מס' גרסה:5

פיתוח תוכנה מבוסס java

S-Emulator קיץ 2025 -

מרצה: **אביעד כהן** [aviadco@mta.ac.il](mailto:aviadco@mta.ac.il)

בודק: **איתי כהן**  itaych@mta.ac.il

התרגיל מנוסח בלשון זכר, אך מכוון לכלל המגדרים והתחושות בצורה שווה

תוכן העניינים

[דרישות הקורס 3](#_Toc203931651)

[כללי 3](#_Toc203931652)

[איך להגיש תרגילים באיחור, ולהישאר בחיים 4](#_Toc203931653)

[הנחיות כלליות לכתיבת התרגיל 5](#_Toc203931654)

[S-Emulator 7](#_Toc203931655)

[מוטיבציה 7](#_Toc203931656)

[S-Emulator – תיאור המע' 8](#_Toc203931657)

[הנחיות כלליות למימוש התרגיל 11](#_Toc203931658)

[תרגיל 1 – מימוש **S-Emulator** כאפליקציית Console (20%) - הגשה: 30.8.25 12](#_Toc203931659)

[פרטים יבשים 12](#_Toc203931660)

[דרישות 12](#_Toc203931661)

[חלוקה למודולים 15](#_Toc203931662)

[איך מתחילים ? (המלצה...) 16](#_Toc203931663)

[בונוסים 16](#_Toc203931664)

[סבבה, סיימתי. מה ואיך להגיש ? 16](#_Toc203931665)

[תרגיל 2 – מימוש **S-Emulator** כאפליקציית JavaFX ( 40%) – הגשה: 1.10.25 17](#_Toc203931666)

[פרטים יבשים 17](#_Toc203931667)

[דרישות 17](#_Toc203931668)

[איך מתחילים ? 19](#_Toc203931669)

[בונוסים 20](#_Toc203931670)

[סבבה, סיימתי. מה ואיך להגיש ? 20](#_Toc203931671)

[תרגיל 3 – מימוש **S-Emulator** כאפליקציית Client - Server (35%) – הגשה: 29.10.25 21](#_Toc203931672)

[פרטים יבשים 21](#_Toc203931673)

[דרישות 21](#_Toc203931674)

[חלוקה למודולים 24](#_Toc203931675)

[איך מתחילים ? 25](#_Toc203931676)

[בונוסים 25](#_Toc203931677)

[סבבה, סיימתי. מה ואיך להגיש ? 26](#_Toc203931678)

[נספח א' – מנגנון ההרצה 27](#_Toc203931679)

[נספח ב' – מנגנון ההרחבה 28](#_Toc203931680)

[נספח ג' – תיאור הפקודות 29](#_Toc203931681)

[נספח ד' – מנגנון ההרכבה 37](#_Toc203931682)

[נספח ה' – תיאור מבנה המערכת באמצעות קובץ XML 39](#_Toc203931683)

[סכמה תרגיל 2 41](#_Toc203931684)

[נספח ו' – תיאור גרפי של הסכמה 42](#_Toc203931685)

דרישות הקורס

## כללי

1. בקורס אין בחינה אך חובה להגיש תרגילים (סה"כ 4).
2. המלצתי היא להגיש את התרגילים ביחידים. אולם אם אין ברירה - את מרבית התרגילים (3) ניתן להגיש בזוגות, אך לא בשלישיות / רביעיות / חמישיות או יותר

(כן, גם אם מדובר בשלישיה / רביעיה / חמישיה הצועדת יחדיו לאורך שנים מאז גיל הגן והגישה עד עכשיו את כל הפרוייקטים ביחד).

את תרגיל ה - reflection חובה להגיש ביחידים.

1. בעבודה משותפת על תרגיל יש להקפיד על מעורבות אקטיבית של כלל המגישים בכל חלקי התרגיל.
2. במידה והוגדר בונוס לתרגיל מסוים, ציון הבונוס יתווסף לציון התרגיל בלבד (ולא לציון הסופי של הקורס כולו).
3. התרגילים יוגשו דרך מערכת Mama. מוגדר רכיב 'מטלה' נפרד לכל תרגיל.
4. לפני שליחת התרגיל יש לבדוק שהוא עובד ומכיל את הקבצים המעודכנים ביותר, על מערכת "נקייה".

בצעו את סט הפעולות שאתם מצפים מן הבודק לבצע וודאו כי הכל מתנהל כראוי וכסדרו.

1. ניתן להחליף את השותפ/ה בכל תרגיל, ללא צורך באישור או הודעה למרצה.

הניקוד על כל תרגיל נזקף לזכות הסטודנטים שבצעו אותו בלבד.

1. יש להעלות את התרגיל רק עבור אחד מבני הזוג ולהוסיף את שם בת/בן הזוג ומספר תעודת הזהות שלה/ו בקובץ ה readme המצורף (פרטים בהמשך).

## איך להגיש תרגילים באיחור, ולהישאר בחיים

1. ניתן להגיש תרגיל עד חמישה ימים איחור, כאשר עבור כל 24 שעות איחור – תורד נקודה אחת מציון התרגיל ; תרגיל שיוגש באיחור של יותר משבוע (ללא סיבה מוצדקת) – **פשוט לא יבדק**.
2. במידה והגשתם תרגיל אולם הבודק נתקל בבדיקתו במצב שפשוט לא מאפשר את המשך הבדיקה (למשל כישלון בטעינת קובץ הבדיקה) – הרי שאתם מוגדרים כתקלת level 0. במקרה של תקלה שכזו הבודק יידע אתכם ויאפשר לכם לבדוק, לתקן ולהגיש מחדש את התרגיל כדי שאפשר יהיה לבודקו אחרי הכל.   
   שימו לב **כי בכל במקרה של הגשה חוזרת** בגלל level 0 – הציון לתרגיל יתחיל מ 90, ללא שום קשר לאופי הבעיה ו/או התיקון (גם אם התיקון היה "קטן". גם אם התיקון היה בגלל בלבול בהגשה של גרסה קדומה יותר של הקבצים. גם אם הכלב אכל לכם את שיעורי הבית)
3. בתרגילים השונים ניתן לממש בונוסים (פרטים בהמשך).

המטרה של הבונוס היא לעזור לכם להעלות את הציון ולא להורידו !

רוצה לאמר: אל תגישו באיחור רק בשביל להספיק לפתח בונוס.

בונוס מפתחים **אם ורק אם** סיימתם את כל דרישות הבסיס להגשה, יש לכם עוד מספר ימים, וברצונכם לנסות ולהגדיל את הציון ע"י בונוס.

היות וכך, ולמען הסר כל ספק: **לא ייבדקו** הבונוסים עבור תרגילים שהוגשו באיחור (שאינו מוצדק).

בהתאם לכך, אני שומר לעצמי חירות רבה יותר בשינוי כזה או אחר של מי מסעיפי הבונוס, גם במהלך התרגיל עצמו.

1. עומס בלימודים, בעבודה, בחיים, בגלל הילדים או ההורים, שכנים וחברים (או בכל תחום אחר) אינו נחשב כסיבה לגיטימית לבקשת הארכה.
2. במקרה של בקשה להארכה (מכל סיבה שהיא, לרבות מילואים ומחלה) יש לפנות למרצה מראש על מנת לקבל אישור.  
   הפנייה תתבצע במייל.
3. לאנשי הקבע – הישארות של שבת בבסיס אינה נחשבת כסיבה לגיטימית להארכה (מכיוון שזה חלק מהסדר העבודה בצה"ל); יציאה לאבט"ש כן נחשבת כמילואים ויש להגיש אישור ממפקד הבסיס.
4. ניתן לערער על ציון של תרגיל לכל היותר שבוע מיום פרסום המשוב והציון במע' המאמא.

כדי לערער יש לשלוח מייל לבודק בצירוף כל הסיבות והטענות שלכם.

## הנחיות כלליות לכתיבת התרגיל

* במהלך הקורס יוצגו דוגמאות והסברים מבוססים על כתיבה בסביבת הפיתוח (IDE) – Intellij IDEA.

אתם מוזמנים (ומעודדים בזאת) לפתח גם כן את התרגיל בסביבת העבודה intellij.

ניתן לקבל רישיון חינם לשימוש בגרסת ultimate, רק בשל היותכם סטודנטים במכללה (כבר שווה !)

יחד עם זאת, כל אחד רשאי לבחור לעבוד בסביבת העבודה הנוחה והמוכרת לו. כך או אחרת הגשת התרגיל אינה כוללת את סביבת הפיתוח אלא אך ורק הרצה ידנית מ cmd (כמו פעם...).

שימו לב: מבחינתכם, לבודק פשוט אין intellij (או כל ide אחר) ולכן זו אפילו לא אופציה.

חיסכו ממני (ומכם) את כתיבת המייל המבקש זאת.

* יש להגיש את התרגילים בתור קובץ zip/rar (לא 7Z !)

הקובץ יכיל:

1. כל הקבצים הרלבנטים להפעלת התרגיל (jar/war – פרטים בגוף התרגיל).
2. קובץ אצווה ( == batch) שיכיל את הפקודה שמריצה את התרגיל.
3. קובץ readme שיכיל את פרטי המגיש/ים, כמו גם הנחיות כלליות להרצה התרגיל וכל הנחות שלקחתם במהלך התרגיל ואתם סבורים שחשוב כי הבודק יכיר. דמיינו כי בכל שאלה/תקלה שיתקל בהן הבודק, יעמוד לרשותו רק קובץ ה readme שלכם. דאגו להבהיר ולהסביר את כל הדברים שיכולים להשתבש ו/או שבעטיים ייתכנו בעיות/שאלות/תהיות וכיוצב'.

כמו כן, כל הנחה שאתם מניחים בעצמכם לגבי אופן מימוש התרגיל (בין אם בלוגיקת התרגיל ובין אם בהנחה טכנולוגית) צריכה להיות רשומה בקובץ.

על קובץ הreadme להיות בפורמט word או pdf (**לא notepad !**).   
חי נפשי – אם מישהו מגיש readme כקובץ טקסט פשוט -ירד לו ניקוד מהתרגיל...

* דווקא בגלל שאין זהו קורס שבו יכנסו לנבכי הקוד ויבדקו כל שורה ושורה, יש להקפיד ביתר שאת על קוד נקי, מסודר, קריא ויעיל. בפרט:
* הימנעו משכפול קוד
* פונקציות ארוכות מדי (בדר"כ יותר מגודל עמוד)
* בחירת שמות גרועים למחלקות, לפונקציות ולמשתנים
* הזחה (אינדנטציה) נכונה
* imports מיותרים
* יש להקפיד להשתמש ב-modifiers בצורה נבונה:
* מחלקה שלא אמורים לבנות אובייקטים שלה אמורה להיות מוגדרת כ-abstract
* קבועים יש לסמן כ- final
* משתנים של המחלקה רצוי להגדיר כ-private אלא אם יש סיבה לגיטימית לבחירה אחרת.
* יש להקפיד על מוסכמות בסגנון הכתיבה – שמות מחלקות יתחילו באות גדולה, שמות חבילות, משתנים ופונקציות באות קטנה, שמות קבועים יהיו מורכבים רק מאותיות גדולות וכו'. ראו מסמך java coding conventions שהועלה למאמא.
* התמודדות עם קלט שאינו תקין (במקומות הרלבנטים) היא חלק בלתי נפרד מחווית המפתח (לטוב ולרע...).

יש לוודא קלט תקין מהמשתמש בכל שלב ולהחזיר הודאות שגיאה קריאות, אינפורמטיביות במידה והקלט אינו תקין. (למשל: לא להגיד שהקובץ לא תקין – אלא מה לא תקין בקובץ בצורה מפורטת...)

* כל הקלט והפלט בתרגילים השונים יהיה באנגלית בלבד.

אין להציג או לתמוך בקבלת קלט ו/או הצגת פלט בעברית או בכל שפה אחרת.

כל הקלטים באנגלית יהיו case insensitive, כלומר אין חשיבות ל capital case. דוגמא: MoMo=mOmO

* הוראות שגויות שייגרמו לאפליקציה שלא לרוץ יורידו נקודות, ולכן רצוי מאוד שתנסו להתקין את האפליקציה בעצמכם לפי ההוראות שתכתבו.
* **זהו תרגיל מתגלגל. המטרה היא לבנות בסיס ראשוני בתרגיל הראשון, ולהמשיך ולהשתמש בו, ככל האפשר (ואפשר !) במהלך התרגילים הבאים. השקיעו חשיבה ותכנון בעיצוב הפתרון תוך מחשבה על איך מה שתעשו היום ישרת אתכם מחר. (זה כלל נכון לחיים, לא רק לתרגיל זה).**
* חלק מהעבודה בתרגילים היא קבלת החלטות בנושאים שאינם מפורטים במדויק. מטרת התרגיל היא לתרגל את הנושאים המרכזיים הנלמדים בקורס. על כן, בכל מקום שלא מופיעה דרישה מדויקת – מוטל עליכם לבחור בדרך ההגיונית ביותר שנראית לכם ולציין את בחירתכם בקובץ ה Readme אשר מוגש עם התרגיל. אם יש ספק לגבי אופן פעולתכם אתם מעודדים לשאול האם הפתרון שאתם חושבים לתת לסוגיה מסוימת הוא קביל ולגיטימי (שאלות בפורום, מייל למרצה וכו)
* התרגיל מתקיים כולו במסגרת ג'אווה גרסה 21. הקפידו להוריד, לעבוד, לקמפל ולהריץ עם הגרסה המתאימה בלבד.
* **ווידוא הגשת התרגיל טרם הגשתו:**
* **יש לוודא כי ההגשה שלכם רצה היטיב על מע' נקייה, באופן שבו גם הבודק יריץ אותה,** על מערכת נקייה וללא תוצרי לוואי אחרים של הפעלות קודמות שלכם.
  + **הבודק יבצע את הבדיקה על מע' windows 10. כל מי שמפתח על גבי mac/linux – זכותכם – אבל גם חובתכם לוודא כי אתם רצים היטיב על windows 10. למען הסר ספק, לא תתבצע בדיקה על מע' הפעלה אחרת. כמו כן לא תהיה התחשבות בתקלות שמקורם רק בשל עבודה על מע' הפעלה שונות (ולא שאמורות להיות תקלות כאלה..)**
  + **יש לוודא כי כל קבצי הבדיקה השונים שהועלו ל mama נטענים בהצלחה ע"י המע' שלכם טרם ההגשה. בדיקת הבודק תתחיל מבדיקה בסיסית המבוססת בצורה גסה על קבצים אלה. חבל ליפול Level 0 על שטות שיכולתם לעלות עליה בשנייה עוד בשלב הפיתוח.**
  + **כאמור, הגשה חוזרת בשל תקלות level 0 תתחיל מראש מציון של 90. בלי שום יוצא מן הכלל. הקדימו תרופה למכה.**
* **בחלק מהתרגילים ניתנת אפשרות למימוש דרישות בונוס.**
  + **ישנם 2 סוגי בונוסים:**
    1. **בונוס בתוך טווח התרגיל - יכול להביא אתכם לכל היותר לציון 100, ולחפות במקרה והורדו לכם נקודות בשל תקלות.**
    2. **בונוס מחוץ לטווח התרגיל – יכול להעלות את ציונכם אף מעבר ל 100 (וכן, יש כפל מבצעים לטובת הלקוח).**
  + **בכל מקרה יש לבצע את הבונוס אם ורק אם סיימתם את כל דרישות הבסיס ההכרחיות לתרגיל.**
  + **חלק מהבונוסים בתרגילים השונים הם כאלה שנועדו "להקדים תרופה למכה" – מימוש דרישה בתרגיל n אשר בכל מקרה תגיע כדרישה חובה בתרגיל n+1.**

**הדבר נועד לעודד אתכם להוריד את העומס הצפוי בתרגיל n+1, מתוך הנחת יסוד שתרגיל n הוא קל יותר ומרווח יותר.**

**תכננו את עבודתכם בהתאם ושאפו "להקדים תרופה למכה", במידת האפשר. (וגם זה כלל חשוב לחיים, בלי קשר לתרגיל ולקורס).**

* + **פירוט הבונוסים, משקלם ונקודותיהם מפורט בגוף התרגיל הספציפי.**
  + **אם כבר מממשים בונוס, יש לממש את כולו, עפ"י דרישתו כדי לזכות במלוא הניקוד שהוא מקנה. בכל מקרה ההחלטה על ניקוד הבונוס היא בידי הבודק/מרצה בלבד (המגמה היא להיות נדיבים ככל האפשר...)**
  + **כאמור, ולמען הסר כל ספק – ניקוד הבונוס מתווסף לניקוד התרגיל הספציפי שבו הוא מומש ולא לניקוד הסופי של הקורס. לא ניתן לקבל ציון סופי בקורס שהוא מעל ל 100 (גם אם בזכות הבונוסים הגעתם לציון כזה(.**

S-Emulator

## מוטיבציה

כולנו מתכנתים, ברמה כזו או אחרת.  
אנחנו מכירים שפות תיכנות שונות ומגוונות, בהן טובות ובהן גם JavaScript... (הייתי חייב)

כל שפת תכנות באשר היא בסוף מורצת ע"י המעבד, כלומר מומרת בסדרה של תהליכים שונים ומגוונים החל מקוד המקור – הקריא והברור למתכנת – אל פקודות בסיסיות אשר המעבד יודע להריץ.  
הדבר המעניין הוא שעם סט יחסית מצומצם של פקודות בסיס של מעבד (עשרות פקודות במעבדים של ימינו) ניתן למעשה לייצר את כל העולם הדיגיטלי שאנו חווים בחיי היום יום: e-commerce, LLMs, traffic control, space missions, social media וכו'.

הנה לכם שאלה מעניינת:  
האם ניתן להסתדר ולעבוד עם מעבד שיש בו פחות פקודות ?   
מהו המספר המינימלי של פקודות המאפשרות את כל העושר הדיגיטלי שאנו חווים ?

לשם כך צריך לפנות לאזור התאורטי של מדעי המחשב.  
במדעי המחשב קיימים מספר דוגמאות של מודלים חישוביים. מודל חישובי הוא מכניזם תאורטי, מתמטי, המאפשר להציג באמצעותו את כלל הדברים הניתנים לחישוב, תוך שמירה על פשטות (יחסית) וקלות הבנה של יחידותיו הבסיסיות.  
(שימו לב עסקינן בפישוט - לא ביעילות; אין שום טענה או ציפיה שהמודל החישובי התאורטי יהיה "יעיל" יותר מהמעבד הפרקטי)  
  
המודל עוסק בחישוב של פונקציות מתמטיות, מספריות – אולם באמצעות קפיצה מחשבתית קלה אפשר לראות איך כל העולם הדיגיטלי שלנו ניתן להתייחסות ולהמרה כאוסף של פונקציות מתמטיות.

אחד מהמודלים האלה הוא מודל הנקרא **שפת S**.  
המודל מציע 3 פקודות (בשביל המדקדקים – יש גם פקודה רביעית אבל היא די משעממת) – ובאמצעות 3 הפקודות האלה בלבד – ניתן לייצר ולהביע את כל העולם הדיגיטלי שאנו חיים בו !

בתרגיל זה נכיר את מודל שפת ה S.  
נממש את יחידותיו הבסיסיות, ומעליהן נבנה עוד ועוד מכשירים וכלים שיאפשרו לנו "להעמיק" את יכולות השפה (synthetic sugars).  
נבנה סביבת עבודה שבמסגרתה ניתן יהיה להריץ ולדבג (!) ביצוע של תוכנית ולבסוף נבנה מודל מבוזר של שירות ענן המאפשר לכמה משתמשים להריץ את תוכניותיהם (כלומר תוכניות ה S שלהם) בתצורה של serverless על ארכיטקטורות מעבד מגוונות.

המע' תבנה בשלושה חלקים (לאורך כל הקורס):  
**חלק א'** – מימוש S-Emulator כמע' בסיסית ביותר. תפעול המע' יבוצע באמצעות ממשק console  
**חלק ב'** – הרחבת והעשרת S-Emulator עם עוד יכולות מתקדמות יותר; יצירת ממשק גרפי מבוסס Java FX  
**חלק ג'** – הפיכת S-Emulator לשרת המאפשר גישה מרחוק למספר משתמשים ולתפעול ולריצה של תוכניות מרחוק.

הערה:  
אם עשיתם כבר את קורס חישוביות, או אם אתם עושים אותו במקביל לקורס זה – אתם עתידים להיתקל בשפת S.  
חוקי השפה שונו במעט כדי ליצור אחידות בהגדרתה, אולם השינויים הם לא כאלה הגורעים (או המעשירים) את השפה מעבר להגדרתה הבסיסית (למשל שימוש בתוויות מסוג אחד בלבד וכו')

כמו כן, אם קורס חישוביות מפחיד אתכם, אם נכשלתם במבחן, אם המילה 'הוכחה' מדירה שינה מעינכם – אל דאגה ! כאן אינכם מתעסקים בפן התאורטי של הקורס, כי אם אך ורק בפן הפרקטי – כתיבת מנגנון שמאפשר לסמלץ את שפת S.  
  
בשום שלב לא תידרשו לכתוב הוכחות או להבין אותן; לא תדרשו להשתמש במשפטים או אקסיומות; למעשה אפילו לא תידרשו לכתוב תוכנית בעצמכם – אלא רק לכתוב מנגנון המאפשר את הגדרת השפה וריצתה.

**בקיצור יהיה כיף !**

## S-Emulator – תיאור המע'

### מבנה השפה

שפת S מאפשרת להגדיר במסגרתה תוכניות. תוכנית מכילה (לכל היותר) 4 סוגים של הוראות 'בסיסיות'.  
לכל הוראה (instruction) נגדיר את הסוג שלה, ואת אופן פעולתה.  
הקלט (והפלט) בשפה הוא מספרים טבעיים בלבד (0, 1, 2, 3…).  
  
ההוראות עובדות על/עם משתנים.   
כמות המשתנים אינה מוגבלת, ובכל רגע נתון אפשר לבחור להשתמש באיזה משתנה שרוצים (אין אקט של הגדרה/אתחול וכו').  
  
לכל הוראה בשפת S יכולה להיות מוצמדת תווית מסויימת (Label).   
במסגרת התוכנית ניתן יהיה להתייחס לתווית זו כסימון ההוראה אליה רוצים לקפוץ.  
  
תוכנית בשפת S מחזירה פלט אחד ויחיד (אחרי הכל מתארת פונקציה מתמטית...)   
הפלט ישב במשתנה מיוחד, קבוע, אשר ייקרא y. (האות y קטנה)

### משתנים

המשתנים בשפת S ייוצגו באמצעות אותיות. המשתנים, כאמור, מייצגים מספרים טבעיים בלבד (0, 1, 2, 3…)  
  
המשתנים בשפה מחולקים לשלוש משפחות:

* **משתני קלט**:  
  משתנים אלה ייצגו את הקלט שנכנס לתוכנית בעת הרצתה.   
  הם יסומנו באות x (קטנה) ולאחריה מספר סדרתי רץ המתחיל מ 1: x1, x2, … , xn (על קלט בגודל n)כלומר בראשית ריצת התוכנית, משתנה הקלט הראשון ייכנס אל x1 הבא בתור ייכנס אל x2 וכו'  
  התוכנית יכולה לפנות גם למשתנה קלט שלא ניתן לה בתחילתה (נגיד x758...). ערכם ההתחלתי של כל משתני הקלט שלא אותחלו הוא 0
* **משתני עבודה**:  
  משתנים אלה יהוו "משתני עבודה" שבהם התוכנית עושה שימוש כדי להגשים את ייעודה.   
  הם יסומנו באות z (קטנה) ולאחריה מספר סדרתי רץ המתחיל מ 1: z1, z2, … ∞ משתני העבודה זמינים במסגרת תפעול התוכנית ואפשר להשתמש במי מהם, לאו דווקא על פי מספרם הסידורי.  
  בראשית ריצת התוכנית כל משתני העבודה מאותחלים ל 0.
* **משתנה התוצאה**:  
  ההגדרה בשפת S היא כי הפלט של התוכנית יהיה בדיוק תוצאה מספרית אחת והיא תשב במשתנה מיוחד שייקרא y (קטנה).

### תוויות

תוויות הן מחרוזות המוצמדות להוראות.  
המטרה של התווית היא לסמן הוראה מסויימת כך שאפשר יהיה במידת הצורך לבצע קפיצה במסגרת ריצת התוכנית להוראה זו.  
לכל הוראה יכולה להיות מוצמדת לכל היותר תווית אחת בלבד. התווית תופיע בצידה השמאלי של ההוראה.  
  
התוויות בשפת S יסומנו באות L (גדולה) ולאחריה מספר רץ המתחיל מ 1: L1, L2, … ∞

במסגרת שפת S נגדיר תווית מיוחדת שתקרא **EXIT** (capital case) אשר תייצג למעשה עצירה אפקטיבית של ריצת התוכנית. (אפשר לחשוב עליה כעל תווית המייצגת את סוף התוכנית)

### הוראות

שפת S מכילה 4 פקודות בסיסיות בלבד:

1. **V ←V + 1**  
   הוראה זה פועלת על משתנה כלשהוא (מיוצג כאן ע"י הסימון V), ומגדילה את ערכו, יהיה אשר יהיה, ב 1
2. **V ←V - 1**  
   הוראה זה פועלת על משתנה כלשהוא (מיוצג כאן ע"י הסימון V), ומקטינה את ערכו, יהיה אשר יהיה, ב 1  
   באם הערך שלו הוא כבר 0 – אזי הוא יישאר 0 (אין ייצוג למספרים שליליים בשפת S)
3. **IF V ≠ 0 GOTO L**  
   הוראה זו פועלת על משתנה כלשהוא (מיוצג כאן ע"י הסימון V), ובודקת את ערכו.   
   אם ערכו שונה מ 0 – היא קופצת להוראה המסומנת בתווית L.  
   באם אין תווית שכזו בפקודות התוכנית, אזי התוכנית נחשבת אינה חוקית;  
   אם יש יותר מתווית אחת שכזו במסגרת התוכנית – הקפיצה תתבצע אל ההוראה הראשונה המכילה תווית זו.
4. **V ←V**  
   הוראה זו היא הוראת סרק שלמעשה לא מבצעת כלום (NO-OP).  
   למעשה היא משימה את ערכו של משתנה (המיוצג ע"י הסימון V) בעצמו.

### הרצה של תוכנית

תוכנית בשפת S היא למעשה אוסף של הוראות המסודרות בסדר מסויים אחת אחרי השנייה.  
  
**אתחול ריצה**:

* כל משתני הקלט יאותחלו במידע של הקלטים שהתקבלו: (x1..xn)  
  שימו לב: אפשר להפעיל את התוכנית על איזה מספר קלטים שרוצים, גם אם התוכנית לא עושה שימוש בקלטים אלה.
* כל משתני העבודה (z1, z2, …) יאותחלו ל 0
* משתנה התוצאה (y) יאותחל ל 0

**ביצוע ריצה**:

ריצת התוכנית מתחילה מהפקודה הראשונה בסדרת ההוראות, אך אינה חייבת להסתיים בפקודה האחרונה בתוכנית (למשל אם יש קפיצה באמצע ל **EXIT**).  
כל הוראה מבוצעת ועשויה לשנות את ערכם של מי מהמשתנים השונים.  
  
אחרי כל הוראה עוברים להוראה הבאה בתור אחריה.  
יוצאת דופן לכלל זה היא הוראה מס' 3:  
בהוראה זו אם מתקיים התנאי V ≠ 0 – ההוראה הבאה תהיה זו המוצמדת לה התווית L.   
רק אם התנאי אינו מתקיים – נמשיך להוראה הבאה בתור.

**סיום ריצה**:

כאשר מסיימים את הפקודה האחרונה בתוכנית, או כאשר נדרשים לקפוץ אל התווית **EXIT** – אז התוכנית נעצרת.  
תוצאת התוכנית הפורמלית היא ערכו של המשתנה y - יהיה אשר יהיה.  
(אנחנו נאפשר כמובן הצצה לערכם של כלל המשתנים שנעשה בהם שימוש במסגרת ריצת התוכנית..)

P:

[**L1**] x1← x1 – 1

y ← y + 1

IF x1 ≠ 0 GOTO **L1**

דוגמא:

מה תהיה תוצאת הריצה: P(4) ?  
  
ומה תהיה תוצאת הריצה: P(3949,9,29,485,294,5039) ?

### ממתיקים מלאכותיים

כתיבה בשפת S הגולמית, כשרק 4 פקודות בסיסיות נתונות לידינו, יכולה להיות משימה קשה וסיזיפית.  
על כן, במסגרת מודל השפה, נגדיר מספר "ממתיקים מלאכותיים" (Synthetic Sugars) אשר יאפשרו לנו להגדיר פקודות נוספות "ברוח השפה".   
כל ממתיק מלאכותי למעשה ניתן להרחבה (expansion) ולתיאור ע"י הפקודות הבסיסיות של שפת S.   
במסגרת ממתיק מלאכותי מסויים ניתן לעשות שימוש בממתקים מלאכותים אחרים.  
כך למעשה ניצור את היכולת לכתוב תוכניות מורכבות יותר עם פקודות "מתקדמות", אשר יכולות להתפרק/להתרחב/להיתרגם, בסופו של דבר, לאוסף הפקודות הבסיסיות של השפה (קומפיילר לשפת מכונה ? מישהו ?)  
  
במסגרת הממתיקים המלאכותיים נגדיר גם מטה-מנגנונים:  
אלו הם מנגנוני קוד שלמעשה מייצרים קוד אחר בדמותם (מטה-תכנות).  
(נשמע מוזר כרגע, אבל תכלס דברים שכולנו ראינו ושמענו עליהם כמו הפעלה של פונקציות, הרכבה של פונקציות וכו')

עיקבו אחר תיאור הממתיקים המלאכותיים ואופן פעולתם [בנספח ג'](#synthetic).

### פונקציות

תוכנית בשפת S מתארת פונקציה מתמטית. במסגרת הגדרה של פונקציה מתמטית ניתן להשתמש בפונקציה אחרת.  
בהמרה לשפת S, בהינתן תוכנית מסויימת, נוכל להשתמש בתוכנית זו במסגרת תוכנית אחרת.  
נתייחס לתוכנית בה נשתמש בתוך תוכנית אחרת כאל פונקציה, ונגדיר מנגנון הפעלה (ממתיק מלאכותי) שיאפשר לנו הרצה של פונקציה זו.  
הפונקציות הן למעשה תוכניות בשפת S. לא תצטרכו לכתוב אותן, הן יינתנו לכם במסגרת הקלט למע'. רק תצטרכו לכתוב מנגנון שמאפשר להשתמש בהם.

### יכולות נוספות

S-Emulator תתמוך במגוון יכולות נוספות (שיפותחו במסגרת כלל התרגילים):

* ביצוע הרחבה של תוכנית עד לפקודות היסוד (על פי דרגות)
* כתיבת DEBUGGER שיאפשר הרצה של התוכנית שלב אחר שלב
* ניהול ריצה serverless באמצעות קרדיטים
* ועוד..

## 

## הנחיות כלליות למימוש התרגיל

1. **המטרה היא לבנות מנוע מערכת גנרי, כזה שידע לקבל את הפרטים לגבי אופי המע' מתוך קובץ נתונים בפורמט XML (עבודה עם XML'ים תילמד במהלך הקורס כמובן).**

**מנוע המערכת הגנרי ילך וישתכלל מתרגיל לתרגיל, בהתאם לפיצ'רים השונים. כך תוכלו לחוות מהלך שלם של מוצר החל מרעיון קטן במימוש בסיסי וכלה במנוע מע' המניע אפליקציית ווב שלמה.**

1. **כחלק מהמע' תצטרכו לחשוב ולבחור לבד את מבני הנתונים השונים שישרתו את הצרכים של דרישות המע'.   
   זהו לא קורס במבני נתונים או באלגוריתמים, ומבני הנתונים/אלגוריתמים שתבחרו לממש לא חייבים להיות היעילים ביותר או האופטימליים. מספיק שהם יעבדו בצורה נכונה (ללא טעויות) ובזמן סביר.**
2. **ממשקי המשתמש השונים יפעלו מול מנוע המע' אשר יפותח מתרגיל לתרגיל בהתאם לדרישות.**
3. **כל הממשקים הפונים אל המשתמש החיצוני ומציגים לו מידע ו/או מבקשים ממנו מידע מספרי, חייבים להיות מבוססי ספירה המתחילה מ 1. גם אם פנימית אתם מממשים את מי מהרכיבים כמערך או רשימה (אז בסיס הספירה מתחיל מ 0) – עליכם להקפיד ולודא כי כלפי חוץ "תדברו" אך ורק במונחים של בסיס 1.**
4. **המע' כולה תתואר בשפה האנגלית בלבד, עם ממשק משתמש המתנהל משמאל לימין (במקומות הרלוונטים)**
5. **המע' כולה תיכתב ותורץ בסביבת העבודה של ג'אווה 21**

תרגיל 1 – מימוש **S-Emulator** כאפליקציית Console (20%) - הגשה: 30.8.25

## פרטים יבשים

צפי תחילת עבודה: **6.8.25** תאריך הגשה: **30.8.25**

צפי זמן לביצוע: **4 שבועות** ציון אפשרי מקסימלי: 105

משקל התרגיל: **20%** קושי: **סביר**

**מטרת התרגיל העיקרית**

1. הקמת מנוע המע' הבסיסי
2. יצירת ממשק console לתפעול המערכת

## דרישות

1. בתרגיל זה תקימו את תשתית העבודה הראשונית והבסיסית של S-Emulator.   
   את התשתית "תתפעלו" באמצעות ממשק console פשוט המציג תפריט פקודות שדרכו מפעילים את המע'.  
   בתרגיל תממשו מספר סוגים של ממתיקים מלאכותיים (כפי המוגדר [בנספח ג'](#synthetic)), תגדירו את היישויות העקריות במע' (פקודות, תוכנית, הרצה) ותאפשרו תפעול של תוכנית מלאה.  
   כל אלה יישבו בתוך "מנוע" המערכת, אשר ידע להגיב לכל פנייה המגיעה משכבת ממשק המשתמש, לעבד את הקלט ולהחזיר פלט רלבנטי.
2. יש לוודא תקינות קלט כחלק מכל אינטרקציה עם המשתמש, ובכל מקום שבו זה רלוונטי:

אם אתם מצפים לקבל מספר – לא לקרוס כי הכניסו לכם בטעות (או בכוונה) טקסט וכו'.

בכל מקרה של תקלה יש להיות מאוד ברורים במסר שמעבירים חזרה למשתמש: מה קרה ? מה הייתה מהות התקלה ? היכן שזה רלבנטי, איך לתקנה וכו'.

חישבו איך להיות ידידותיים למשתמש ולעולם אל תניחו כי מי שמשתמש באפליקציה שלכם הוא מתכנת בעצמו או מישהו שמגיע מהתחום ו"מכיר" איך דברים עובדים לבד.

1. **אין צורך** להשתמש בצבעים שונים במהלך תרגיל זה בעת ההדפסה ל console.

יתרה מזאת, ישנו צפי רב (ניסיון מהסמסטרים הקודמים) כי ניסיון לעשות כן תוך שימוש בספריות צד שלישי קורס אצל הבודק,

מעוות את כל תצוגת המסך וגורם לחוסר יכולת לבדוק את ההגשה.

גם אם בדקתם את זה אצלכם וזה עבד.

גם אם בדקתם במחשב של השכנה וזה עבד.

כמו כן **אין** לנקות את המסך בין פקודה לפקודה.

1. עליכם לכתוב ממשק משתמש בתצורת console.

ממשק המשתמש יכיל סט סופי של פקודות שדרכן ניתן יהיה להפעיל את המערכת.

אחרי הצגת תפריט הפקודות יש לחכות לקלט מהמשתמש באשר לפעולה אותה הוא רוצה לבצע. לאחר ביצוע הפעולה (שאולי תגרור בקשת קלט נוסף מהמשתמש) יש להציג את הפלט החוזר ממנה (לכל פקודה יש פלט החוזר ממנה) ואז להציג שוב את התפריט וחוזר חלילה.

**שימו לב**:

* ישנן פקודות שאין הגיון לבצע אותן אם לא קדמו להן פקודות אחרות. במידה וזה קורה יש להציג הודעת שגיאה רלוונטית למשתמש ולאפשר את המשך פעילות המע'.
* בכל המקומות שבהם מציגים "רשימות של דברים" וצריך לאפשר למשתמש לבחור פריט(ים) מרשימה – יש לאפשר בחירה זו ע"י הצמדת מספר לכל אחת מהאפשרויות ולאפשר לו לבחור על פי המספר המזהה של האפשרות מהרשימה (או באמצעות כמה מספרים במקומות הרלוונטיים). המספרים יתחילו מ 1 (לא מ 0)  
  **אין להניח** שהמשתמש הולך להקליד לכם מלל חופשי של תיאור האפשרות !

להלן רשימת הפקודות שיש לתמוך בהן:

1. קריאת קובץ פרטי המע'

פקודה זו טוענת את פרטיו הראשוניים של גליון מסוים מתוך קובץ נתונים בפורמט XML.

קבצי דוגמא מתאימים הועלו מבעוד מועד לאתר הקורס ואתם מוזמנים להורידם ולבחון אותם בהתאם.   
קבצי הדוגמא אינם ממצים את סך האפשרויות במע', ועל כן אתם יותר ממעודדים לייצר לעצמכם קבצי בדיקה נוספים כדי לבדוק את המע' בצורה יסודית וטובה יותר כאוות נפשכם..

יש לבקש מהמשתמש נתיב מלא לקובץ ה XML אותו הוא רוצה לטעון למע'.   
הנתיב יכול להכיל רווחים בתוכו (למשל "program files") ויש לוודא כי הדבר לא מכשיל אתכם (ולא שהוא אמור).   
הנתיב יכיל רק אותיות באנגלית (לא ג'יבריש של אותיות בעברית וכו')

הקובץ יהיה קובץ XML שפרטיו וחוקיו המפורטים מובאים [בנספח ה'](#xml) לתרגיל זה.   
אתם מצופים לעבור על פרטים אלה ולהתייחס אליהם כחלק אינטגרלי מהגדרת התרגיל.

עליכם לוודא בדיקת קלט לקובץ ה XML ולוודא כי הקובץ מכיל מידע תקין ואמין.

(מובטח כי הקובץ יהיה תקין schema-wise אבל לא בהכרח תקין application-wise...)

בפרט יש לוודא את הפרטים הבאים:

1. הקובץ קיים, והוא מסוג XML (די לבדוק לשם כך כי הוא נגמר בסיומת .xml)
2. אין הפניה לתווית שלא קיימת במסגרת פקודות התוכנית

במידה והקובץ לא תקין יש לדווח זאת למשתמש בצורה ברורה כך שניתן יהיה להבין מה לא תקין בקובץ.   
אין לקרוס על exception במידה וקובץ מתגלה כאינו תקין ; יש לאפשר למע' להמשיך לפעול במצב זה.

(כחלק מבדיקת התרגיל יטענו למערכת קבצים שאינם חוקיים כדי לבדוק מהי התגובה).

במידה והקובץ נמצא תקין – יש לטעון את פרטיו למע' ולדווח על כך שהקובץ נמצא תקין ונטען במלואו למשתמש.  
  
יש לאפשר למשתמש לטעון כמה קבצים אחד אחרי השני (כלומר להפעיל את הפקודה כמה פעמים רצוף).

כל קובץ תקין "דורס" לחלוטין את כל פרטי הקובץ (התקין) שהיה טעון לפניו במע' (ככל שהיה כזה).

כל נסיון טעינה של קובץ תקול לא דורס את פרטי הקובץ (התקין) האחרון שהיה במע' (ככל שהיה כזה)

**דגשים:**

* + - פקודה זו מוצגת ומאופשרת תמיד. אפשר לבחור בה בכל רגע נתון במע'.

1. הצגת תוכנית  
   פקודה זו תציג למשתמש את התוכנית הטעונה במע'.  
   יש להציג את הפרטים הבאים:

* שם התוכנית
* שמות הקלטים (על פי הסדר) שהתוכנית משתמשת בהם (אוסף כל ה xi המופיעים בתוכנית)
* שמות התוויות (על פי הסדר) שהתוכנית משתמשת בהם. אם התווית EXIT מופיעה – יש לציין אותה בסוף הרשימה
* הוראות התוכנית  
  את ההוראות יש להציג על פי סדר הופעתן בקובץ ה XML.  
  ההוראות יודפסו בפורמט הבא:

# <number> (B|S) [LABEL] <command> (cycles)

הסבר:

* + מס' ההוראה. התו # ואחריו מספר המתחיל מ 1
  + סוג ההוראה. יוצג באמצעות התו B עבור פקודה בסיסית או S עבור פקודה סינטטית.  
    מוקף בסוגריים עגולים ().
  + תווית נמצאת בתוך סוגריים מרובעים [ ].   
    כמות התווים בתוך הסוגריים היא 5 כך שהיא תוכל להכיל תווית דו ספרתית (סה"כ 3 תווים)  
    אם אין תווית עדיין מציגים את הסוגריים המרובעים עם 5 רווחים
  + הפקודה עצמה על פי הסוג שלה כפי שהוגדר בחלק הכללי עבור פקודות הבסיס, או כפי שהוגדר [בנספח ג'](#synthetic) עבור הפקודות הסינטטיות.  
    שימו לב כי שמות המשתנים תמיד באותיות קטנות.  
    יש להשתמש בתווים -> עבור סימון החץ
  + כמות ה cycles שהפקודה הזו שווה, מוקפים בסוגריים עגולים ()

דוגמאות להדפסת פקודות:

#3 (B) [ L3 ] IF x1 != 0 GOTO L34 (2)

#4 (B) [ L34 ] x2 <- x2 – 1 (1)

#5 (S) [ ] y <- 0 (1)

(רמז: שאלו את הצ'ט על פרמוט של מחרוזת עם String.format...)

**דגשים:**

פקודה זו לא רלוונטית במידה ואין כרגע קובץ תקין טעון במע'.

1. Expand (הרחבה)

פקודה זו תאפשר לבצע "הרחבה" של התוכנית, כלומר לקחת את כל הפקודות הסינטטיות ולמעשה "לפתוח"/"להרחיב" אותן לפקודות שמהן הן מורכבות.   
עיקבו אחר [נספח ב'](#expansion) כדי להבין את עיקרי המנגנון וכיצד הוא עובד.  
  
יש להציג למשתמש את הדרגה המקסימלית של התוכנית ולבקש ממנו עד לאיזו דרגה הוא ירצה לבצע את ההרחבה.  
(כמובן יש לוודא כי הוא מכניס לכם מספר בתחום הנדרש ואם לא – להציג הודעת שגיאה מתאימה ולתת לו לנסות שוב)  
  
לאחר שהמשתמש הכניס את הדרגה הנדרשת המע' תבצע את ההרחבה עד לדרגה הנדרשת ותציג את כלל הפקודות במבנה המתואר בפקודה מס' 2.  
הפעם, לאחר כל פקודה, באם היא נגזרה מפקודה אחרת, נציין את הפקודה שיצרה אותה.  
יש להוסיף את הסמן >>> ולאחריו את פירוט הפקודה היוצרת באותו אופן שהוגדר בפקודה מס' 2  
  
דוגמא:

#5 (S) [ ] y <- 0 (1) <<< #3 (S) [ ] y <- 5 (2) <<< #1 (S) [ L4 ] IF y = 5 GOTO L5 (2)

**דגשים:**

פקודה זו לא רלוונטית במידה ואין כרגע קובץ תקין טעון במע'.

1. הרצה של תוכנית  
   פקודה זו תאפשר למשתמש להריץ את התוכנית. עיברו על [נספח א'](#execution) כדי להבין את פעילות מנגנון ההרצה.  
   בפתח הרצת התוכנית המשתמש יתבקש לבחור באיזה דרגת הרחבה הוא רוצה להריץ את התוכנית.  
   לאחר מכן יש לקבל מהמשתמש את הקלטים לתוכנית, ואז התוכנית תורחב לדרגה הנדרשת ותורץ על הקלטים שהוכנסו.  
   בסיום ההרצה יש להציג את ערכו של y (התוצאה הפורמלית) ואת ערכם הסופי של כלל המשתנים שנעשה בהם שימוש במהלך הריצה.  
     
   הנה השלבים:
2. הציגו למשתמש את הדרגה המקסימלית וקבלו ממנו את הדרגה בה הוא ירצה להריץ.   
   דרגה 0 פירושה שלא מבצעים שום הרחבה ומריצים את התוכנית AS IS.
3. הציגו למשתמש את משתני הקלט שהתוכנית עושה בהם שימוש וקבלו ממנו את הקלטים לתוכנית.  
   הקלטים יהיו רשימת מספרים מופרדת בתו פסיק (,).  
   שימו לב: המשתמש יכול לתת לכם כמה קלטים שהוא רוצה, הן יותר והן פחות מן המצופה.   
   לכל היותר התוכנית לא תעשה בהם שימוש או תתייחס אליהם כאל 0... (נעים לכם ? לא ? יופי. JS)
4. בצעו את הלוגיקה (הרחבה והרצה של התוכנית)
5. הציגו למשתמש את התוכנית שהורצה בפועל (על פי הכללים של פקודה 2 ו 3)
6. הציגו למשתמש את ערכו של y (תוצאתה הפורמלית של התוכנית)
7. הציגו למשתמש את ערכם של שאר המשתנים בתוכנית. יש להציג בתצורה של v = value למשל z3 = 4  
   יש להציג את כלל המשתנים, הן משתני הקלט (x'ים), הן משתני העבודה (z'ים) והן המשתנה y, וזאת בסדר קבוע:  
   y הוא הראשון, אחריו כל ה x'ים מסודרים על פי הסדר מהנמוך לגבוה ולאחריהם כל ה z'ים מסודרים על פי הסדר מהנמוך לגבוה.
8. הציגו למשתמש את כמות ה cycles שצרכה כלל התוכנית.

**דגשים:**

פקודה זו לא רלוונטית במידה ואין כרגע קובץ תקין טעון במע'.

1. הצגת הסטוריה/סטטיסטיקה

פקודה זו תאפשר את הצגת הסטוריית ההרצות של התוכנית הטעונה כרגע במע'.  
יש לשמור את כלל ההרצות שבוצעו ולהציג עבור כל הרצה את הפרטים הבאים:

1. מס הריצה. מתחיל מ 1
2. דרגת ההרצה
3. הקלטים שהוכנסו
4. ערכו של y בגמר הריצה
5. כמות ה cycles שנצרכו

**דגשים:**

פקודה זו לא רלוונטית במידה ואין כרגע קובץ תקין טעון במע'.

1. יציאה מהמערכת

פקודה זו מסיימת את פעולת התוכנית.

## חלוקה למודולים

בתרגיל זה **חובה** לייצר (לפחות) 2 מודולים (מהם תפיקו בהמשך 2 jar'ים):

* + 1. ממשק ה ui, המציג את התפריטים השונים, אחראי על קליטת קלט מהמשתמש והחזרת הפלט למשתמש.

שימו לב זהו המודול "האקטיבי", המניע את כל המע'. הוא זה האחראי על פנייה ותפעול מנוע המערכת.

כפועל יוצא, כל ההדפסות של מידעים למשתמש (System.out.println) מתבצעות **אך ורק** מתוך מודול זה ; במודול זה יושבת מטודת ה main ; מודול זה אחראי על לולאת תפעול המע' העיקרית, הצגת התפריטים, איסוף הקלט מהמשתמש, הצגת הפלטים למשתמש וכו'.

* + 1. מנוע המערכת, האחראי על קבלת הפקודות (ממודול ה ui), ביצועם והחזרת פלטים מתאימים.

שימו לב שמודול זה "פסיבי", והוא **רק** מגיב לבקשות ולפקודות המתקבלות ממקורות בלתי ידועים לו (בתרגיל זה מודול #1). בתרגילים הבאים מקורות נוספים יפנו אליו לקבלת מידע וחשוב מאוד להקפיד על כך **שמודול זה אינו מכיר/מודע למי פונה אליו.**

## איך מתחילים ? (המלצה...)

התחילו מהבנת מבנה המע'. אני ממליץ להגות על נייר (אמיתי, כמו פעם..) מבנה כללי של המחלקות העיקריות, הקשרים והיחסים בינהן.

צרו פרויקט חדש ב Intellij אשר ישמש כפרויקט האב לכלל התרגילים.   
בתוך הפרויקט צרו מודול נפרד עבור מנוע המע'. המנוע יכיל את כלל החלקים הנדרשים לתפעול המע' (הגדרה של פקודה, תוכנית, הרצה וכו').  
המנוע יחשוף סט של יכולות (הפקודות השונות בתפריט), יחזיק מופע של המע' ויתווך את הפקודות השונות מולה, הלוך וחזור. התחילו בבניית תשתית המנוע (מומלץ לעטוף את המנוע בממשק ולהתחיל לתרגל (ולהתרגל) לחוויית העבודה הנ"ל.)  
  
התחילו בבניית האובייקטים העיקרים (פקודה, תוכנית, מנגנון ההרצה). תוכלו להתחיל להרים את המע' גם לפני קריאת הנתונים מקובץ ולוודא כי היא מתחילה לעבוד. ממשו ראשונה דווקא את פקודה 2 שתאפשר לכם לראות כי בידכם המידע הנדרש ואתם מציגים אותו כראוי.  
בהתאם להתקדמות, המשיכו במימוש פקודה 1 (על כלל היביטי הטעינה), ו/או עיברו לממש את פקודת ההרצה. מייד אחריה ממשו את פקודה 5 – היא אמורה להיות די קלה ופשוטה.  
אני ממליץ לשמור לסוף את כל ההתעסקות עם מנגנון ההרחבה.  
  
לסיום, צרו מודול נוסף שהוא מודול שכבת ה UI (console). זה המודול שבו תשב בסופו של דבר מטודת ה main הראשית שתתפעל את כלל המע'. זה המקום היחיד שבו מוצג פלט (System.out.pritnln) ונאסף קלט (scanner) מהמשתמש. זה המקום המכיל את לולאת התפריט הראשית המניעה את כלל המע'. שכבת ה UI תכיל הפנייה (reference) למופע המנוע (שבתוכו מכיל הפנייה למופע המע') וכך תוכל להעביר ולתרגם לו את הפקודות הנאספות מהמשתמש ולהציג חזרה את הפלטים החוזרים מהמנוע.

## בונוסים

|  |  |  |  |
| --- | --- | --- | --- |
| # | מהות | למה שווה לי ? | כמה שווה לי ? |
| 1 | **שמירה וטעינה של המע'**  בבונוס זה המצב הנוכחי של המע' (כולל כלל מידע הרצות העבר) נשמר לקובץ חיצוני (באיזה פורמט וטכניקה שתבחרו).   יש להוסיף פקודה שמאפשרת לשמור את מצב המע' וגם פקודה המאפשרת לטעון מצב קיים (מקובץ שנשמר זה לא מכבר), וזאת בניגוד לטעינה רגילה מקובץ ה XML של התרגיל.  יש לאפשר למשתמש לבחור את הנתיב המלא כולל שם הקובץ (בלי הסיומת) שהוא היה מעוניין לשמור את המע' אליו (ולטעון אותה ממנו) | כי עם תכנון נכון זה אמור להיות משהו כמו 4-5 שורות... | 5 נקודות **(מעל ל 100)** |

## סבבה, סיימתי. מה ואיך להגיש ?

יש להגיש קובץ zip המכיל:

1. 2 jar'ים (לפחות) שהם כל הקוד שלכם, בצירוף קובץ אצווה (batch) שהפעלתו תריץ את התוכנית

(כלומר תבצע java -jar <class name> וכו').

1. קובץ ההגשה יכיל גם קובץ **readme** שיכיל הסבר על המערכת, בחירותיכם השונות במקומות שבהם היו לכם בחירה, או כל דבר נוסף העולה על דעתכם שחשוב שהבודק ידע.
2. יש לכלול בקובץ ה **readme** גם תיעוד והסבר כללי (וממצה) של המחלקות העיקריות ותפקידם.
3. יש לכלול בקובץ ה **readme** גם פירוט של המגישים שם, ת.ז. ואי מייל זמין ורלבנטי (!!) – במידה ויהיה צורך ליצור קשר.
4. יש לכלול בקובץ ה **readme** קישור ל Github שלכם המכיל את קוד הפרויקט.
5. במידה ומימשתם את מי מהבונוס(ים) לעיל, ציינו את שם הבונוס שמימשתם בראשית קובץ ה readme כדי שהבודק ידע לבדוק בהתאם.

בונוס שימומש אבל לא יתועד – לא ייבדק !

הגשה באיחור, שאינה באישור, תבטל כל מימוש בונוס. אין להגיש באיחור בשביל להספיק לעשות בונוסים.   
תכננו את הזמן בהתאם.

תרגיל 2 – מימוש **S-Emulator** כאפליקציית JavaFX ( 40%) – הגשה: 1.10.25

## פרטים יבשים

צפי תחילת עבודה: **3.9.25** תאריך הגשה: **25.1.10**

צפי זמן לביצוע: 4 **שבועות +** ציון אפשרי מקסימלי: **133**

משקל התרגיל: **40%** קושי: **מאתגר**

**מטרות התרגיל העיקרית**

1. מימוש תצוגת ותפעול המע' כאפליצקיה גרפית JavaFX
2. הוספת תמיכה בציטוט תוכניות (פונקציות)
3. הוספת תמיכה במנגנון ההרכבה
4. הרצה צעד אחר צעד, לפנים או לאחור

## דרישות

1. בתרגיל זה תממשו שכבת משתמש גרפית דרכה המשתמש יוכל לתפעל את המע' ולצרוך את המידע.

השכבה הגרפית תציג את S-Emulator בצורה "נורמלית", כמו כל שאר סביבות הפיתוח בהן ניתן לצפות בקוד ולהריץ אותו.  
בזמן צפייה בקוד, המע' תאפשר לסמן את כל ההוראות שעושות שימוש בתווית מסויימת או במשתנה מסויים, כמו כן לבצע הרחבה הלוך וחזור של התוכנית.

בזמן הרצה של התוכנית ניתן יהיה לקבל מבט על כלל המשתנים שלה, אך גולת הכותרת היא ה debugger: נאפשר להריץ את התוכנית פקודה אחרי פקודה, תוך חשיפת מצב המשתנים על כל צעד וצעד ובכך למעשה להצליח "לדבג" תוכנית ב S.  
(למגדילי הראש תינתן הזכות (!) לפתח גם את היכולת "לרוץ לאחור"... 😊)

מבחינת המנגנון הלוגי תתמכו בממתיק מלאכותי של ציטוט תוכניות (פונקציות), כמו גם נתמוך במנגנון של הרכבת פונקציות.

לטובת הכנת האפליקציה סופק לכם קובץ סקיצה גרפי המציג בקווים כלליים ביותר את מבנה המע' מבחינת מיקומי הרכיבים הגרפים. יש להיצמד אל קווי המתאר הכללים המתוארים בקובץ זה.

1. המשתמש יבחר קובץ תוכנית ראשוני להעלותו.   
   הקבצים שיועלו יהיו רק קבצים מהפורמט של תרגיל 2. עיקבו אחר [נספח ה'](#xml) כדי לראות את ההבדלים בין הסכמות השונות.

כשם שהיה בתרגיל 1, יש לאפשר לטעון כמה קבצים אחד אחרי השני. כל קובץ שנטען בהצלחה מחליף את קודמו. ברגע נתון יש (לכל היותר) רק קובץ אחד הטעון במע'.  
טעינת קובץ ה xml תתבצע באמצעות file chooser dialog בלבד ! (אין להניח שהקובץ יחכה לכם בספרייה ייעודית, או שהבודק יקליד את תוכנו לתוך תיבת טקסט וכו'.).  
הקובץ יכול להימצא בכל ספרייה חוקית על המחשב (כולל ספריה עם רווחים). **נא לוודא כי הדבר אינו מכשיל אתכם**

טעינת הקובץ תתבצע באמצעות Task של JFX. יש להציג progress bar או מכוון התקדמות כרצונכם לתהליך הטעינה.  
תהליך הטעינה עצמו, טכנית, הוא קצר. על כן הוסיפו השהיה קלה (ברמה של שנייה או שתיים) כדי לסמלץ את ההתקדמות.  
  
עליכם לבצע בדיקות תקינות קלט של הקובץ כפי שבצעתם בתרגיל 1.  
בנוסף עליכם לוודא את הדברים הבאים:

* 1. אין הפניה לפונקציה שאינה מוגדרת במסגרת הקובץ מהתוכנית הראשית.
  2. אין הפניה לפונקציה שאינה מוגדרת במסגרת הפונקציות המוגדרות בקובץ.

אם הקובץ אינו תקין אין לטעון אותו למע' ויש להחזיר הודעה מפורטת למשתמש מדוע אינו תקין.  
אם הקובץ תקין הוא נטען למע' בהצלחה וניתן לצפות בפרטיו ולהפעילו בהתאם.

1. המע' תתמוך בשני ממתיקים מלאכותיים נוספים, כשהעקרי בהם הוא ציטוט התוכניות.  
   זהו ממתיק מלאכותי מיוחד שפיתוחו דורש פיתוח דינמי של הפקודה, בניגוד לממתיקים הקודמים.   
   עיקבו אחר [תיאור הפקודה](#quote) לפרטים מלאים בנושא.  
     
   בהמשך לפיתוח ממתיק זה, המע' תתמוך גם ביכולת לבצע "הרכבה" של פונקציות, כלומר קריאה לפונקציה (תוכנית) אחת תשרשר את הפלט שלה כקלט לפונקציה (תוכנית) אחרת.  
   עיקבו אחר [נספח ד'](#composition) לפרטים מלאים בנושא.
2. המע' תציג את פקודות התוכנית הנטענת בטבלה מסודרת שבה יופיעו כלל הפרטים של כל פקודה.  
   העמודות בטבלה יהיו פרטי הפקודה כפי שהוגדרו בתרגיל 1, פקודה 2:

* מספר הפקודה
* סוג הפקודה (B\S)
* תווית
* מבנה הפקודה
* מחזורים (cycles)

המשתמש יוכל לקבל סיכום של כלל הפקודות בתוכנית, ומתוכן חלוקה של כמה פקודות הן בסיסיות וכמה סינטטיות  
  
המשתמש יוכל לבצע כמה פעולות בהקשר הטבלה:

* ביצוע הרחבה/צמצום של התוכנית (expansion). אחרי כל פעולה שכזו הטבלה תציג את התוכנית בדרגה הנדרשת.  
  יש להציג את דרגתה המקסימלית של התוכנית ואת הדרגה הנוכחית שלה בפורמט של Y / X
* בחירת תווית/משתנה והארת (highlighting) כל הפקודות (השורות בטבלה) אשר עושות שימוש בתווית/משתנה זה
* בחירה להצגה של פונקציה שעושים בה שימוש במסגרת התוכנית (במקום התוכנית עצמה)

בבחירת פקודה בטבלה, תוצג בתחתיתה מידע על השרשרת ההסטורית של הפקודה.  
יש להציג את המידע בצורה של טבלה המכילה את המידע על כל פקודה ופקודה (כמו בטבלה הראשית) כך שהפקודה ההסטורית ביותר נמצאת בתחתית הטבלה והפקודה האחרונה (שהולידה את הפקודה המסומנת בטבלה הראשית) נמצאת בראשיתה.

1. המע' תאפשר לבצע את ההרצה של התוכנית המוצגת כרגע (על פי הדרגה הנוכחית) תוך הצגת מצב המשתנים הסופי בתום הריצה (בדומה לפקודה 4 בתרגיל 1).   
   המשתמש יבחר להתחיל ריצה חדשה (כפתור ייעודי).  
   לאחר מכן יתבקש להכניס את הקלטים לריצה. הפעם יש לייצר פקד מתאים שיאפשר לו להכניס קלטים ככל שירצה ולא בדמות מחרוזת עם פסיקים.   
   כשיבחר, יוכל ללחוץ על כפתור תחילת הריצה או תחילת תהליך ה DEBUG (מפורט בהמשך).  
   בסיום הריצה המשתנים יוצגו על פי הסדר שהוגדר בפקודה 4 (תרגיל 1) ויוצגו בטבלה שתציג את שם וערך המשתנה.  
   כמו כן יוצגו כמובן כמות ה cycle'ים שהיא עלתה.  
     
   אם המשתמש בחר להריץ במצב debug, הרי שהוא יוכל להחליט מתי להתקדם לפקודה הבאה (באמצעות כפתור ייעודי)  
   בכל רגע נתון תצבע בטבלה השורה של הפקודה המבוצעת כרגע. אחרי ביצוע כל שורה אפשר יהיה לראות את מצב כלל המשתנים.   
   המשתנים ששונו בפקודה הנוכחית שבוצעה זה עתה ייצבעו בצבע בולט כדי להבליט את השינוי שחל בהם.  
   כמות ה cycle'ים שנצברה תשתנה עם התקדמות בתהליך ריצת התוכנית.  
   במסגרת ה DEBUG יש לתמוך בפקודות הבאות:

* התחלת הריצה במצב DEBUG
* התקדמות צעד אחד קדימה (step over)
* עצירת התוכנית באמצע (stop)
* שחרור ה DEBUG והמשך ריצת התוכנית כרגיל (resume)

1. המע' תאפשר לצרוך את מידע הרצות העבר של התוכנית (בדומה לפקודה 5 בתרגיל 1).  
   המידע יוצג בטבלה שתכיל את המידעים הבאים:

* מס הריצה. מתחיל מ 1
* דרגת ההרצה
* הקלטים שהוכנסו (מחרוזת מופרדת בפסיקים)
* ערכו של y בגמר הריצה
* כמות המחזורים שהתוכנית צרכה

בחירה של הרצה מסויימת מן העבר תאפשר להציג את מצב כלל המשתנים כפי שהם היו בסוף הריצה. (כלומר המשתמש יוכל לראות לא רק את y, אלא את כל שאר המשתנים).  
בפרט יוצגו גם ערכם של משתני הקלט. יש לאפשר למשתמש להשתמש במידע זה ולמעשה להפעיל את התוכנית מחדש על הקלטים הנ"ל (כך יחסך לו הצורך להכניס אותם בעצמו בשנית אם ירצה להפעיל את התוכנית על אותם הקלטים או על קלטים דומים לה במעט)

1. כחלק מבדיקת המערכת ישונה גודל המסך (resize) ותיבדק המע' שלכם במסך בגודל שונה. עליכם לדאוג לסידור נכון של רכיבי ה ui ולוודא את תקינותם גם בגודל קטן. מומלץ מאוד להשתמש ב scroll pane (יש ללמוד עליו לבד) כדי להציג תוכן גדול בתוך שטח מסך קטן.

**למען הסר כל ספק ומניעת כל התחכמות שהיא, אין "לטפל" ב resize ע"י כך שפשוט תמנעו מהמסך להיות resizable.**

**במידה ומימשתם בונוסים בתרגיל הקודם אין הכרח לגרום להם לפעול גם בתרגיל זה, אולם אם זה מתאפשר זה יחמם את ליבי (למען הסר ספק, חימום ליבי אינו מתורגם להעלאת נקודות).**

## איך מתחילים ?

התחילו בהקמת השלד הראשי של המע' ובהתאם לקווי המתאר של האפליקציה.   
  
המשיכו ביכולת לבצע טעינה של הקובץ שתתחבר למימוש הקיים בצד המנוע.  
התרכזו ביצירת הרכיב המציג את טבלת התוכנית הנוכחית על כלל המידעים שבה. ממשו את היכולות הויזואליות של התעסקות עם הצגת המידע בטבלה (highlight, expansion וכו').  
המשיכו למימוש מנגנון ההרצה הרגילה (כפי שהיה בתרגיל 1), תוך הצגת המידעים הרלוונטים.  
בשלב זה אמור להיות לכם קל גם לממש את רכיב ההסטוריה/סטטיסטיקה "על הדרך")

המשיכו להתעסקות עם יכולות ה debugger והנסיון לפרק ריצה לצעד אחר צעד, תוך קבלת כלל הנתונים שלה והצגתם שלב אחרי שלב.  
בשלב זה יש לכם את כלל המנגנון של תרגיל 1 מאופשר ופעיל במסגרת ממשק גרפי. סחטיין !  
עיברו לממש את המנגנונים הלוגים של התרגיל (quoting, composition) במנוע המע' ועכשיו תוכלו לבדוק אותם כהלכה באמצעות הממשק הגרפי וה debugger שבניתם..

## בונוסים

היכן שזה רלבנטי (בונוס 1 ו 2) על הבונוסים להגיע "מכובים" בתור התחלה כך שהבודק "יפתח" אותם רק לטובת הבדיקה שלהם.

|  |  |  |  |
| --- | --- | --- | --- |
| # | מהות | למה שווה לי ? | כמה שווה לי ? |
| 1 | אפשרות החלפת skin למערכת. בבונוס זה תממשו (לפחות) 2 ערכות צבעים נוספות על הערכה הדיפולטית, ותאפשרו למשתמש להחליף את ערכות הצבעים. שימו לב ערכת הצבעים כוללת (לכל הפחות):   1. החלפת רקע של כל המסך 2. החלפת המראה של הכפתורים על המסך 3. החלפת הפונט והגודל של כל ה label על המסך | כי זה בכל זאת תרגיל בממשק גרפי – ומה יותר גרפי מזה ??? | **5 נקודות**  **(ועד ל 100)** |
| 2 | ממשו 2-3 אנימציות המלוות את תהליך ההפעלה  האנימציה צריכה להתבצע לכל היותר במשך 2 שניות  יש לאפשר גם לנטרל את האנימציות הנ"ל, כך שהיא לא תאיט את התקדמות המערכת. | כל כך קל... | 5 נקודות  (ועד ל 100) |
| 3 | **a close up of a woman screaming with the words come back below herStep Back ! (הבנתם ??)** במסגרת הרצת DEBUG, אפשרו למשתמש לבחור לפסוע צעד לאחור, כלומר לגלגל את התוכנית צעד אחד אחורה. במסגרת פעולה זו כלל המשתנים יוצגו בהתאם לפוזיציה שלהם כפי שהייתה בצעד הקודם | לא כזה מסובך כמו שזה נשמע... (וגם, להיות בראיון ולהגיד שמימשתם debugger (!) עם step back (!!) 🡨 !!! | 8 נקודות **(מעל ל 100)** |
| 4 | **TreeTableView** אפשרו למשתמש לבחור להציג את פקודות התוכנית כטבלת-עץ (יש רכיב כזה, נשבע !)  ההיביט העצי יגיע כמובן בכל הקשור ל expansion של התוכנית. בכל רגע נתון התוכנית עצמה היא "העלים" של העץ ואפשר לראות מיידית לכל עלה לאיזה ענף הוא שייך, מי העלים שנמצאים אחריו/לפניו וכו'. תצוגה זו מתווספת לתצוגת הטבלה ההכרחית | לחובבי ה UI שרוצים להרגיש גאים בעצמם ! | 10 נקודות **(מעל ל 100)** |
| 5 | **Specific Expansion** אפשרו למשתמש לבחור פקודה סינטטית מסויימת ולבצע הרחבה/קיפול רק שלה.  אם מבצעים expansion \collapse כללי של כלל הטבלה – הוא כמובן ימשיך לחול על כלל הפקודות | מאוד שימושי, ויכול לעזור מאוד (!!) בשלבי DEBUG של מנגנון ה quote & expansion | 5 נקודות (ועד ל ל100) |
| 6 | **כתבן התוכניות החרוץ ! (הומז' לבעיית הבונה החרוץ.. רק חנון יבין זאת)** הוסיפו מנגנון המאפשר למשתמש לכתוב תוכניות בעצמו ! המנגנון יאפשר למשתמש לפתוח תוכנית חדשה ולבחור את כתיבת הפקודות באמצעות רכיבים גרפים בלבד: יהיה עליו לבחור להוסיף פקודה חדשה, לבחור את הסוג שלה, את הארגומנטים הנוספים שיש לה (ככל שקיימים כאלה) וכו'. את התוכנית שבנה המשתמש אפשר להריץ כרגיל. המשתמש יוכל לשמור את התוכנית לקובץ XML מהפורמט של קבצי התרגיל. במסגרת בונוס זה אין צורך לתמוך ב Quote & composition | כל מילה מיותרת. | 15 נקודות **(מעל ל 100)** |

## סבבה, סיימתי. מה ואיך להגיש ?

יש להגיש קובץ zip המכיל:

1. jar (אחד או יותר) שהוא כל הקוד שלכם, בצירוף קובץ אצווה (batch) שהפעלתו תריץ את התוכנית
2. קובץ ההגשה יכיל גם קובץ **readme** שיכיל הסבר על המערכת, בחירותיכם השונות במקומות שבהם היו לכם בחירה, כמו גם כל דבר נוסף העולה על דעתכם שחשוב שהבודקת תדע.
3. יש לכלול בקובץ ה **readme** גם תיעוד והסבר כללי (וממצה) של המחלקות העיקריות החדשות ותפקידם. (חישבו מה יסייע לבודק להיכנס ביתר קלות לקוד שלכם ולהבין מי נגד מי...)
4. יש לכלול בקובץ ה **readme** קישור ל Github שלכם המכיל את קוד הפרויקט.
5. במידה ומימשתם את מי מהבונוסים לעיל, ציינו את שם הבונוס בראשית קובץ ה readme כדי שהבודק ידע לבדוק בהתאם.

בונוס שימומש אבל לא יתועד – לא ייבדק !

תרגיל 3 – מימוש **S-Emulator** כאפליקציית Client - Server (35%) – הגשה: 29.10.25

## פרטים יבשים

צפי תחילת עבודה: **1.10.25** תאריך הגשה: **29.10.25**

צפי זמן לביצוע: **4 שבועות +** ציון אפשרי מקסימלי: **107**

משקל התרגיל: 35**%** קושי: **ווא'עליה אלל'בחטי**...

**מטרות התרגיל העיקרית**

1. מימוש המערכת כאפליקציית client-server.
2. הוספת מנגנון הקצאות ומשתמשים

## דרישות

1. בתרגיל זה נממש את היכולת לתפעל את S-Emulator כ-ספק "שירותי ענן" המציע למשתמשיו את היכולת להריץ תוכניות S על פי קרדיטים שברשותם.  
   המע' תאפשר למספר משתמשים להתחבר אליה בו זמנית. לכל משתמש יהיו קרדיטים בחשבונו.  
   המע' תאפשר למשתמשים השונים לראות את כלל התוכניות/פונקציות המועלות למע' ולבחור להריץ את מי מהן, כך שכמות ה cycle'ים שדורשת הרצת תוכנית תרד מחשבונו של המשתמש המריץ.  
   לתרגיל זה מצורפת סקיצה כללית לגבי נראות המע'. עיקבו אחרי הקווים הכלליים שהיא מתווה.
2. מודל ההרצה:  
   המע' תדע להציע מגוון ארכיטקטורות שונות (שקולות למעבדים מתוחכמים יותר/פחות).  
   כל ארכיטקטורה יודעת להריץ סט מסויים של פקודות השפה (בסיסיות + סינטטיות).   
   ככל שהארכיטקטורה מתקדמת יותר היא תאפשר להריץ פקודות סינטטיות מורכבות יותר – אולם עלות הפעלתה יקרה יותר.  
   ככל שהארכיטקטורה בסיסית יותר היא תאפשר להריץ תוכניות המכילות פקודות בסיסיות יותר – אולם עלות הפעלת זולה הרבה יותר.  
   הטבלה הבאה מתארת את סוגי הארכיטקטורות המתקיימות במע' והפרמטרים הנלווים להן.  
   כל דור מתקדם כולל את היכולות של סך הדורות שקדמו לו:

|  |  |  |
| --- | --- | --- |
| דור | עלות הפעלה (קרדיטים) | תמיכה בפקודות |
| I | 5 | * NEUTRAL * INCREASE * DECREASE * JUMP\_NOT\_ZERO |
| II | 100 | * ZERO\_VARIABLE * CONSTANT\_ASSIGNMENT * GOTO\_LABEL |
| III | 500 | * ASSIGNMENT * JUMP\_ZERO * JUMP\_EQUAL\_CONSTANT * JUMP\_EQUAL\_VARIABLE |
| IV | 1000 | * QUOTE * JUMP\_EQUAL\_FUNCTION |

1. השרת (tomcat) יכיל את מנוע המערכת ויחשוף את היכולות בדמות endpoints שונים.  
   אף לא אחד מהקליינטים השונים יכול לדבר ישירות אחד עם השני.   
   כל "השיחה" והעברת המידעים בינהם מתבצעת אך ורק דרך/באמצעות השרת.

**S-Emulator  
Server  
(Tomcat)**

**HTTP**

Client

Client

Client

**…**

1. לכל משתמש במע' יש שם ייחודי.  
   ההתחברות למע' מתחילה בעמוד הלוגין בו יידרש המשתמש להכניס את שמו.  
   באם שם המשתמש כבר קיים אזי יש להעביר הודעת שגיאה מתאימה למשתמש ולאפשר לו להתחבר מחדש.  
   (**אין** "להגדיל ראש" ולעבוד עם סיסמאות או כל תהליך של sign up, וזאת כדי להקל על הבודק)  
     
   לאחר התחברות מוצלחת יועבר המשתמש למסך ניהול התוכניות.
2. העלאת קבצים:  
   כל משתמש רשאי להעלות קבצי תוכניות למע' (קובץ ה XML בגרסה של תרגיל 2).  
   הפעם, בניגוד לתרגיל 1 ו 2, המע' צוברת את הקבצים המועלים אליה (ולא מחליפה אותם אחד בשני.)  
   כל קובץ למעשה "מעשיר" את המע' בעוד תוכנית (תיקרא תוכנית ראשית) וייתכן וגם "תורם" פונקציות עזר אשר התוכנית הראשית עושה בהן שימוש.  
     
   הקובץ נבחר ממחשב המשתמש ומועלה אל השרת.   
   העלאת הקובץ יכולה וצריכה להתבצע בהתאם לדוגמא שנלמדה בכיתה.   
   **אין שום צורך** להשתמש בספריות צד שלישי (apache commons וכיוצב') בשביל להעלות את הקובץ

**אין שום צורך** לשמור את תוכן הקובץ בשרת, ויש להימנע מכך בכל מחיר !   
לשרת (של הבודק) אין הרשאות לכך ואתם תקרסו !

כשתוכן הקובץ מגיע לשרת, יבוצעו בו בדיקות הקלט כפי שבוצעו במהלך תרגיל 1 (ולא את אלה שבוצעו במהלך תרגיל 2).   
היות ועכשיו צוברים את הקבצים, ובכל קובץ ייתכנו הפניות לפונקציות, הרי שיש לבדוק ולוודא גם את הדברים הבאים:

1. לכל תוכנית (ראשית) יש שם ייחודי משלה.
2. האם התוכנית עושה שימוש בפונקציה שלא קיימת במע' עד כה (לאו דווקא במסגרת הקובץ עצמו).
3. האם התוכנית מכילה כבר תיאור פונקציה שקיימת במע' (במידה וכן זו תקלה והקובץ לא מתקבל)

באם הקובץ תקול – הודעה מתאימה הכוללת את סיבת התקלה צריכה להופיע למשתמש והקובץ **אינו** מוכר כתקין בצד השרת.   
אם הקובץ תקין – הודעה מתאימה צריכה להגיע למשתמש  
שימו לב: אין צורך לתמוך או להמשיך להציג את העלאת הקובץ באמצעות task של JFX תוך הצגת progress bar או הטמעת השהיה מלאכותית. גם ככה עכשיו זו תהיה פעולה אסינכרונית כי היא מתרחשת מול השרת ותיקח את הזמן שלה...

1. ניהול קרדיטים:  
   המע' תשקף בכל רגע נתון את כמות הקרדיטים הקיימת למשתמש.  
   המשתמש יוכל לטעון את חשבונו בקרדיטים נוספים באמצעות כפתור ייעודי. (לא נממש מנגנון קנייה/סליקה).

תו"כ הפעלת תוכניות (יתואר בהמשך) המשתמש יוכל לראות את כמות הקרדיטים העדכנית שלו מתעדכנת בהתאם לצריכתו.

1. יש להציג את כלל המשתמשים המחוברים כרגע למע'.  
   עבור כל משתמש יש להציג את הפרטים הבאים בטבלה:
   * שם
   * כמה תוכניות ראשיות העלה למע'
   * כמה פונקציות משנה תרם למע'
   * כמות קרדיטים נוכחית
   * כמות קרדיטים שניצל
   * כמות הרצות שהסתיימו

טבלת המשתמשים צריכה להתעדכן באופן רציף כך שתשקף את המשתמשים הקיימים במע'.

1. אם נבחר משתמש מטבלת המשתמשים יש להציג את המידע ההסטורי/סטטיסטי של הרצותיו.  
   (אותו המידע שהוצג בתרגיל 1 ו 2)  
   במידה ומבטלים את בחירת המשתמש – יש להציג את המידע ההסטורי/סטטיסטי של המשתמש עצמו (הקליינט המפעיל)  
   לכל הרצה יש להוסיף מידע (עמודה נוספת) של סוג הארכיטקטורה עליה היא בוצעה.  
   בהצגת המידע הסטטיסטי אפשר לבחור בכפתור ייעודי שיאפשר לעבור למסך 3 כדי להציג את פרטי המשתנים המלאים של הפעלת התוכנית.
2. יש להציג את כלל התוכניות הראשיות שהועלו עד כה למע'.  
   עבור כל תוכנית יש להציג את הפרטים הבאים בטבלה:
   * שם התוכנית
   * שם המשתמש שהעלה את התוכנית
   * כמות הוראות
   * דרגה
   * כמה פעמים היא הורצה במע'
   * עלות קרדיטים ממוצעת של הרצותיה

טבלת התוכניות צריכה להתעדכן באופן רציף כך שתשקף את התוכניות המתקיימות במע'.

בבחירת תוכנית מסויימת המשתמש יוכל ללחוץ על כפתור ייעודי שיעביר אותו למסך 3 - "הרצת תוכנית"

1. יש להציג את כלל הפונקציות שהועלו עד כה למע'.  
   עבור כל פונקציה יש להציג את הפרטים הבאים בטבלה:
   * שם הפונקציה
   * שם התוכנית שבמסגרתה היא הועלתה
   * שם המשתמש שהעלה אותה למע'
   * כמות הוראות
   * דרגה

טבלת הפונקציות צריכה להתעדכן באופן רציף כך שתשקף את כלל הפונקציות הקיימות במע'.

1. מסך הרצת התוכנית מאפשר למשתמש את כלל היכולות כפי שמומשו מבעוד מועד בתרגיל 2.  
   המסך נראה זהה כמעט לחלוטין לאפליקציה שפותחה בתרגיל 2.  
     
   הפונקציונליות נשארת זהה – רק שעכשיו היא מתורגמת לעבודה מול השרת ולא במסגרת תהליך הקליינט:  
   פקודת ההרצה תישלח את המידע הרלוונטי לשרת והביצוע של התוכנית יתבצע הלכה למעשה בשרת.  
   פקודת ה debug גם היא תתבצע מול השרת. כל פקודה של המשתמש תבוצע פרטנית אצל השרת

במסגרת הגדרות ההרצה יש לבחור את ארכיטקטורת המעבד עליה רצים (I – IV).   
במידה ובתוכנית יש פקודות שאינן נתמכות בארכיטקטורה יש לציין את כמותן בשורת הסיכום של טבלת ההוראות.  
הרצת התוכנית תתאפשר רק לאחר שמרחיבים אותה למצב שבו סך הפקודות שנמצאות בה מאופשרות להרצה בארכיטקטורה הנבחרת.  
  
במסגרת ההרצה יש להתייחס לזהות המשתמש המריץ את הפקודה.   
בתחילת ההרצה יש לגבות את כמות הקרדיטים של עלות הארכיטקטורה מהמשתמש. בזמן ההרצה בארכיטקטורה ובדרגה שנבחרה, כל cycle שנצרך במסגרת הרצת התוכנית יתורגם לקרדיטים היורדים מחשבונו של המשתמש (1 cycle = 1 credit)  
במידה ובמסגרת הרצת התוכנית נגמרים למשתמש הקרדיטים – יש להודיע לו על כך בהתאם, ואז ביצוע התוכנית שלו נעצר והוא מוחזר למסך התוכניות.  
  
במידה והמשתמש רוצה להתחיל הרצה (רגילה או debug) וכמות הקרדיטים שלו קטנה מכמות הקרדיטים הממוצעת הנדרשת להרצת התוכנית, בתוספת עלות הארכיטקטורה – יש להתריע על כך בפניו ולא לאפשר לו את הריצה עד שיטען מספיק קרדיטים לחשבונו.

יש להציע למשתמש כפתור ייעודי שיאפשר לו לחזור חזרה למסך התוכניות באם ירצה.

1. חלק מפרטי המידע במע' צריכים להתעדכן בצורה אוטומטית עבור כל משתמש בצורה של Pull כפי שלמדנו בשיעור והודגם בדוגמא המסכמת (אם בשיטה של all או של delta fetching או כל דבר באמצע – כל מקרה לגופו).   
   אפשר לבצע את ה Pull בטווח זמן של עד 2 שניות, אולם זמן סביר ל pull הוא סדר גודל של חצי/שנייה.
2. אין צורך לבצע שמירה של נתונים מעבר למופע הנוכחי של השרת. במילים אחרות כשהשרת יורד - כל הנתונים נעלמים.

לא שומרים את המשתמשים שנרשמו, הגליונות וההרשאות שניתנו וכו'.

1. יש לדאוג ולוודא כי מסכי המע' לסוגם מתנהגים יפה בresize ולא מחסירים שום פרט
2. חלק ניכר מהרכיבים כבר קיימים לכם במסגרת תרגיל 2. יש לשאוף למחזר ולהשתמש בהם כמה שרק אפשר.

## חלוקה למודולים

יש לייצר מודול נוסף שממנו ייבנה war המכיל את כל ה jar'ים הנדרשים (Jar של מנוע המע' ; gson.jar ; אולי אחרים ?).  
  
בנוסף, יש לייצר מודול חדש עבור האפליקציה של תרגיל 3, אשר יכיל את כלל המסכים והרכיבים הנדרשים לה.  
מודול זה יכול וצריך להתבסס על הרכיבים הקיימים לכם כבר במסגרת תרגיל 2.

## איך מתחילים ?

אני ממליץ להתחיל במעבר יסודי על הדוגמא המסכמת של הקורס.   
היא תקנה לכם שליטה בסיסית בדרך העבודה עם http client ובעבודה מול השרת ומהלך החיים והתפקידים השונים של כל רכיב ושחקן בסיפור.

אני מציע להתחיל לפתח את המע' כך שאתם לוקחים את האפליקציה של תרגיל 2 ולמעשה "מכשירים" אותה במסגרת עבודה של שרת-לקוח (כלומר כל הפעולות מול המנוע יתורגמו לבקשות http וכו')  
לאחר שתסיימו זאת התחילו בלפתח את מודל המשתמשים בצד השרת ובהתאם את מסך הלוגין של משתמש ואחריו את מסך התוכניות. התחילו מלאפשר להעלות קבצים ולראות את כלל התוכניות הנצברות במע'.   
כל משתמש שמתחבר למע' יוכל לראות רשימה זו גם כן ולקבל עליה עדכונים שוטפים וכו'.

המשיכו בפיתוח טבלת הפונקציות והמידע עליהן שנגזר מהתוכניות הראשיות.  
  
עכשיו המשיכו לממש את השינויים במנגנון הריצה המתחשבים בקרדיטים של המשתמש.  
לבסוף התעסקו עם היכולת להציג סטטיסטיקה וריצות עבר עבור כלל המשתמשים

לאורך פיתוח האפליקציה יש לשאוף להשתמש כמה שאפשר ברכיבים הקיימים בתרגיל 2, גם כדי להקל משמעותית על עומס הפיתוח ובעיקר כדי להתנסות (ולהוכיח לעצמכם !) איך אפשר לפתח רכיב פעם אחת ולהשתמש בו בהקשרים נרחבים אחרים.

ממליץ מאוד (!), לאורך כל העבודה, להקפיד לעבוד ולבחון את עצמכם דרך postman כאמצעי וידוא כי הסרבלטים עובדים כראוי ללא תלות בקליינט כזה או אחר. כך תוכלו לוודא כי צד השרת עובד טוב (up to a degree) טרם כתיבת החלק הרלבנטי בקליינט.

## בונוסים

|  |  |  |  |
| --- | --- | --- | --- |
| # | מהות | למה שווה לי ? | כמה שווה לי ? |
| 1 | Chat:  אפשרו למשתמשים לקיים chat בינהם תו"כ התחברותם למע'.  ה chat צריך להתנהל כולו אל מול השרת בצורה שבה כולם רואים את מה שכולם כותבים. (מאוווווווווווד דומה לדוגמא הסופית) | היש מדהים מזה ??? | 5 נקודות  (ועד ל 100) |
| 2 | יחסי תוכניות ופונקציות הציגו למשתמש בצורה ברורה את היחסים בין תוכניות ראשיות לפונקציות שהן משתמשות בהן. בחירת תוכנית ראשית (בטבלת התוכניות) תדגיש את הפונקציות שהיא עושה בהן שימוש בטבלת הפונקציות; בחירת פונקציה (בטבלת הפונקציות) תדגיש את כל התוכניות שעושות בהן שימוש | התעסקות UI שבשלב זה כבר משולה ל'פיסת עוגה'... | 5 נקודות  (ועד ל 100) |
| 3 | השלכות הארכיטקטורה  הדגישו למשתמש את השורות בטבבלת ההוראות הנתמכות בארכיטקטורת המעבד (בצורה מסויימת, לבחירתכם, נגיד בירוק) ואת ההוראות שאינן נתמכות בארכיטקטורה (בצורה מסויימת, לבחירתכם, נגיד באדום) | בקטנה... | 5 נקודות  (ועד ל 100) |

## סבבה, סיימתי. מה ואיך להגיש ?

יש להגיש קובץ zip המכיל:

1. WAR אחד בדיוק אשר יושם בספריית tomcat\webapp ויעבור deployment אוטומטי.   
   על WAR זה לכלול את כל התלויות שלכם. אין להניח כי יסופקו לכם תלויות מבחוץ (למשל Gson וכו').
2. ספרייה עבור הקליינט החדש, בה יישבו כל ה jar'ים הנדרשים כדי לתפעל את האפליקציה הרלבנטית.   
   יש לספק לאפליקציה קובץ batch שמפעיל אותה (כמו שהיה בתרגיל 1 ו 2). קובץ זה ישב גם כן בספריה הרלוונטית.  
   שימו לב כי האפליקציה שלכם צריכה להכיר אוטומטית את השרת ואת ה context path הרלוונטי לה   
   אפשר כמובן להניח את localhost:8080 כדומיין של השרת; אפשר להניח כי לא נשנה את שם קובץ ה WAR שהגשתם.
3. קובץ ההגשה יכיל גם קובץ **readme** שיכיל הסבר על המערכת, בחירותיכם השונות במקומות שבהם היו לכם בחירה, כמו גם כל דבר נוסף העולה על דעתכם שחשוב שהבודקת תדע.
4. יש לכלול בקובץ ה **readme** גם תיעוד והסבר כללי (וממצה) של המחלקות העיקריות החדשות ותפקידם. (חישבו מה יסייע לבודק להיכנס ביתר קלות לקוד שלכם ולהבין מי נגד מי...)
5. יש לכלול בקובץ ה **readme** קישור ל Github שלכם המכיל את קוד הפרויקט.
6. במידה ומימשתם את מי מהבונוסים לעיל, ציינו את שם הבונוס **בראשית** קובץ ה readme כדי שהבודק ידע לבדוק בהתאם.

בונוס שימומש אבל לא יתועד – לא ייבדק !

הגשה באיחור, שאינה באישור, תבטל כל מימוש בונוס. אין להגיש באיחור בשביל להספיק לעשות בונוסים.   
תכננו את הזמן בהתאם.

נספח א' – מנגנון ההרצה

נספח זה מתאר היביטים נדרשים במנגנון ההרצה של התוכנית.  
  
התוכנית היא אוסף של פקודות סדרתיות המתרחשות על פי רוב אחת אחרי השנייה.   
יחד עם זאת קיימת פקודה אחת שמאפשרת למעשה לבצע "קפיצה" אל פקודה אחרת שאינה (בהכרח) הפקודה "הבאה בתור" (פקודה מס' 3).  
על כן, במסגרת הרצת התוכנית יש לדעת לזהות את הפקודה הבאה בתור בצורה לוגית ולא בצורה סדרתית רגילה.

במסגרת הרצת התוכנית יש לאתחל ולשנות את מצבם של המשתנים השונים שעושים בהם שימוש.  
זיכרו כי כל משתנה מתחיל מהערך 0 ואין צורך (ואין דרך) לציין זאת במפורש.   
(כלומר אם פתאום יש פקודה המשתמשת ב 700Z לראשונה – הרי שערכו הוא 0).  
הוא הדין גם עבור משתני קלט (X'ים) אשר לא הוכנסו לריצה – ערכם כמובן 0.  
הוא הדין, כמובן, עבור משתנה הפלט – y

הרצת תוכנית נעצרת כאשר מסיימים את הפקודה האחרונה בסדר הפקודות, או אם נתקלים בפקודה 3 השולחת אותנו אל התווית EXIT.   
שימו לב: לא תקבלו הרצה של תוכנית שלא מסתיימת (זאת בניגוד לקורס חישוביות, ששם זה הכיף !)

בבואכם לממש פקודה סינטטית, יש לממשה בצורה "פשוטה" במסגרת הכלים בשפת ג'אווה – כלומר אין צורך לעקוב אחר החוקים של שפת S.  
דוגמא: הפקודה הסינטטית V ← 0 מאפסת את ערכו של V. בביצוע הפקודה אפשר פשוט לשלוף את המשתנה הרלוונטי ולעדכן אותו ל 0 (על אף שאין פקודה כזו בשפת S הפורמלית).

כל פקודה מגדירה כמה "מחזורי מעבד" (Cycles) היא לוקחת.  
  
2 הפקודות הבסיסיות (הוספת והורדת 1) לוקחות מחזור אחד כל אחת.  
פקודת הקפיצה (מספר 3) לוקחת 2 מחזורים.  
פקודת ה NO-OP לוקחת 0 מחזורים (אפקטיבית לא מבצעת כלום)  
  
כל פקודה סינטטית מגדירה כמה מחזורים היא "עולה" לביצוע.  
כאשר מבצעים את הפקודות השונות במסגרת הרצת תוכנית, יש לסכום את סך המחזורים שלהן וכך נדע כמה מחזורי מעבד ארכה התוכנית.

נספח ב' – מנגנון ההרחבה

נספח זה מתאר את ההיביטים השונים של מנגנון ההרחבה

מנגנון ההרחבה הוא מנגנון אשר לוקח פקודה מסויימת ולמעשה מייצר במקומה אוסף פקודות אחר המקיים את הלוגיקה שלה עצמה.   
הפקודות הבסיסיות אינן ניתנות להרחבה. הן בסיסיות ומייצגות את יחידת הביצוע הקטנה ביותר של המעבד אותה לא ניתן לפרק.  
כלל הפקודות הסינטטיות ניתנות להרחבה (לכן הן נקראות סינטטיות...)

היות וייתכן שפקודות סינטטיות יהיו מורכבות בעצמן מפקודות סינטטיות אחרות – הרי שייתכן שאחרי ההרחבה הראשונה נקבל סט פקודות אותו ניתן להרחיב בשנית. ואז בשלישית וכו', עד אשר נגיע לתוכנית המכילה פקודות בסיסיות בלבד.  
(**שימו לב**: מובטח כי לא יתקיים מעגל כך שפקודה סינטטית א' תסתמך על פקודה סינטטית ב' אשר בתורה תסתמך על פקודה סינטטית א' – וכך למעשה תמיד נסיים בתוכנית המכילה פקודות בסיסיות בלבד.)

לכמות ההרחבות שנבצע בתוכנית נתונה, עד אשר נגיע לתוכנית המורכבת מפקודות בסיסיות בלבד - נקרא "דרגת התוכנית" (degree).  
בהינתן תוכנית מסויימת נרצה להגדיר מהי הדרגה המקסימלית שלה.  
לכל פקודה ניתן להגדיר את דרגתה – כמה מחזורי הרחבה צריך לעבור עד אשר מקבלים אוסף פקודות בסיסיות בלבד.  
כך, עבור תוכנית מסויימת, נוכל לחשב את דרגתה המקסימלית על ידי כך שנמצא את הפקודה עם הדרגה המקסימלית.  
הדרגה של הפקודות הבסיסיות היא 0.  
עליכם יהיה לחשב ולהסיק לבד עבור כל פקודה סינטטית מהי דרגתה.  
  
נסמן ב Q את פקודת המקור, אותה אנו רוצים להרחיב.  
לאחר הרחבה של פקודה Q נקבל אוסף של פקודות (בסיסיות או סינטטיות). הפקודות הנ"ל למעשה נוצרו בעקבות פקודה Q.  
המע' תדע לשמור מידע עבור כל פקודה מאיזו פקודה היא הורחבה / נגזרה / נוצרה / נולדה וכו'.  
בהינתן פקודה נוכל לתאר את שרשרת ההסטוריה שהולידה והביאה עד הלום.

דגשים (חשובים !) למנגנון ההרחבה:

1. אם על פקודה Q יש תווית במקור – תווית זו צריכה להישמר ולהופיע על הפקודה הראשונה בפקודות ההרחבה.  
   כך למעשה אם יש בהמשך התוכנית המקורית פקודה שתכננה לקפוץ ל Q, אזי אחרי ההרחבה היא תקפוץ אל הפקודה הראשונה של ההרחבה

**[L1] y = 0**

**[ ] x2 ← x2 - 1**

**[ ] y ← y + 1**

…

[ ] x1 ← x1 + 1

[ ] IF x1 = 2 GOTO L1

**[L1] y ← x2**

[ ] x1 ← x1 + 1

**…**

[ ] IF x1 = 2 GOTO L1

במידה וגם במסגרת ההרחבה נדרשים לשים תווית על הפקודה הראשונה – הרי שאז אפשר להקדים את כל ההרחבה בפקודת הניוטרל (שלא עושה כלום ולכן לא משנה את מהות התוכנית) ועליה לשים את התווית המקורית.

1. אם במסגרת הרחבת פקודה Q עושים שימוש במשתני עזר כלשהם (Z'ים) – אזי אחרי ההרחבה צריך לוודא כי כל ה Z'ים אינם חוזרים על עצמם, וכי כל הרחבה מגיעה עם סט ה Z'ים שלה
2. אם במסגרת הרחבת פקודה Q עושים שימוש בתוויות כלשהן (L'ים) – אזי אחרי ההרחבה צריך לוודא כי כל ה L'ים אינם חוזרים על עצמם, וכי כל הרחבה מגיעה עם סט ה L'ים שלה

נספח ג' – תיאור הפקודות

S-Emulator תתמוך בארבע פקודות בסיסיות ובמגוון פקודות סינטטיות.  
  
פקודות בסיסיות הן הפקודות בבסיס השפה ומהוות פעולה אטומית אחת שלא ניתנת לפירוק או לביטוי באמצעות פקודות אחרות.  
פקודות סינטטיות הן פקודות אשר ניתנות לפירוק/להרחבה לפקודות אחר השומרות כמכלול על משמעות הפקודה המקורית.  
פקודה סינטטית יכולה להתפרק לפקודות סינטטיות אחרות.  
שימו לב: מובטח כי לא יתקיים מעגל, כך שפקודה סינטטית Q מתפרקת תוך שימוש בפקודה סינטטית P ו P מתפרקת תוך שימוש בפקודה סינטטית Q.

בנספח זה יתוארו כלל הפקודות שהמע' תתמוך בהן ואשר עליכם לממש במסגרת התרגילים השונים.  
שימו לב: יש פקודות שנדרשות למימוש רק במסגרת תרגיל 2. אין צרוך לממש את כולן בתרגיל הראשון (אם כי במימוש נכון – זה אמור להיות פיסת עוגה, ויחסוך לכם עבודה בהמשך הדרך...)

עבור כל פקודה יוגדר השם שלה, כמות המחזורים שהיא צורכת, משמעותה, האופן בו היא תוצג למשתמש, וסט הפקודות שהיא מתפרקת אליו (היכן שרלוונטי)  
עליכם לממש את הפקודות הנ"ל במנגנון המימוש שלכם ולאפשר להשתמש בהן (אם בצורתן המקורית Q ואם אחרי הרחבתן) במסגרת התוכניות במע'.

הנה רשימת הפקודות שהמערכת תומכת בהן:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | שם הפקודה | תלויה ב | cycles | נדרש מתרגיל |
| 1 | Increase |  | 1 | 1 |
| 2 | Decrease |  | 1 | 1 |
| 3 | Jump Not Zero |  | 2 | 1 |
| 4 | Neutral |  | 0 | 1 |
| 5 | Zero Variable |  | 1 | 1 |
| 6 | Goto Label |  | 1 | 1 |
| 7 | Assignment | Goto Label | 4 | 1 |
| 8 | Constant Assignment | Zero Variable | 2 | 1 |
| 9 | Jump Zero | Goto Label | 2 | 1 |
| 10 | Jump Equal Constant | Goto Label, Assignment, Jump Zero | 2 | 1 |
| 11 | Jump Equal Variable | Jump Zero, Goto Label | 2 | 1 |
| 12 | Quotation | Assignment | 5 | 2 |
| 13 | Jump Equal Function | Quotation,  Jump Equal Variable | 6 | 2 |

* **Increase**

מטרה:   
הפקודה מקדמת באחד את ערכו של משתנה מסויים (מסומן כאן כ V).  
זוהי פקודת בסיס ואין לה הרחבה.  
Cycles: 1  
תצוגת משתמש: V ← V + 1 (דוגמא: (x2 ← x2 + 1  
שם הפקודה בקובץ: INCREASE   
ארגומנטים לפקודה: אין

* **Decrease**  
  מטרה:   
  הפקודה מחסרת באחד את ערכו של משתנה מסויים (מסומן כאן כ V).  
  במידה וערכו כבר 0 – אז הוא נשאר 0 ולמעשה אין לה משמעות.  
  זוהי פקודת בסיס ואין לה הרחבה.  
  Cycles: 1  
  תצוגת משתמש: V ← V - 1 (דוגמא: (x2 ← x2 - 1  
  שם הפקודה בקובץ: DECREASE  
  ארגומנטים לפקודה: אין
* **Jump Not Zero**  
  מטרה:   
  הפקודה בודקת את ערכו של משתנה מסוים (מסומן כאן כ V).   
  אם הוא שונה מ 0 אז היא קופצת אל תווית כלשהיא(מסומנת כאן כ L).   
  אם ערכו 0 – היא ממשיכה לפקודה הבאה אחריה.  
  זוהי פקודת בסיס ואין לה הרחבה.  
  Cycles: 2  
  תצוגת משתמש: IF V!=0 GOTO L (דוגמא: (IF x2 != 0 GOTO L1  
  שם הפקודה בקובץ: JUMP\_NOT\_ZERO   
  ארגומנטים לפקודה:
  + JNZLabel: שם התווית אליה קופצים (1L)
* **Neutral**  
  מטרה:   
  הפקודה משימה את ערכו של משתנה מסויים (מסומן כאן כ V) בעצמו, כלומר אפקטיבית לא משנה אותו ואין לה שום אפקט על התוכנית.  
  זוהי פקודת בסיס ואין לה הרחבה.  
  Cycles: 0  
  תצוגת משתמש: V ← V (דוגמא: (x2 ← x2  
  שם הפקודה בקובץ: NEUTRAL  
  ארגומנטים לפקודה: אין
* **Zero Variable**  
  מטרה:   
  הפקודה עובדת על/בהקשרו של משתנה מסויים ומשימה לו את הערך 0.  
  דרך העבודה תהיה פשוט להוריד את ערכו של המשתנה ב 1 ולהמשיך לעשות זאת כל עוד הוא לא הגיע ל 0  
  במסגרת ההרחבה יעשה שימוש בתווית חופשיה תסומן כאן ב 1L  
  Cycles: 1  
  תצוגת משתמש: V ← 0 (דוגמא: (x2 ← 0  
  שם הפקודה בקובץ: ZERO\_VARIABLE  
  ארגומנטים לפקודה: אין

L1x2 ← x2 – 1

IF x2 ≠ 0 GOTO L1

* **Goto** **Label**  
    
  מטרה:   
  הפקודה עוברת לתווית אחרת.  
  פקודה זו אינה עובדת בהקשר של משתנה מסויים, ומבצעת את הקפיצה ללא שום התנייה.  
  דרך העבודה תהיה לבחור משתנה עבודה כלשהוא (מיוצג כאן באמצעות z1) להגדיל אותו ב 1 (כך שהוא בטוח לא 0) ואז לקפוץ לתווית המסופקת (מיוצגת כאן באמצעות L)  
  Cycles: 1  
  תצוגת משתמש: GOTO L (דוגמא: GOTO L1)

z1 ← z1 + 1

IF z1 ≠ 0 GOTO L

שם הפקודה בקובץ: GOTO\_LABEL  
ארגומנטים לפקודה:

* + gotoLabel: שם התווית אליה קופצים

* **Assignment**  
    
  מטרה:   
  הפקודה עובדת על/בהקשר של משתנה מסויים ומשימה לו את ערכו של משתנה אחר  
  דרך העבודה תהיה לאפס את משתנה המטרה (יסומן כאן כ V) ואז להוריד את ערכו של המשתנה המושם (יסומן כאן כ 'V) תוך שמעלים את ערכו של משתנה עבודה כלשהוא (מיוצג כאן באמצעות z1).  
  כעת הרסנו את ערכו של המשתנה המושם. לכן נחזור ונעלה אותו ואת משתנה המטרה בזמן שאנו מורידים את ערכו של משתנה העבודה ב 1 כל פעם, וזאת כל עוד ערכו של משתנה העבודה אינו 0  
  במסגרת ההרחבה יעשה שימוש בשלוש תוויות חופשיות, מסומנות כאן ב L1,L2,L3  
  Cycles: 4  
  תצוגת משתמש: 'V ← V (דוגמא: (z2 ← x4

V ← 0

IF V’ ≠ 0 GOTO L1

GOTO L3

L1 V’ ← V’ – 1

z1 ← z1 + 1

IF V’ ≠ 0 GOTO L1

L2 z1 ← z1 - 1

V ← V + 1

V’ ← V’ + 1

IF z1 ≠ 0 GOTO L2

L3 V ← V

שם הפקודה בקובץ: ASSIGNMENT  
ארגומנטים לפקודה:

* + assignedVariable: שם המשתנה אותו משימים ('V)

* **Constant** **Assignment**

V ← 0

V ← V + 1

V ← V + 1

….

V ← V + 1

מטרה:   
הפקודה מבצעת השמה של קבוע (K) אל תוך משתנה.  
**שימו לב**: K ידוע בעת פיתוח הפקודה.  
דרך העבודה תהיה לאפס את משתנה המטרה (יסומן כאן כ V) ואז לבצע K פעמים פעולה של הוספת אחת למשתנה המטרה.  
Cycles: 2  
תצוגת משתמש: V ← K (דוגמא: (z2 ← 5

K פעמים

שם הפקודה בקובץ: CONSTANT\_ASSIGNMENT  
ארגומנטים לפקודה:

* + constantValue: מספר. ערך הקבוע (K) אשר משימים למשתה V
* **Jump Zero**  
    
  מטרה:   
  הפקודה מבצעת קפיצה לתווית (תסומן באמצעות L) אם משתנה מסויים (יסומן כאן כ V) שווה ל 0.  
  דרך העבודה תהיה לבדוק אם V הוא לא 0 (פקודת בסיס) ולקפוץ לסוף הפיתוח של הפקודה. אחרת (הוא 0) לקפוץ לתווית L.  
  במסגרת ההרחבה יעשה שימוש בתווית חופשית L1   
   Cycles: 2  
  תצוגת משתמש: IF V = 0 GOTO L1 (דוגמא: (IF X2 = 0 GOTO L1

IF V ≠ 0 GOTO L1

GOTO L

L1 y ← y

שם הפקודה בקובץ: JUMP\_ZERO  
ארגומנטים לפקודה:

* + JZLabel: שם התווית אליה קופצים (1L)
* **Jump Equal Constant**

מטרה:   
הפקודה מבצעת קפיצה לתווית (תסומן באמצעות L) אם משתנה מסויים (יסומן כאן כ V) שווה ל ערכו של קבוע K.  
**שימו לב**: K ידוע בעת פיתוח הפקודה.  
דרך העבודה תהיה לבצע K פעמים הורדות של V עם בדיקה אם הגיע ל 0.   
אם ערכו 0 אזי V < K ולכן לא מבצעים קפיצה ל L.  
אם אחרי K ההורדות V עדיין שונה מ 0 – סימן ש V > K ולכן לא מבצעים קפיצה ל L.   
אם זה לא קרה – סימן שאחרי K הורדות הוא בדיוק 0 ואז מבצעים קפיצה ל L.  
במסגרת ההרחבה יעשה שימוש בתווית חופשית L1, ובמשתנה חופשי z1  
 Cycles: 2  
תצוגת משתמש: IF V = K GOTO L (דוגמא: (IF X2 = 5 GOTO L

z1 ← V

IF z1 = 0 GOTO L1

z1 ← z1 – 1

IF z1 ≠ 0 GOTO L1

GOTO L

L1 y ← y

K פעמים

שם הפקודה בקובץ: JUMP\_EQUAL\_CONSTANT  
ארגומנטים לפקודה:

* + JEConstantLabel: שם התווית אליה קופצים (L)
  + constantValue: ערך הקבוע (K) אליו משווים את המשתנה (V)
* **Jump Equal Variable**

z1 ← V

z2 ← V’

L2 IF z1 = 0 GOTO L3

IF z2 = 0 GOTO L1

z1 ← z1 – 1

z2 ← z2 – 1

GOTO L2

L3 IF z2=0 GOTO L

L1 y ← y

מטרה:   
הפקודה מבצעת קפיצה לתווית (תסומן באמצעות L) אם משתנה מסויים (יסומן כאן כ V) שווה ל ערכו של משתנה אחר (יסומן כאן 'V).  
דרך העבודה תהיה להוריד את ערכם של המשתנים ב 1 כל פעם עד שאחד מהם מגיע ל 0. אם השני גם הגיע ל 0 – או אז מבצעים קפיצה ל L, אחרת לא.   
במסגרת ההרחבה יעשה שימוש בתווית חופשית L1,L2,L3, ובשני משתנים חופשיים z1,z2  
 Cycles: 2  
תצוגת משתמש: IF V = V’ GOTO L (דוגמא: (IF X2 = z1 GOTO L

שם הפקודה בקובץ: JUMP\_EQUAL\_VARIABLE  
ארגומנטים לפקודה:

* + JEVariableLabel: שם התווית אליה קופצים (L)
  + variableName: שם המשתנה אליו משווים ('V)
* **Quote Program** **(ציטוט תוכניות)**  
    
  מטרה:  
  מטרת פעולה זו היא לאפשר להשתמש בפלט של תוכנית מסויימת (Q) במסגרת תוכנית אחרת (P)  
  בפשטות: תוכנית היא אוסף של פקודות שמבצעות משהו. התוצר שלה יימצא ב y. במסגרת השימוש בפעולה זו "ניצוק" את כל השורות של התוכנית המצוטטת (Q) במקום המתאים בתוך התוכנית הראשית (P) ונשים את ערכו של y של Q בתוך משתנה היעד הרלוונטי שהוגדר אצל P  
    
  הבעיה עלולה להתעורר מאחר ומי שכתב את Q כתב אותה כתוכנית עצמאית, כלומר היא משתמשת בתוויות, במשתני קלט, משתני עבודה ו y מתוך מחשבה על עצמה בלבד.   
  אם נשלב אותה בתוך P, בהחלט ייתכן כי גם היא במקרה (או שלא) משתמשת באותם (חלקם או כולם) תוויות, משתני הקלט, משתני העבודה, ובוודאי ב y לטובת האישית.   
  כדי שהמשתנים וערכם לא יתערבבו יש צורך לעשות הפרדה.  
    
  הנה קווים כללים שיש להקפיד עליהם במסגרת תהליך החילוף וציטוט השורות:
  + יש לאתר כל משתני הקלט ש Q עושה בהם שימוש (x1..xn) ולהחליף אותם באוסף של משתני עבודה חופשיים (z1..zn) ב P
  + יש להחליף את y של Q במשתנה עבודה חופשי נוסף (zy) ב P
  + יש לאתר את כל התוויות ש Q עושה בהם שימוש (L1..LM) ולהחליף אותם באוסף של תוויות חופשיות אחרות ב P.  
    שימו לב שיש להחליף הן את התוויות של ההוראות והן תווית היעד במשפטי ה GOTO השונים.
  + יש לשתול n פקודות השמה בין zi ← xi (אתחול כל תחליפי משתני הקלט ב Q עם האברים הרלוונטים שנשלחים אליה מ P)
  + יש להחדיר את כל ההוראות של Q (אחרי כל החילופים שתוארו לעיל) אל תוך המקום המתאים ב P
  + יש להוסיף הוראה אחרונה ב P, מייד אחרי שורות Q של השמה של ערכו של zy אל תוך ערך המשתנה V. הוראה זו צריכה להופיע עם תווית חופשית נוספת ב P (Lend).
  + יש לעבור על כל הוראות Q ובמידה ויש שם משפט GOTO EXIT – יש להחליף אותו עם Lend (כך למעשה מה שהיה "יציאה" מהתוכנית Q עם איזה ערך שלא יהיה ב y – יהיה מתורגם לשורה האחרונה של הציטוט שלמעשה משימה את ערכו של zy במשתנה היעד V)

כל תוכנית ב S משולה להפעלת פונקציה מתמטית.   
ציטוט Q בתוך P משול אם כן להפעלה של פונקציה על ארגומנטים שהיא מקבלת לתוכה.  
  
בכתיב שלנו, מטעמי נוחות (ראו בהמשך מנגנון ההרכבה), נשתמש בתצורת הכתיב הבאה לטובת "הפעלת פונקציה":

**(**<Function\Program Name>,V1,V2,…,VN**)**

הסבר:

* + הסוגריים העגולים מסמנים הפעלה של פונקציה
  + מייד לאחריהם יופיע שם הפונקציה. במקרה שלנו הוא שם התוכנית
  + לאחריהם, מופרדים בפסיקים, יופיעו כל הארגומנטים שמועברים לפונקציה המופעלת.
  + סדר הופעת הארגומנטים יתפרש בתוך הפונקציה המופעלת כ x1..xn.
  + פונקציה שלא מקבלת שום קלט תופעל ע"י עטיפתה בסוגריים עגולים בלבד
  + מובטח כי מחרוזת הפעלת הפונקציה (...) לא תכיל רווחים  
    (כלומר בין שמות המשתנים והפונקציה לפסיקים המפרידים בינהם אין שום רווחים)

במידה ומשתמשים בפקודה זו, אזי הקובץ יכיל גם אלמנט המכיל את הגדרת הפונקציה.  
לכל פונקציה יצויין:

* + שם פורמלי (בו משתמשים כדי להפנות אליה)
  + שם תצוגה למשתמש – זהו השם בו יש להשתמש כשמציגים את הפקודה למשתמש
  + רשימת פקודות: זוהי התוכנית עצמה המייצגת את מימוש הפונקציה (אותה יש לצטט – Q)

Cycles: 5 (+ כמות ה cycles של התוכנית המצוטטת)  
תצוגת משתמש: V ← (Q,V1,V2,..) (דוגמא: x2 ← (ID,z58))

שם הפקודה בקובץ: QUOTE  
ארגומנטים לפקודה:

* + functionName: שם הפונקציה להפעלה (לציטוט) - Q
  + functionArguments: מחרוזת. הארגומנטים אותם צריך להעביר ל Q.  
    הארגומנטים השונים יופרדו בפסיק (,)

דוגמא:

P:

L1 z1 ← (Q,x1,y)

z1 ← z1 + 1

x2 ← x2 - 1

IF x2 ≠ 0 GOTO L1

נתונות P ו Q כמודגם.

תהליך ההחלפה של Q:

* + Q משתמשת בקלט x1 ו x2 ועל כן נמצא 2 משתני עבודה חופשיים ב P שיחליפו אותם לכל אורך Q. יהיו אלה z2,z3
  + Q משתמשת במשתנה עבודה אחד z1 ועל כן נמצא משתנה עבודה חופשי ב P שיחלף אותו לכל אורך Q. יהיה זה z4
  + Q כותבת לתוך y ולכן נחליף אותו במשתנה עבודה חופשי נוסף ב P. יהיה זה z5
  + Q משתמשת בתווית L1 ועל כן נמצא תוית חופשית ב P שתחליף אותה ב Q בכל מופעיה. תהיה זו L2

Q:

L1 IF x1 = 0 GOTO EXIT

x1 ← x1 – 1

x2 ← x2 - 1

z1 ← z1 + 1

y ← y + 1

IF x1 ≠ 0 GOTO L1

* + Q משתמשת ב EXIT ולכן נחליף אותו בתווית חופשית ב P. תהיה זו L3.

Q’:

L2 IF z2 = 0 GOTO L3

z2 ← z2 – 1

z3 ← z3 - 1

z4 ← z4 + 1

z5 ← z5 + 1

IF z2 ≠ 0 GOTO L2

אם כן, התוכנית 'Q אחרי ההחלפות תראה כך:

P:

L1 y ← y

z2 ← x1

z3 ← y

L2 IF z2 = 0 GOTO L3

z2 ← z2 – 1

z3 ← z3 - 1

z4 ← z4 + 1

z5 ← z5 + 1

IF z2 ≠ 0 GOTO L2

L3 z1 ← z5

z1 ← z1 + 1

x2 ← x2 - 1

IF x2 ≠ 0 GOTO L1

עכשיו אפשר לבצע החלפה של פקודת הציטוט עם התוכן "המתוקן" של התוכנית 'Q:

דגשים:

* + ההוראה בריבוע הכחול מגיעה מעקרון מנגנון ההרחבה הכללי (שימור התווית של ההוראה המוחלפת ע"י הוראת סרק)
  + ההוראות בריבוע הסגול מגיעות משתילת 'Q בתוך P במקום פקודת הציטוט
  + ההוראות בריבועים הכתומים הן חלק ממנגנון הציטוט.
  + הפקודות שאינן נמצאות במסגרות צבעוניות הן למעשה הפקודות המקוריות מ P שלא שונו בשום צורה ואופן
* **Jump** **Equal** **Function**

z1 = Q(x1..)  
IF V = z1 GOTO L

מטרה:   
הפקודה מבצעת קפיצה לתווית (תסומן באמצעות L) אם משתנה מסויים (יסומן כאן כ V) שווה לתוצאת הפעלה של פונקציה (תסומן כאן כ Q). במסגרת ההפעלה יצוינו מי הם הארגומנטים המועברים אל הפונקציה   
דרך העבודה תהיה לבצע את ציטוט התוכנית Q ולקבל את ערכה למשתנה עבודה, ואז לקפוץ אל התווית במידה וערכו של המשתנה V זהה לערכו של משתנה העבודה.   
במסגרת ההרחבה יעשה שימוש במשתנה חופשי z1   
 Cycles: 6  
תצוגת משתמש: IF V = Q(x1,…) GOTO L (דוגמא: (IF X2 = ID(x2) GOTO L

שם הפקודה בקובץ: JUMP\_EQUAL\_FUNCTION  
ארגומנטים לפקודה:

* + JEFunctionLabel: התווית אליה קופצים במידה וההשוואה בוצעה כהלכה
  + functionName: שם הפונקציה להפעלה (לציטוט) - Q
  + functionArguments: מחרוזת. הארגומנטים אותם צריך להעביר ל Q

נספח ד' – מנגנון ההרכבה

מנגנון ההרכבה מאפשר לבצע הרכבה בין פונקציות, כלומר לאפשר מצב שבו תוצאתה של פונקציה אחת משמשת כקלט עבור פנוקציה אחרת.  
  
הפקודה הסינטטית "ציטוט תוכניות" משולה למעשה ל"הפעלת פונקציה". על כן אם ברצוננו לבצע "הרכבה" של פונקציות, הרי שהמטרה תהיה להגדיר את הארגומנטים הנשלחים לתוכנית המצוטטת כקריאות/הפעלות של הפוקנציה המתוארת על ידן.

מימוש מנגנון ההרכבה יבוא לידי ביטוי במסגרת הפקודה "ציטוט תוכניות", תוך תמיכה בניתוח של הארגומנטים השונים המועברים לתוכנית המצוטטת (Q)

על מנת לסמן הפעלה של פונקציה במסגרת ארגומנט ל Q, נשתמש באותו הכתיב המגדיר את ציטוט התוכנית:

**(**<Function\Program Name>,V1,V2,…,VN**)**

כל הפעלת פונקציה יכולה לקבל כמה ארגומנטים שהיא רוצה/צריכה. כל ארגומנט יופרד גם כן בפסיקים בלבד ללא רווחים.  
דוגמא:  
נניח כי S (פונקצית העוקב) ו + (פונקצית החיבור) הן שמות של פונקציות (כלומר מאחוריהן עומדות תוכניות בשפת S), ונרצה להשתמש בהם במסגרת תוכנית mul (פונקצית הכפל המבצעת כפל בין 2 משתנים), אזי הפקודה תראה כך:

z1 ← (mul,(+,x1,y),(S,(S,x2)))

כלומר קיבלנו:

z1 = (x1 + y) \* (x2 + 2)

על מנת לבצע את ההרכבה יש לנתח את הארגומנטים השונים הנשלחים לתוכנית Q ולבצע הערכה שלהם. ברגע שהמידע זמין אפשר לקרוא לתוכנית ולהפעיל אותה על הארגומנטים הנ"ל.

**תמיכה בהרכבה במסגרת מנגנון ההרחבה**:

במסגרת מנגנון ההרחבה, קיים כבר מימוש שמניח כי הארגומנטים הנשלחים ל Q הם רק שמות של משתנים אחרים (x1, y, z1 וכו').  
עכשיו ייתכן ויישלח ל Q גם ארגומנט בדמות הפעלה של פונקציה. במידה ומזהים ארגומנט שכזה – הרי שאז נתייחס אליו כאל פקודת ציטוט תוכניות עצמאית אשר מפעילה את תת-הפונקציה (+ למשל) על הארגומנטים שלה, כלומר נשתול פקודת ציטוט נוספת (עבור +) במסגרת הרחבת פקודת הציטוט הנוכחית:

בהינתן הדוגמאות של P ו Q מההסבר על הפקודה הסינטטית של ציטוט תוכניות, נשים לב כי הפעם P מעבירה ל Q את תוצאת הפעלת הפונקציה + כארגומנט ראשון.

הפיתוח וההחלפה של Q עצמה ('Q) נותר כשם שהיה:

Q’:

L2 IF z2 = 0 GOTO L3

z2 ← z2 – 1

z3 ← z3 - 1

z4 ← z4 + 1

z5 ← z5 + 1

IF z2 ≠ 0 GOTO L2

Q:

L1 IF x1 = 0 GOTO EXIT

x1 ← x1 – 1

x2 ← x2 - 1

z1 ← z1 + 1

y ← y + 1

IF x1 ≠ 0 GOTO L1

P:

L1 z1 ← (Q,(+,x1,y),y)

z1 ← z1 + 1

x2 ← x2 - 1

IF x2 ≠ 0 GOTO L1

ולמעשה השינוי הוא רק באופן שבו נפתח את P, וספציפית בערכו של z2:

P:

L1 y ← y

z2 ← (+,x1,y)

z3 ← y

L2 IF z2 = 0 GOTO L3

z2 ← z2 – 1

z3 ← z3 - 1

z4 ← z4 + 1

z5 ← z5 + 1

IF z2 ≠ 0 GOTO L2

L3 z1 ← z5

z1 ← z1 + 1

x2 ← x2 - 1

IF x2 ≠ 0 GOTO L1

במסגרת הפעלת פקודת ציטוט התוכנית המשתמשת בהרכבה יש לספק כמובן את כלל   
הפונקציות הפנימיות שבהן נעשה שימוש.

שימו לב:  
ניתן לבצע קינון של כמה פונקציות שרוצים, כולל קינון של אותה הפונקציה.  
המנגנון נועד לעבוד בצורה רקורסיבית ולמעשה רק יוסיף עוד ועוד פיתוחים והרחבות   
לתוכנית הקיימת.

נספח ה' – תיאור מבנה המערכת באמצעות קובץ XML

תיאור המע' ומרכיביה בתרגילים השונים נתון באמצעות קובץ XML.

במהלך הבדיקה (של שלל התרגילים), תיבדק המערכת באמצעות מספר קבצים שונים, חלקם חוקיים וחלקם תקולים, במטרה לראות האם וכיצד המערכת מגיבה לשגיאות.

בחנו היטיב את קבצי הדוגמא שהועלו למע' ה Mama וודאו כי אתם מבינים את פרטיהם ומבניהם.

היכן שמצויין במפורש case **in**sensitive הכוונה היא שאין חשיבות ל case של האותיות באנגלית.   
במקרה זה הערך milk זהה לערך MiLk  
בכל מקרה אחר הנחת היסוד היא כי יש חשיבות ל case של האותיות (כלומר case sensitive). במקרים אלה הערך milk שונה מהערך MiLk.

היכן שמצויין שהמחרוזת יכולה להכיל רווחים – המדובר הוא רק על רווחים בתוך המחרוזת.   
אם מופיעים רווחים בתחילתה/סופה יש להתעלם מהם (רמז: המטודה trim() על המחלקה String)

כברירת מחדל כל ה element'ים הם mandatory, אלא אם מצויין במפורש כי הם אופציונלים.

A close-up of a person's face

Description automatically generatedמבנה המע' מאוגד תחת האלמנט **S-Program**:

|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | S-Program | אלמנט זה מכיל את כלל פרטי התוכנית |
| 2 | Attribute | name | מתאר את שם התוכנית.  מחרוזת היכולה להכיל רווחים |
| 3 | Element | S-Instructions | אלמנט המכיל את סך ההוראות של התוכנית |

A computer code with text

Description automatically generated with medium confidence

האלמנט **S-Instruction** מתאר הוראה אחת בודדת

|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | S-Instruction | אלמנט ראשי המתאר הוראה אחת בודדת |
| 2 | Attribute | type | מגדיר את סוג ההוראה. האפשרויות הן basic (עבור מי מ 4 הפקודות הבסיסיות של השפה) ו synthetic עבור כל שאר סוגי הפקודות (הממתיקים המלאכותיים) |
| 3 | Attribute | name | שם הפקודה. הערכים האפשרים:   * NEUTRAL (פקודה בסיסית) * INCREASE (פקודה בסיסית) * DECREASE (פקודה בסיסית) * JUMP\_NOT\_ZERO (פקודה בסיסית) * ZERO\_VARIABLE (פקודה סינטטית) * ASSIGNMENT (פקודה סינטטית) * GOTO\_LABEL (פקודה סינטטית) * CONSTANT\_ASSIGNMENT (פקודה סינטטית) * JUMP\_ZERO (פקודה סינטטית) * JUMP\_EQUAL\_CONSTANT (פקודה סינטטית) * JUMP\_EQUAL\_VARIABLE (פקודה סינטטית) |
| 4 | Element | S-Variable | אלמנט זה מגדיר את שם המשתמש שהפקודה מתבצעת בהקשרו. (כמעט לכל פקודה יש משתנה שהיא פועלת בהקשרה). מחרוזת תווים של שמות משתנים (x,y,z עם מספרים), ללא רווחים. יכול גם להכיל את המחרוזת הריקה |
| 5 | Element | S-Label | אלמנט זה מגדיר את שם התווית המוצמדת לפקודה. אלמנט זה הוא אופציונלי ולא חייב להופיע. מחרוזת תווים של שמות תוויות (L עם מספרים), ללא רווחים. |
| 6 | Element | S-Instruction-Arguments | אלמנט המגדיר את סך הארגומנטים האפשריים המלווים את הפקודה. אלמנט זה הוא אופציונלי ואינו חייב להופיע. (לא לכל פקודה יש צורך להוסיף ארגומנטים) |
| 7 | Element | S-Instruction-Argument | אלמנט המתאר ארגומנט אחד עבור הפקודה במסגרתה הוא מוגדר |
| 8 | Attribute | name | שם הארגומנט.  עיקבו אחר נספח ג' כדי לדעת מי הם הארגומנטים האפשריים לכל פקודה |
| 9 | Attribute | value | ערך הארגומנט. מחרוזת תווים שהקשרה נקבע על פי סוג הארגומנט (תווית, משתנה, מספר וכו'..) |

## סכמה תרגיל 2

סכמה זו מוסיפה מספר יכולות חדשות**:**

* תמיכה בפקודות סינטטיות נוספות:  
  ה Attribute שנקרא name של האלמנט S-Instruction יכול להכיל בנוסף גם את האפשרויות הבאות:
  + QUOTE (פקודה סינטטית)
  + JUMP\_EQUAL\_FUNCTION (פקודה סינטטית)
* **A computer code with red and black text

  Description automatically generated**תמיכה בפונקציות (תוכניות משנה) שנעשה בהן שימוש במסגרת הוראות התוכנית הראשית:

|  |  |  |  |
| --- | --- | --- | --- |
| # | סוג | שם | מהות |
| 1 | Element | S-Functions | אלמנט המגדיר את אוסף הפונקציות השונות שיעשה בהן שימוש במסגרת התוכנית זהו אלמנט אופציונלי שאינו חייב להופיע |
| 2 | Element | S-Function | אלמנט זה מייצג תיאור של פונקציה אחת בודדת |
| 3 | Attribute | name | מתאר את השם הפורמלי של הפונקציה. שם זה צריך להיות ייחודי, ובאמצעותו ניתן יהיה לפנות ולזהות את הפונקציה שעושים בה שימוש. מחרוזת ללא רווחים |
| 4 | Attribute | user-string | מתאר את המחרוזת שתוצג למשתמש כשצריך להציג את שם הפונקציה כלפי חוץ. מחרוזת ללא רווחים |
| 5 | Element | S-Instructions | אלמנט המכיל את סך ההוראות של הפונקציה.  הוא זהה בתוכנו לאלמנט המכיל את הוראות הפונקציה הראשית |

נספח ו' – תיאור גרפי של הסכמה

סכמה 1V:

A diagram of a software system

AI-generated content may be incorrect.

סכמה 2V:

A computer screen shot of a computer

AI-generated content may be incorrect.