי ה-if.

התנאי הבוליאני עשוי לכלול ביטויים מורכבים, לפי המוגדר בתרגיל 3.

ו. משפט while

בראשית ביצוע משפט while משוערך התנאי הבוליאני Exp. במידה וערכו true, יבוצע ה-Statement, והריצה תחזור לשערוך של Exp. במידה וערכו false, יבוצע ה-Statement שנמצא בקוד אחרי ה-while.

התנאי הבוליאני עשוי לכלול ביטויים מורכבים, לפי המוגדר בתרגיל 3.

ז. משפט break

ביצוע משפט break בגוף לולאה יגרום לכך שהמשפט הבא שיתבצע הוא המשפט הבא אחרי הלולאה הפנימית ביותר בתוכה ה-break מופיע.

ביצוע משפט break בגוף case של משפט switch יגרום לכך שהמשפט הבא שיתבצע הוא המשפט הבא אחרי משפט הswitch הפנימי ביותר בתוכו ה-break מופיע.

ח. משפט switch

בראשית ביצוע משפט switch משוערך הביטוי Exp. הוא מושווה לרשימת ה-case-ים במשפט באותה השוואה כמו אופרטור ==, וברגע שנמצא ערך שווה ה-statement המוגדר ב-case שלו יבוצע.

מכיוון שהדקדוק של השפה לא בדק כי כל ערך מופיע רק פעם אחת, יש לבצע את ההשוואה לפי הסדר. כלומר, אם הערך Exp שווה לערכים של שני case-ים, יתבצע ה-Statement של ה-case המופיע ראשון.

אם לא קיים אף case מתאים, יתבצע הdefault (אם קיים, אחרת לא יתבצע דבר).

אם לא מופיע משפט break בסוף הcase, המשפט הבא שיתבצע הוא המשפט הראשון של הcase שמופיע בקוד אחרי הcase הנוכחי.

אחרי המשפט האחרון המופיע בתוך ה-switch, בין אם מסתיים בbreak ובין אם לא, יתבצע המשפט הבא אחרי משפט ה-switch.

ט. משפט return

במידה וזהו משפט return Exp, ראשית ישוערך Exp וישמר ברגיסטר $v0 או במקום המתאים במחסנית.

הפקודה הבאה שתבוצע אחרי return היא הפקודה הבאה אחרי ה-call בפונקציה הקוראת. במידה והפונקציה הנוכחית היא main, קריאה ל-return תסיים את התכנית.

5. פונקציות פלט

קיימות שתי פונקציות פלט בשפת FanC. הראשונה מקבלת פרמטרים מספריים, והשנייה פרמטר מחרוזת. עליכן לכלול את המימוש שלהן בקוד האסמבלי שתייצרו. להלן מימושים מומלצים לשתי הפונקציות, הקוראים את הארגומנטים מהמחסנית. ניתן לשנות אותן כל עוד האפקט זהה.

הפונקציה printi:

```
lw $a0,0($sp)
li $v0,1
```

```
syscall
jr $ra
```

הפונקציה print:

```
lw $a0,0($sp)
li $v0,4
syscall
jr $ra
```

6. טיפול בשגיאות

תרגיל זה מתמקד בייצור קוד אסמבלי ולא מוסיף שגיאות קומפילציה מעבר לאלה שהופיעו בתרגיל 3.

יש לדאוג שהקוד המיוצר מטפל בשגיאת חלוקה באפס שהוזכרה בפרק הסמנטיקה.

7. קלט ופלט המנתח

קובץ ההרצה של המנתח יקבל את הקלט מ-stdin.

את תכנית האסמבלי השלמה יש להדפיס ל-stdout (באמצעות הפונקציות המתאימות במחלקה CodeBuffer). הפלט ייבדק על ידי הפניה לקובץ של stdout ו-stderr והרצה על ידי סימולטור Spim.

8. הדרכה

כדאי לממש את התרגיל בסדר הבא:

1. פונקציות להקצאת ושחרור רגיסטרים מתוך pool הרגיסטרים האפשרי.
2. חישובים לביטויים אריתמטיים. התחילו מחישובים פשוטים והתקדמו לחישובים מורכבים יותר. בדקו אותם בעזרת הסימולטור.
3. חישובים לביטויים בוליאניים מורכבים. בדקו אותם בעזרת הסימולטור.
4. שמירת וקריאת משתנים במחסנית.
5. רצף של statements.
6. מבני בקרה.
7. קריאה לפונקציות הפלט.
8. קריאה לפונקציות.

מומלץ ליצור mips program template אליו תוכלו להעתיק קטעי קוד אסמבלי קצרים שיצרתן בשלבי עבודה מוקדמים. כך תוכלו להריץ ולבדוק את הקוד שאתן מייצרות בטרם יצרתן תכנית מלאה.

מומלץ להיעזר במבני הנתונים של stl. מומלץ לכתוב מחלקות למימוש פונקציונליות נחוצה. כדאי מאוד להיעזר בתבנית העיצוב (design pattern) singleton.

9. הוראות הגשה

שימו לב כי קובץ ה-Makefile מאפשר שימוש ב-STL. אין לשנות את ה-Makefile.

יש להגיש קובץ אחד בשם ID1-ID2.zip, עם מספרי ת"ז של שתי המגישות. על הקובץ להכיל:

- קובץ flex בשם scanner.lex המכיל את כללי הניתוח הלקסיקלי
- קובץ בשם parser.ypp המכיל את המנתח

- את כל הקבצים הנדרשים לבניית המנתח, כולל *.output שסופקו כחלק מתרגיל 3 וקבצי *.bp שסופקו כחלק מתרגיל זה, אם השתמשתן בהם.

בנוסף יש להקפיד שהקובץ לא יכיל:

- את קובץ ההרצה
- קבצי הפלט של flex ו-bison
- את קובץ Makefile שסופק כחלק מהתרגיל

יש לוודא כי בביצוע unzip לא נוצרת תיקיה נפרדת. **על המנתח להיבנות על השרת csl2 ללא שגיאות באמצעות קובץ Makefile שסופק עם התרגיל.** הפקודות הבאות יגרמו ליצירת קובץ ההרצה hw5:

```
unzip id1-id2.zip
cp path-to/Makefile .
make
```

פלט המנתח צריך להיות ניתן להרצה על ידי הסימולטור. כך למשל, יש לוודא כי תכניות הדוגמה באתר מייצרות פלט זהה לפלט הנדרש. ניתן לבדוק את עצמכם כך:

```
./hw5 < path-to/t1.in >& t1.il
cd path-to-spim/
./spim -file path-to/t1.il > t1.res
diff path-to/t1.res path-to/t1.out
```

יריץ את המנתח, ייצר קובץ אסמבלי, יריץ את spim עליו ללא שגיאות, ו-diff יחזיר 0.

הגשות שלא יעמדו בדרישות לעיל יקבלו ציון 0 ללא אפשרות לבדיקה חוזרת.

בדקו היטב שההגשה שלכן עומדת בדרישות הבסיסיות הללו לפני ההגשה עצמה. מומלץ לכתוב גם טסטים נוספים שיבדקו את נכונות המימוש עבור מבני הבקרה השונים.

שימו לב כי באתר מופיע script לבדיקה עצמית לפני ההגשה בשם selfcheck. תוכלו להשתמש בו על מנת לוודא כי ההגשה שלכן תקינה.

בתרגיל זה (כמו בתרגילים קודמים בקורס) ייבדקו העתקות. אנא כתבו את הקוד שלכן בעצמכן.

בהצלחה!