



פרויקט - חלק 3

הקדמה

הפרויקט זה נמשך את הקומפיילר משפט--C, שהכרנו בחלקים הקודמים של הפרויקט, לשפת המבונה Riski. נשתמש בلينקר כדי לשלב מספר קבצי שפת מוכנה לקובץ ריצה. את קובץ הריצה יוכל להריץ על המבונה הוירטואלית rx-vm.

לדוגמה, נניח שיש לנו תכנית המורכבת משני מודולים (קבצי קוד מקור): myprog.cmm, extra_funcs.cmm . הקובץ myprog.cmm הוא הקובץ הראשי המכיל את ה-main של התוכנית ואולי כמה פונקציות עזר, והקובץ השני מכיל עוד מספר פונקציות שנדרשות בקובץ הראשי.

כדי לקמפלן נקראו לקומפיילר (שאתם תבנו) עבור כל אחד מהקבצים:

```
rx-cc myprog.cmm
rx-cc extra_funcs.cmm
```

לאחר הקומpileציות הנ"ל קיבל שני קבצים בשפת המבונה Riski שיצר הקומפיילר: rsk .rx-myprog .cmm ו- rsk .rx-extra_funcs .cmm . בנוסף לכך המבונה, מידע לטובת הリンקר. נקרא לリンקר (YSISOPAK לכם) באופן הבא:

```
rx-linker myprog.rsk extra_funcs.rsk
```

הリンקר יקשר בין הקראות לפונקציות לבין מימוש הפונקציות ויפיק קובץ ריצה בשם הקובץ הראשי (הראשי) עם סימנת .e , כלומר במקרה זהה קיבל את e.myprog .

את קובץ הריצה יוכל להריץ על ידי קראיה למוכנה הוירטואלית (YSISOPAK לכם):

```
rx-vm myprog.e
```

מנשך הקומפיילר

```
rx-cc <input_file>
```

1. שם קובץ הריצה של הקומפיילר: cc-cc-rx
2. פרמטר יחיד בשורת הפקודה: שם של קובץ קלט קוד מקור יחיד. הסימנת של שם הקובץ חייבת להיות cmm . עבור קבצי מקור בשפת --C .
3. פלט: הקומפיילר מייצר קובץ באותו השם אך עם סימנת rsk . במקום cmm . המכיל את התוכנית המתורגם לשפת Riski , אם התוכנית חוקית. אם התוכנית אינה חוקית, יש להציג הודעה שגיאה (אין לייצר קובץ פלט במקרה של תוכנית לא חוקית). כמו כן, קובץ rsk . מכיל מידע נוסף בראשיתו (header) לטובת הリンקר, بماşı שיפורט בהמשך.



תיאור השפה -- C

שפת -- C מוגדרת באופן זהה להגדרות בחלוקת הקודמים של הפרויקט.

התאמות הדקדוק

1. פתרון הרוב-משמעותי בדקדוק יעשה באמצעות מגנון הגדרת קדימות/אסוציאטיביות האסימונים של Bison בלבד ואין לשנות את הדקדוק, כפי שנדרשתם בחלק 2.
2. עבור רב משמעות באופרטורים אריתמטיים/לוגיים/בוליאניים יש לישם קדימות ואסוציאטיביות במקובל ב-C/C++ . ניתן להיעזר בסיכון הקדימות בקישור הבא:
http://en.cppreference.com/w/cpp/language/operator_precedence
3. בטיפול ברוב-משמעות מסווג else יש לישם (לא שניי בדקדוק!) הצמדת ה-if ל-if הקרוב.
4. בטיפול ברוב-משמעות בהקשר של פעולה explicit cast יש להחיל את ה-cast של הטיפוס המבוקש על ה-EXP הקרוב הבסיסי ביותר (כלומר, עדיפות גובהה יותר לפעולה cast מאשר להרכבת EXP מורכב יותר מה-EXP הצמוד ל-cast).
5. השינוי היחיד המותר בדקדוק המוגדר בחלק 2 הוא שימוש ב"מרקרים" (כללי אפסיון), בדומה לנלמד בשיעור.
 למשל: { padnext = : quad.M { ↗ M }
 6. מותר שימוש (זהיר) בכללים סמנטיים פנימיים, לדוגמה:
 B: X {some_mid_action(); Y {rule_action(); } ; }
 לפרטים עיינו בתיעוד של Bison בפרק שנקרה בפרק EXP מורה ל-void.

כללים סמנטיים:

משבר לפעולות יצירת הקוד, יש כמובן לאכוף בקומpileר שלכם כלים סמנטיים. כמעט הדגשים המפורטים להלן, הסמנטיקה זהה לסמנטיקה של שפת C :

1. לא ניתן לבצע פעולות אריתמטיות או השוואתיות בין טיפוסים מסוימים. התוכנית צריכה להוביל המרות מפורשות (explicit cast) בשימוש בתחביר המתאים (כמו ב-C) על מנת שפעולות יתבצעו בין ביטויים מטיפוסים זרים. המרה מטיפוס שלם לממשי תשתמש בקוד מוכנת ו-RISKI המוגדר בחלק הבא.
2. טיפוס void משמש רק לטובת טיפוס החזרה מפונקציה. לא ניתן להציגו על משתנה או פרמטר לפונקציה מסווג void. לא ניתן לבצע פעולה בלשון בין שניים (EXP) מטיפוס void. EXP שנוצר מ-STMT חייב להיות מטיפוס void (במידת הצורך, יש להמיר באמצעות cast ל-void).
3. פרמטרים שמיים בקריאות לפונקציות:
 - א. ניתן לקרוא לפונקציה עם פרמטרים בעלי שמות כפי שניתן לעשוות ב-`Python`. פרמטר שמי כתוב בתחביר `label:value`.
 - ב. פרמטרים מיקומיים (לא שם) חייבים להופיע לפני פרמטרים שמיים. הופעה של פרמטר מיקומי אחר פרמטר שמי תגרור שגיאה תחבירית.
 - ג. לא ניתן להעביר שמי פרמטרים מיקומיים מאשר מוגדרים בפונקציה.
 - ד. שם הפרמטר בקריאה שמי חייב להתאים לשם פרמטר מוגדר בהצהרת הפונקציה. שם שאינו קיים יגרור שגיאה סמנטית.
 - ה. לא ניתן להעביר ערך לאוטו פרמטר פעמיים - לא באמצעות פרמטר מיקומי ואחר בר שם, ולא באמצעות שני פרמטרים שמיים זרים. בפלות זו תגרור שגיאה סמנטית.
 - ו. כל פרמטר מוגדר בפונקציה חייב לקבל ערך בקריאה - או באמצעות פרמטר מיקומי או באמצעות פרמטר שמי. פרמטר לא מוגדר יגרור שגיאה סמנטית.



4. בקירות לפונקציה הפורמטרים המועברים צריכים להיות מאותו טיפוס של הפורמטרים המוגדרים.
או התאמת בטיפוסים להערכת ממשק הפונקציה גוררת שגיאה סמנטית. בהתאם, התוכנית
חייבת להכיל המרות מפורשות של ביטוי לטיפוס המתאים לפורמטר אם הם שונים, בn"ל.

5. יתכן ובסוף פונקציה לא תהיה פקודת `return` בקוד המקור – הן מכיוון שמדובר בפונקציה
המחזירה `void` והן בגלל שגיאת תוכניתן. אין צורך להפיק שגיאה במקרה זה, אולם בכל מקרה יש
להבטיח חזרה מהפונקציה לקורא, גם אם חסורה פקודת `return` בסוף הפונקציה. ערך ההחזרה
במקרה זה (אם אינם מוגדר `void`) לא מוגדר (יכול להיות כל ערך).

6. יש לנחל טבלאות סמלים עם תיחום סטטי (static scoping), כלומר, משתנה מוגדר בתחום של ה-
`BLK` שבו הוגדר.

- תחילת פונקציה מתחילה טבלת סמלים ריקה.
- הפורמטרים של פונקציה שייכים לתחום (scope) של הבלוק הראשי של הפונקציה.
- הגדרת משתנה ב-`BLK` פנימי ממסכת משתנה בעל שם זהה שהוגדר בבלוק שמכיל אותו.
- אסור, כמובן, להגיד יותר ממשתנה אחד בעל אותו שם באותו הבלוק (זו שגיאת סמנטית).
- אין משתנים גלובליים.



שפת המטרה: שפת ה – Riski

Riski היא שפה דמוית ASM. הפורמט המחייב של התוכנית הוא:

1. הוראה אחת בכל שורה.
2. ההוראות עצמן תמיד באובייקט גודלות .
3. קוד ההוראה והרגומנטים מופרדים ע"י טו רוח אחד לפחות .
4. בכל תוכנית, הוראות HALT מופיעות בשורה الأخيرة אותה יש לבצע.
5. יש לציין בפקודות קפיצה (BREQZ, BNEQZ, JLINK, UJUMP) את מספר הפקודה אליה קופצים (לא בתובת). מספרה של הפקודה הראשונה הוא **1**.

שפת ה Riski תומכת בפקודות הבאות:

Opcode	Arg	Description
COPYF/COPYI	A B	A=B
SEQUF/SEQUI	A B C	If B=C then A=1 else A=0
SNEQF/SNEQI	A B C	If B!=C then A=1 else A=0
SLETF/SLETI	A B C	If B<C then A=1 else A=0
SGRTF/SGRTI	A B C	If B>C then A=1 else A=0
ADD2F/ADD2I	A B C	A=B+C
SUBTF/SUBTI	A B C	A=B-C
MULTF/MULTI	A B C	A=B*C
DIVDF/DIVDI	A B C	A=B/C
LOADF/LOADI	A B C	A = Mem[B+C]
STORF/STORI	A B C	Mem[B+C] = A

- שימושם לביצוע פעולות סימוט או מתבצעות על寄REGISTERים שלמים (int) ופעולות בעלות סימות F
- מתבצעות על寄REGISTERים ממשיים (float).

Opcode	Arg	Description
CITOF	A B	A = (float)B
CFTOI	A B	A = (int) B

Opcode	Arg	Description
UJUMP	L	goto L
JLINK	L	I0=address of next instruction goto L
RETRN		goto I0
BREQZ	B L	If(B=0) goto L
BNEQZ	B L	If(B!=0) goto L

Opcode	Arg	Description
PRNTC	C	Prints a character of the ASCII value of C
PRNTI	B	Prints the integer value of B
READI	A	Read an integer into A

Opcode	Arg	Description
PRNTF	B	Prints the float value of
READF	A	Read a float into A

Opcode	Arg	Description
HALT		Stop



הערות לטבלה הנ"ל :

1. C,A,B,C סימנים מופשיים , שיכולים לציין רגיסטר או קבוע בדשא:
 - A לציין רגיסטר שלם בן 32 ביט.
 - C,M,C,M,R גרגיסטרים או קבועים.
2. כל הפקודות פרט ל-LOAD ו-NOR פועלות על רגיסטרים בלבד.
3. C מציין תווית (מספר שורה)*.

*ראו הסבר לגבי מספור השורות בפרק "מרחיב זיכרון הפקודות וחישוב יעדי הקפיצה" בהמשך.

משאבי זיכרון ורגיסטרים

1. בתובת זיכרון נתונים (הערך של C+B Mem בפרט הפקודות) הינה מספר שלם בין 0 ל-1023. הזיכרון הוא ממוקן בתים, כלומר בתובת X היא הבית בהיסט X מהתחלת הזיכרון. גודל הנתון שנקבע הוא 32 ביט בין אם לקרוא מספר שלם או ממשי.
* זיכרון הפקודות נפרד מזיכרון הנתונים ומוחבי הכתובות בלתי תלויים. ראו פרטים בפרק "מרחיב זיכרון הפקודות וחישוב יעדי הקפיצה" בהמשך.
* יש זיכרון נפרד לריגיסטרים ממשיים (F) ולרגיסטרים שלמים (I).
2. המעבד מכיל 1024 רגיסטרים מסווג שלם ו-1024 מסווג ממשי בני 32 ביט כל אחד. הריגיסטרים בעלי המספרים הסידוריים 0-1023. נסמנם באות א' (אי') ומספר סידורי החל מאפס, למשל: 121 מצין את הריגיסטר השלישי-עשרה בקובץ הריגיסטרים של המעבד. בדומה עבור רגיסטרים ממשיים נסמנם באות F (אף).
שים-ולב: הריגיסטרים צריכים לשרת רק את המשתנים הזמןניים שמקצת הקומפיילר. אין להשתמש בהם (באופן קבוע) לטובת משתנים שימושיים בתוכנית המקומפלט.
3. הריגיסטר השלם הראשון ו-0 שומר בארכיטקטורה לטובת שמירת בתובת החזרה בפקודות +LINK .RETRN .
4. יש לשמר (להקצאת) רגיסטרים נוספים, לבחירתם, על מנת לנצל את המחסנית ואת רשומות הפעולה. מומלץ להקצאות את הריגיסטרים השמורים ברגיסטרים הנמוכים ואת ההקצאות המשתנות למשתנים הזמןניים לבצע מעל התחומים השמורים, בהתאם לתוכנית המקומפלט.
5. אין צורך לטפל בהקצאה עיליה של רגיסטרים. ניתן להניח שבמהלך פונקציה אחת לא יהיה צורך ביותר מ-900 משתניםזמןניים עבור הקוד של אותה הפונקציה, אולם שימו-לב שפונקציה יכולה לקרוא לעצמה או לכל פונקציה אחרת (כלומר, הקוד הנוצר עבור כל פונקציה Zukunft כמעט ככל מעט לכל מרחב הריגיסטרים).
6. הניחו כי יש מספיק מקום בזכרון המחסנית לצורך אחסון רשותות הפעלה ושמירת רגיסטרים במהלך שרשרת קריאה לפונקציות. אין צורך לבדוק גלישה של המחסנית מעבר לказחה מוגבל הזיכרון. גישה ל זיכרון מעבר למוגדר תגרום לאיורע "חריגה" ועצירה של מכונת Riski. Riski. שימו לב כי לSIMULATOR יש מגבלת הריצה של 5000 שורות אסמל, אך הלינקר יוציא שגיאה אם יוצר קובץ הריצה מעל למספר שורות זה. כמו כן, אין הכרח שתוכנית תעיגע לכמות כזו של שורות, זה בכלל הנראה מחייב על שגיאה במימוש שלכם.

מרחיב זיכרון הפקודות וחישוב יעדי הקפיצה

זכרון הפקודות במכונת Riski נפרד מזיכרון הנתונים וממוקן באופן שונה ממנו. בתובת של פקודה היא מספר השורה (מספר סידורי) של פקודה בקוד המכונה המיציר, החל מ-1 עבור הפקודה הראשונה בזיכרון. קובץ הריצה (e.) נתען לתחילה זיכרון הפקודות (בתובת 1) וריצת מוכנת Riski מתחילה מפקודה (שורה) מס' 1 (כלומר, הפקודה הראשונה בקובץ הריצה היא הפקודה הראשונה שתורוץ ב-M7 של ה-Riski).

בקוד ה - Riski שמייציר הקומפיילר עשויה להופיע פקודות קפיצה, כאשר יעד הקפיצה הוא מספר שורת פקודה יעד בנו"ל. לצורך חישוב יעד הקפיצה בזמן קומפיילציה, יש להשתמש בהטלאה לאחר מכן (backpatching) ובמרקטים מסווג $e \rightarrow M$. בתחילת זה הקומפיילר מניח שקובץ המקור שהוא



מקמפל הוא היחיד בקובץ הריצה, כלומר שהפקודה הראשונה בקובץ שהוא מייצרת תהיה בכתבوبة (שורה 1) בזיכרון הפקודות.

במידה וקובץ `ask` שמייצר הקומפיילר משלב עם עוד קבצי `ask` לתוכנת קובץ הריצה (e.g., הלינקר יציג לתקן את בתוכנות היעד הרלוונטיות בהתאם למיקום היחסיו של כל קובץ `ask` בקובץ הריצה שנוצר. ראה פרטים על תהליך זה בפרק "תמייה ב-linker" בהמשך.

מימוש write בשפת --C

פקודת `write` בשפת --C משמשת להדפסת ערכים לפטל הסטנדרטי. יש למשמש אותה באמצעות הפקודות `PRNTI/PRNTF/PRNTC`.

פקודת `PRNTI` מניחה שהערך שנייתן לה בארגומנט הוא בן 32 ביט. בהתאם, יש להמיר את ה-EXP שבארגומנט של פקודת `write` לגודל של 32 ביט על מנת שיודפס ערך נכון במקרה שהוא של ערך מטיפוס שלם קטן יותר.

במקרה של הדפסת מחוזות, יש להשתמש במספר קריאות מתאימים לפקודה `PRNTC`. במו-כך, יש לטפל בהמרה של היצירופים המיוחדים:

- \n (שורה חדשה בלינוקס).
- \r (גרשיים באמצעות מחוזות)
- \t (טאב).

ההמרה היא לערך הוו ASCII המתאים. למשל, רצף של שני תוים כאלו במחוזות יש להמיר לתוכן המתאים בפלט ולא להדפיס לבסן אותן – בדומה לשימוש ביצירופים הללו ב-`printf` בשפת C.

המרת טיפוסים

כפי שצוין לעיל, הרегистרים בארכיטקטורת Riski הם שלמים או ממשיים בני 32 ביט.

יש צורך בהמרה כאשר מצבים מרגיסטר שלם לתוך רегистר ממשי (ולהיפך). לדוגמה, הפקודה הבאה תחזיר שגיאת זמן ריצה ע"י הסימולטור RX:

`COPYF F1 I1`

במקום זאת יש להשתמש בהמרה מפורשת בעזרת פקודת `RISKI` הבאה:

`CITOF F2 I1`

`COPYF F1 F2`

יש לשים לב שבדירון משתמשים ממשיים ושלמים שניהם נשמרים ב32 ביט. לכן קריאה ובתייה מהדיברונו של משתנה שלם או ממשי בעזרת פקודת `LOAD` או `STORE` אשר איננו מתאימה לו תבצע פעולה על ערך שאינו מוגדר.



הצהרות, הגדרות וקריאה לפונקציות

השפה מאפשרת קריאה לפונקציות המצוירות/¹ או מוגדרות (ممומשות) לפני מקום הקראיה בקוד (בכל קראיה של פונקציה לעצמה, ללא הצהרה מקדימה). הגדרת הפונקציה (המיימוש שלה) יכולה להיות באוטו קובץ מקור (לפני השימוש, או אחריו השימוש אם לפני השימוש הייתה הצהרה מתאימה) או בקובץ מקור אחר. הפונקציות מקבלות מספר כלשהו של פרמטרים (בהתאם להגדלה) ע"פ ערכם (by value) ומחרירות ערך החזרה ייחיד מהטיפוס שהוזכר בעבר הפונקציה.

lezicircumflex; להזכירם, הדקוק מכיל הצהרות על פונקציות מהצורה:

FDEFS → FDEFS FUNC_DEF_API BLK | FDEFS FUNC_DEC_API ε | ε

FUNC_DEC_API → TYPE id () ; |
TYPE id (FUNC_ARGLIST) ;

FUNC_DEF_API → TYPE id () |
TYPE id (FUNC_ARGLIST)

במקרה שנעשתה קריאה לפונקציה שאינה מוצחרת או מוגדרת בקובץ עד לשורת הקוד הקורא, על הקומpileר להבריז על שגיאת קומפילציה (סמנטית). גם קריאה לפונקציה עם פרמטרים מסוג לא מתאים להגדרת הממשק שלה מהוות שגיאת קומפליציה (סמנטית). שימוש-לב כי פונקציה יכולה לקרוא לעצמה, בולם, פונקציה מוכרת מרגע שהוזכרה על הממשק שלה, גם אם לא הסתיים הבלוק שמוביל את הקוד המימוש שלה.

מומלץ להשתמש בטבלת סמלים מתאימה על מנת לשמור את בתובת ההתחלה של הפונקציה (מספר שורה בתוכנית) ונתונים נוספים המדרשים לשימוש בפונקציה בקוד ואימונות שימוש נכון בפרמטרים (בדומה לטבלת סמלים עבור משתנים). על מנת לאפשר קריאה לפונקציה שרק הוצאה אבל טרם מונשה, יש לשמר את המיקומות בהם יש קריאה לפונקציה מוצחרת אך לא מוגדרת. במידה והפונקציה תוגדר בהמשך הקובץ, באחריות הקומpileר להטיל את המיקומות בהם הייתה קריאה לאותה פונקציה בקוד המקורי. אם פונקציה לא תוגדר עד לסיום אותו קובץ מקור, יש לכלול את רשימת מקומות הקריאה לפונקציה בתחלת קובץ ה-*skz* בהתאם להגדרות בפרק "תמייה ב-Linker" בהמשך (הרשייה תשמש את הלינקר להטלת קריאות בין מודולים).

אין תמייה ב-*Function overloading*, כלומר, לכל פונקציה קיימת רק הגדרה ייחודית וממשק (הגדרת פרמטרים וערך החזרה) ייחדים. הגדרת פונקציה שונה מהצורה עלייה (אם קיימת) או קיומם הצהרות לאותה פונקציה עם סוג פרמטרים/ערך היחה שונים או ריבוי הגדרות של פונקציה הן שגיאות סמנטיות.

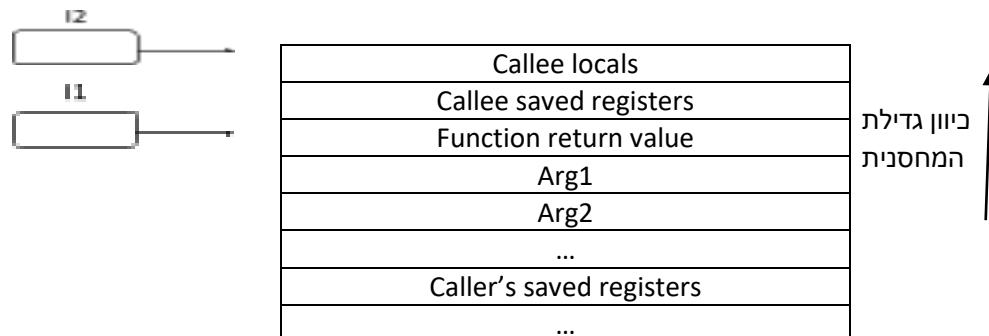
באחריות הקומpileר שלכם לאכו את הדרישה בטור קובץ מקור. הלינקר יdag לאכיפת מגבלה זו בין מודולים (שישנה הגדרה אחת לפונקציה בכל המודולים). הלינקר לא יאמת שה-API שהוזכר בעבר פונקציה במודול אחד תואם ל-API שהוזכר בעבר פונקציה באותו שם במודול אחר.

ניתן לקרוא לפונקציה מספר פעמים במהלך התוכנית, כולל קריאות רקורסיביות, ישירות או עקיפות. בהתאם, השתמשו ברשותה הפעלה בדומה לנלמד בשיעור ורגיסטרים שמורים על מנת לנחל אותה. ניתן לתוכנן את רשותה הפעלה כרצונכם בתנאי שהיא עומדת בדרישות הב"ל.
קריאה לפונקציה יכולה להיעשות על ידי שימוש בפרמטרים מיקומיים, פרמטרים שמיים או שילוב שלהם. שימוש-לב כי אתם עוקבים אחר הכללים הסמנטיים שניתנו.

שימוש-לב: בעת קריאה לפונקציה, כל גציגטר יכול להשתנות. לכן עליו לתהוכן היטב למעבר לפונקציה.



להלן המלצה לאפשרות אחת לניהול רשומות הפעלה, אולם היא אינה מחייבת:
 רגיסטר 10 יכיל את בתובת החזרה מהפונקציה (מחויב מהגדירה ארכיטקטונית של פקודת LINK). רגיסטר 11 יכיל את בתובת בסיס רשומות הפעלה (stack frame) ו- 12 יכיל את המצביע לראש המחסנית. לדוגמה:





התמודדות עם שגיאות קומpileציה והודעת שגיאה

אין הנחת קלט תקין לקומPILEר, בלומר יש להתמודד עם שגיאות. במקרה של גילוי שגיאה בזמן הקומPILEציה, יש לעצור את הקומPILEציה, להוציא לפلت השגיאות הסטנדרטי (stderr/cerr) הודעת שגיאה, ולצאת מהקומPILEר עם החזרת קוד יציאה/שגיאה כמפורט להלן:

1. עברו שגיאה לקסיקלית, קוד שגיאה 1 והודעה במבנה הבא:

`Lexical error: '<lexeme>' in line number <line_number>`

2. עברו שגיאה תחבירית, קוד שגיאה 2 והודעה במבנה הבא:

`Syntax error: '<lexeme>' in line number <line_number>`

כאשר `<lexeme>` הינה הלקסמה הנוכחית בעת השגיאה.

3. עברו שגיאה סמנטית, קוד שגיאה 3 והודעה במבנה הבא:

`Semantic error: <error description> in line number <line_number>`

כאשר `<error description>` הינה הודעת תיאור השגיאה ברכונכם, ו-`<line_number>` הוא מספר שורת השגיאה בקוד המקור.

4. עברו שגיאות אחרות בזמן ריצת הקומPILEר (למשל, שם קובץ קלט לא קיים, כישלון בהקצת ציברוני וכו'), קוד שגיאה 4 והודעה במבנה הבא:

`Operational error: <error description>`

הערה חשובה: אחרי הדפסת כל שגיאה יש לדוח שורה. בנוסף, אין לדוח שורה לפני הדפסת השגיאה.



תמייה ב-Linker

כמו בשפת C, נרצה לאפשר הפרדה של הקוד למספר קבצים ולקמפל כל אחד מהם בנפרד. ה-linker מאפשר לחבר את הקבצים הללו לקובץ ריצה (executable) בוודאי אשר יורץ במחשב המקורי. בתרגיל זה נשתמש ב-linker המספק لكم (ראו הסבר בסוף התרגיל) אשר יוצר קובץ ריצה בוודאי עבור מכונת ה-RX ונאפשר שימוש בפונקציות המוגדרות בקבצים שונים.

לצורך תמייה ב-linker על הקומpileר להוסף header בעל ארבע שורות בדיק (כמפורט בהמשך) שיביל מספר פרטיים נחוצים לצורכי חיבור הקבצים.

ה-linker ישתמש במידע זה כדי לקשר בין הקריאות לפונקציה לבין מיקום הפונקציה בקובץ הריצה המקורי. בנוסף, ה-linker יתקן את כל הקפיצות האבסולוטיות שנעשו בקוד ויוסיף את היחס החדש לפ' המיקום החדש של הקוד בקובץ המקורי.

שימוש לב:

הlienker "מתќן" בתובות קפיצה קיימות, ולא מוסיף בתובות בקוד, כמו בתהller הטלאה. לכן, בכל פקודה קפיצה לפונקציה חייבתobar להיות בתובת קפיצה. במקרה של קפיצה לפונקציה באותו הקובץ, זו תהיה השורה של יעד הקפיצה אליו זה היה הקובץ היחיד (השורה שבמהלך פונקציית היעד באותו קובץ). במקרה שהקפיצה לפונקציה המומשת בקובץ אחר, ניתן לרשום כל בתובת יעד שלינker יוכל להחליפ'(רצוי) שזו תהיה בתובת לא חוקית, על מנת להוות תקלות בתהller שמבצע הלינker).

פונקציית `main` וספריית זמן ריצה

מכונת RX מתחילה את הריצה מכתובת 1 בזיכרון הפקודות. הלינker שם בתחילת קובץ הריצה (החל מכתובת 1) את ספריית זמן הריצה `rsk`. ספריית זמן הריצה מתחילה מספר גיטרים לאפס וקוראת לפונקציית `main`. פונקציית `main` צורכה להיות מוגדרת באחד מקבצי `rsk` המספקים לlienker בשורת הפקודה וחיבת להיות מהטיפוס הבא:

```
; void main()
```

העדר הגדרה של פונקציית `main` באחד מהמודולים או הגדרה של יותר מפונקציית `main` אחת יגרמו לשגיאתlienker. הגדרה של פונקציית `main` בעלת טיפוס ממתק שונה מהנ"ל לא תגרום לשגיאתlienker, אולם ההתנהגות בזמן ריצה תהיה בלתי מוגדרת. החזרה מפונקציית `main` תביא לסיום הריצה (חיב להיות RETRN בסיום `main`, כמו בכל פונקציה אחרת).

ספריית זמן הריצה `rsk`-`runtime` מסופקת לכם עם חומרי הפרויקט. מכיוון שאין העברת פרמטרים וערך חוזר מה-`main`, האופן שבו נקראת `main` מתחול זמן הריצה לא מחייב דבר לגבי מבנה ורשומת הפעלה שלכם ואופן העברת פרמטרים לפונקציות, ולכן ניתן להשתמש בקובץ המספק כפי שהוא ללא שינויים ללא קשר למבנה ורשומת הפעלה שתבחרו.



מבנה header

- 1) <header>
- 2) <unimplemented> func₁,L₁,L₂,L₃,... func₂,L₁,... ...
- 3) <implemented> func₁,L func₂,... L ...
- 4) </header>

באשר func הוא שם פונקציה ו-L היא השורה בה הפקציה ממומשת או נקראת.
שורה ראשונה: בותרת של תחילת ה-.header.

שורה שנייה: ציון כל הפקציות הלא ממומשות בקובץ <unimplemented> - הרשומות מופרדות ע"י רווח.
כל רשותה בשורה מורכבת משלושה חלקים

- (1) שם הפקציה
- (2) מיקומי כל הקראות לפונקציה בקובץ המקומפל (מספריו שורות) מופרדים ע"י פסיק.
והרכבים הנ"ל מופרדים ע"י פסיק.

שורה שלישיית: ציון כל הפקציות ממומשות בקובץ <implemented> - הרשומות מופרדות ע"י רווח.

- (1) שם הפקציה
- (2) המיקום הייחסי להגדרת הפקציה בקובץ המקומפל (מספריו שורה) .
והרכבים הנ"ל מופרדים ע"י פסיק.

שורה רביעית: סגירת ה-.header.

כל המיקומים של פונקציות או קראות להן יתיחסו למספריו השורות בקוד שפת האסמבלי הנוצר אשר מתחילה במספר 1, החל מהשורה שלאחר ה-</header>.

לדוגמה:

- 1) <header>
- 2) <unimplemented> foo,2,10,30 goo,3,11,21 boo
- 3) <implemented> woo,2 voo,9 main,24
- 4) </header>

זה header של קובץ שבו הפקציה foo (חיצונית) נקראת בשורה 2,10 ו-goo (חיצונית) נקראת בשורה 3,11 והפקציה boo מוצהרת אבל לא נקראת ממשום מקום. הפקציות woo ו-voo מומשו בשורות 2 ו-9 בהתאם. כמו כן, הקובץ מכיל את פונקציית ה-main בדרכה 24.

גם כן, הרשומות של הפקציות ב- header מופרדים על ידי רווח.

דוגמאות

- ראו דוגמאות לתוכניות בשפת -- C והפלט המתאים של הקומpileר באתר הקורס, תחת קבצי עזר לחלק זה שלuproject. קחו בחשבון כי הדוגמאות מתאימות לשימוש הפעלה והקצאות הריגיסטרים שנבחרו באותו מימוש וייתכן שבמימוש שלכם יהיה הבדל בפלט, גם אם הוא נכון ומתאים לדרישותuproject. לכן, הבדיקה של המימוש שלכם בפועל צריכה להיות מבוססת על הריצת קובץ הריצה (.e.) על גבי ה-WV של ה-XR והשוואת הפלט לפלט המוצופה.
- חשוב לעקוב אחרי לוח המודעות באתר הקורס.



כלים נוספים

- יש להשתמש בכלים שנלמדו בקורס (Flex, Bison) לצורך כתיבת הפרויקט – במיוחד לצורך מימוש המנתח הלקסיקלי והמנתח התחריבי, כפי שנעשה בחלק הפרויקט הקודמים.
- יש לבתוב את הפרויקט בשפת C או ++ C בלבד.

ניתן להיעזר במבני נתונים סטנדרטיים (רשימות, טבלאות "האש" וכו') המספקים על ידי ספריות חיצונית סטנדרטיות הzmיניות במבנה הווירטואלית של לינוקס המספקת לטובות הפרויקט (למשל, C++ STL). במידה ותשתמשו ב-`STL` C++ ניתן להיעזר בפריטים בקשרו הבא לגבי מבני נתונים שימושיים:

<http://en.cppreference.com/w/cpp/container>

<http://www.yolinux.com/TUTORIALS/CppStlMultiMap.html>

ניתן גם להשתמש במימושים "פתוחים" של מבני נתונים כנ"ל, בתנאי שהשימוש בהם דורש הוספה של קבועים בודדים של המימוש לחומר ההגשה. אין להשתמש בספריות שאינן מותקנות במבנה הווירטואלית של לינוקס המספקת. במקרה של שימוש חיצוני של מבנה נתונים כלשהו, יש לציין בתיעוד המצורף להגשה את המקור לאותו מימוש שיצרפתם.

הLINCKER: rx-linker

באוצר הקורס, תחת קבצי עוזר לחלק זה של הפרויקט, ניתן למצוא את הLINCKER בשם `rx-linker`.

LINCKER יש לחת בשורת הפקודה את רשותת קבץ `rsk` (תוכרי הקומפיילר) שמעוניינים לצרף לטובת קובץ הריצה. אין צורך לחת את קובץ ספריית זמן הריצה `rsk-runtime.rsk` – הLINKER יצרף אותו באופן אוטומטי, כדי שהוסבר בסעיף "פונקציית `main` וספריית זמן ריצה" – אולם, יש להקפיד שקובץ זה יהיה בתיקייה הנ惋יה בעת הפעלת LINKER.

גם אם יש רק קובץ `rsk` אחד בתוכנית, יש צורך להפעיל עליו את LINKER לשם יצירת קובץ ריצה הכליל את ספריית זמן הריצה. כאשר מפעלים את LINKER עם יותר מקובץ אחד, שם קובץ הריצה שיופיע הוא שם הקובץ הראשון בשורת הפקודה עם סימנת `e` במקום `rsk`.

המבנה הווירטואלית zw-ax

באוצר הקורס ניתן למצוא את המבנה הווירטואלית zw-ax, שהוא בעצם סוג של מפרש לשפת Riski. בעזרתה אתם יכולים "להריץ" את קובץ הריצה `e`. שיצר LINKER. zw-ax מצפה לארגומנט בווד בשורת הפקודה – שם של קובץ ה-`e`; כאשר RX נתקל בפקודה `READY`, הוא מדפיס סימן "?" על המסך ומচכה לקלט. ניתן גם להעביר אליו קלט מקובץ בעדרת שימוש ב-`pipe` redirection לקלט הסטנדרטי, במקובל בזינוקס.

יודגש כי יתכן zw-ax יקבל וירץ קוד החורג מן ההנחיות הנ"ל. ככל מקרה עלייכם לייצר קוד העומד בכל ההנחיות והדרישות המפורחות במסמך זה.

חומר עזר

באוצר הקורס ניתן למצוא שני קבועים עוזר בשם: `part3_helpers.cpp` ו-`part3_helpers.h`, בהם תמצאו מימושים והגדות שיכולים לסייע לכם. חשוב להבהיר שאין חובה להשתמש בקבצים האלה.



הוראות הגשה

- מועד אחרון להגשתה: יום ו' 23/01/2026 בשעה 23:55 .
 - שימוש-לב למדיניות בונגש לאחוריים בהגשתה המפורסמת באתר הקורס. במקרה של נסיבות מצדיקות אי-חוור, יש לפנות מראש לצוות הקורס ל蒂יאום דחית מועד ההגשתה.
 - ההגשתה בזוגות. הגשה בבודדים תתקבל רק באישור מראש מצוות הקורס.
 - יש להגיש בצורה מקוונת באמצעות אתר Moodle של הקורס, מחשבונו של אחד הסטודנטים. הקפידו לוודא כי העלתם את הגרסה של ההגשתה אותה התכוונתם להגיש. לא יתקבלו טענות על אי-התאמה בין הקובץ שנמצא ב-Moodle לבין הגרסה ש"התכוונתם" להגיש ולא יתקבלו הגשות מאוחרות במקרים אלו.
 - יש להגיש קובץ ארכיב מסוג TAR-Bzipped2-TAR בשם מהצורה (שרשור מספרי ת.ז – 9 ספרות): `<student1_id>-<student2_id>.tar.bz2`
 - בארכיב יש לכלול את הקבצים הבאים:
 - את כל קבצי הקוד בהם השתמשתם (`Bison`, `Flex`, `headers`) וכל קובץ קוד מקור הנדרש לבניית המנתח).
 - * יש להקפיד שהקובץ שלכם יהיה קרייא ומתווד פנימית ברמה סבירה כך שגם "זר" יוכל להבין את השימוש שלכם!
 - יש לכלול גם קבצי מימוש מבני נתונים ממוקור חיצוני שנעזרתם בהם, במידה ואינם חלק מהספריות הסטנדרטית של C/C++ המותקנות במכונת לינוקס המספקת. במקרה זה יש לתעד את המקור של אותם קבצים במסמך התיעוד החיצוני. אין לדרש התקינה של שום קובץ או ספרייה נוספת, מעבר למאה שככל בהגשתה שלכם ולמה שמוטקן במכונה הוירטואלית, על מנת לבנות ולהריץ את הקומפיילר שלכם.
 - הובנה את קובץ הריצה של הקומפיילר `makefile`
 - * תזכורת: שם קובץ הריצה של הקומפיילר יהיה `cc-xx`
 - קובץ הארכיב צריך להיות "שטווח" (כלומר, שלא ייצור ספריות משנה בעת הפעילה אלא הקבצים ייווצרו בספריה הנוכחית).
 - כמו בחלקים הקודמים, סבבetta הבדיקה המכונה המכונה הוירטואלית של לינוקס המספקת לכם. הקפידו לבדוק את ההגשתה שלכם במכונה זו ועל תשאירו זאת לרגע האחרון.
- תרגיל שלא הצליח להתקמלן קיבל 0.

בצלחה!