תורת הקומפילציה
תרגיל 5

ההגשה בזוגות בלבד

שאלות במייל בלבד אל hilap@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 4. תחביר השפה הוא כפי שהכרתן ומימשתן בתרגיל בית 4. 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: addu, subu, mulo, divu, 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.

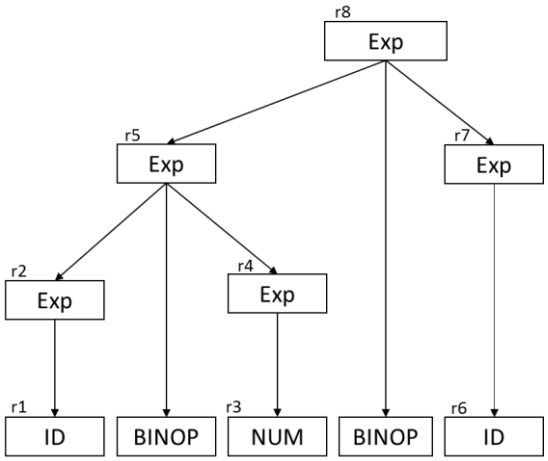
במידה ותרצו להשתמש בפקודות נוספות - מותר להשתמש בכל מה ש-Spim מסוגל להריץ (יש לבדוק את עצמכם עם הגרסה המצורפת לתרגיל, 9.1.18, ולא באף גרסה אחרת).

ב. רגיסטרים זמניים

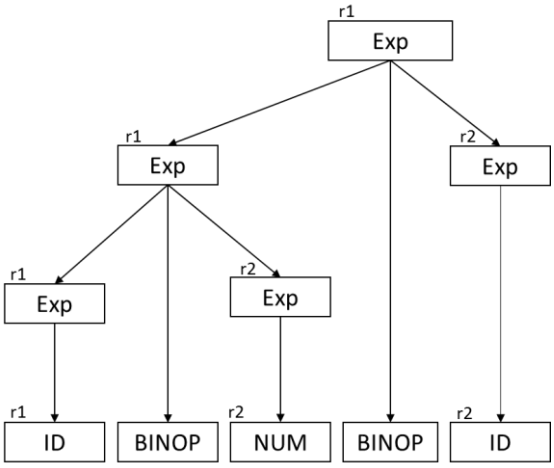
שימו לב כי ב-MIPS אין משתנים זמניים ולכן העבודה עם ביטויים מספריים בתרגיל תהיה ברגיסטרים. בתור רגיסטרים זמניים לפעולות ניתן להשתמש בכל הרגיסטרים שאינם שמורים

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 לחישוב ביטויים בוליאניים (דרישה זו תובהר בפרק הסמנטיקה).

ג. לייבלים

ב-MIPS יעדים של קפיצות מסומנים על ידי לייבלים אלפאנומריים 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 כל 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.

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.

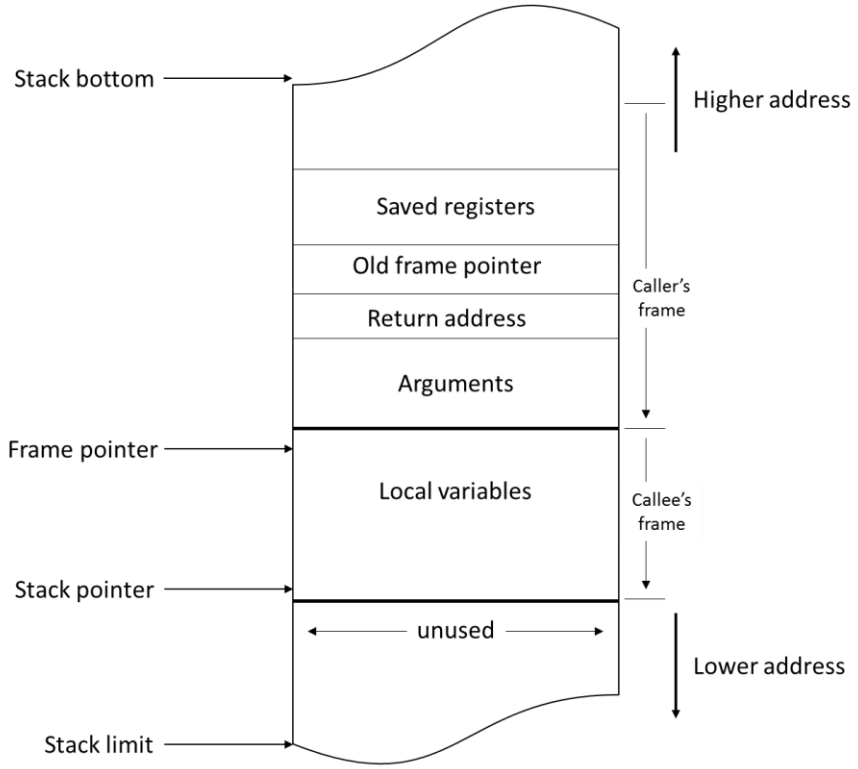
ד. דאטה

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).

לצורך העבודה עם באפר הקוד נתונה לכם מחלקה CodeBuffer בקובץ bp.hpp. במחלקה תוכלו למצוא מתודות לעבודה עם באפר קוד וביצוע backpatching, מתודה שתדפיס את באפר הקוד ל-stdout בתוספת הצהרה על אזור text, ובנוסף 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 בהם.

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.



רשומת הפעלה: הצעת הגשה

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. הסדר המופיע בתרשים הוא

בגדר המלצה, ניתן לסדר את הרכיבים בכל סדר שנוח לכן כל עוד אתן מצליחות לשחזר אותם בצורה נכונה.

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

4. סמנטיקה

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

א. ביטויים חשבוניים

יש לממש פעולות חשבוניות לפי הסמנטיקה של שפת C. שני הטיפוסים המספריים הינם unsigned, כלומר מחזיקים מספרים אי-שליליים בלבד. חילוק יהיה חילוק שלמים.

אין צורך לבצע overflow לפעולות חשבוניות על byte, זוהי דרישת בונוס.

השוואות רלציוניות בין שני טיפוסים 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. 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, יש להשתמש רק ברגיסטר אחד לתוצאה הסופית כמשתנה ביניים לצורך ביצוע sw (שמירה לזיכרון).

ג. 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, ניתן להחזיר אותו ב-$v0 או במקום מוסכם מראש ברשומת ההפעלה, והקוד הקורא יוכל לקרוא אותו משם.

ד. 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.

הכלל הבוליאני עשוי לכלול ביטויים מורכבים, לפי המוגדר בתרגיל 4.

ה. 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.

הכלל הבוליאני עשוי לכלול ביטויים מורכבים, לפי המוגדר בתרגיל 4.

ו. משפט break

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

ז. 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, לא יתבצע דבר.

ח. 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. דרישות בונוס

הדרישות המפורטות בפרק זה **אינן חובה** וניתן לקבל ציון מלא בתרגיל בלעדיהן. מימוש מלא של כל דרישה יזכה ב-8 נקודות בונוס לתרגיל (שהן נקודה לציון הסופי של הקורס) או 16 נקודות בונוס, כמצוין ליד הסעיף. ייתכן ניקוד חלקי. שימו לב כי יש לממש את דרישות הבונוס בלי לפגוע בדרישות האחרות של התרגיל.

א. גלישה נומרית (8 נק')

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. 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. עבור דרישת הבונוס, יש לוודא כי גם תוצאת פעולה חשבונית מסוג byte תניב תמיד ערך בטווח הערכים המותר, על ידי truncation של התוצאה, בדיוק כפי שהיא מתבצעת עבור int.

ב. שגיאה בשימוש במשתנה שלא אותחל (16 נק')

ניתן, על פי כל הבדיקות התחביריות והסמנטיות שהוגדרו לשפת FanC, להשתמש במשתנה שהוגדר אך עדיין לא הושם לתוכו ערך. כך למשל, לא תתקבל שום שגיאה עבור הקוד הבא:

```
int a;
int b = a;
```

ניתן לזהות זאת בקלות בזמן ריצה, אך נבקש אזהרה על כך בזמן קומפילציה. במידה ומזוהה מצב בו משתמשים במשתנה ללא אתחול, יש להדפיס הודעת שגיאה הכוללת את מספר השורה בה מופיע השימוש ולסיים את הקומפילציה:

```
"line %d: use of uninitialized variable\n"
```

על מנת שלא לפגוע בדרישות התרגיל האחרות, אין להודיע על שגיאה במקרה ומשתמשים במשתנה בתוך ענף של מבנה בקרה בו אותחל. כך למשל, הקוד הבא תקין:

```
int a;
if (x < 10) // x is defined and initialized
{
      a = 8;
      printi(a);
}
```

מכיוון שלפני השימוש ב-a, ישנו אתחול של a. לעומת זאת, אם אחרי ה-if יתווסף כעת עוד שימוש ב-a, עליו ניתן להחזיר שגיאה.

הדרכה: כדאי לשמור בטבלת הסמלים דגל האם המשתנה אותחל. במידה והמשתנה מאותחל ב-scope שונה משהוגדר, יש לשמור את דגל האתחול ל-scope הנוכחי ול-scope-ים פנימיים יותר.

ביציאה מ-scope, להעביר את רשימת האתחולים, ובמידה של מספר ענפים, למזג אותה עם רשימת האתחולים של הענפים האחרים. שימו לב כי יש דמיון מסוים למיזוג הרשימות של שיטת backpatching.

שימו לב כי **דרישה זו לא תיבדק אוטומטית**. יש לשלוח מייל בעת הגשת התרגיל עם מספרי ת"ז של שתי המגישות עם בקשה לבדוק אותה. **לא יתקבלו בקשות לבדיקה בדיעבד.**

7. טיפול בשגיאות

תרגיל זה מתמקד בייצור קוד אסמבלי, ולכן שגיאת הקומפילציה היחידה הנוספת לאלה שהופיעו בתרגיל 4 היא השגיאה בבונוס 2.

יש לדאוג שהקוד המיוצר מטפל במספר שגיאות שהוזכרו בפרקי הסמנטיקה והבונוס:

1. חלוקה באפס
2. שימוש במשתנה לא מאותחל (בונוס)

8. קלט ופלט המנתח

קובץ ההרצה של המנתח יקבל את הקלט מ-stdin.

את תכנית האסמבלי השלמה יש להדפיס ל-stdout. הפלט ייבדק על ידי הפניה לקובץ של stdout ו-stderr והרצה על ידי סימולטור Spim.

9. הדרכה

כדאי לממש את התרגיל בסדר הבא:

1. פונקציות להקצאת ושחרור רגיסטרים מתוך pool הרגיסטרים האפשרי.
2. חישובים לביטויים אריתמטיים. 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.
8. קריאה לפונקציות הפלט.

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

בנוסף מומלץ לא להתחיל במימוש דרישות הבונוס עד שסיימתן את כל הדרישות האחרות.

10. הוראות הגשה

שימו לב כי קובץ ה-Makefile מיועד לגרסת gcc 4.7.1, ומאפשר 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 מהתרגיל 4 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 t2 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 תכנית הדוגמה באתר מייצרת פלט זהה לפלט הנדרש. ניתן לבדוק את עצמכם כך:

```
./hw5 < path-to/t1.in 2>&1 > 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.

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

בהצלחה!