

הטכניון – מכון טכנולוגי לישראל

ארגון ותכנות המחשב

תרגיל 1 - חלק רטוב

המתרגל האחראי על התרגיל: עידו סופר

שאלות על התרגיל – ב- Piazza בלבד.

הוראות הגשה:

- ההגשה בזוגות.
- על כל יום איחור או חלק ממנו, שאינו באישור מראש, יורדו 5 נקודות.
 - ניתן לאחר ב-3 ימים לכל היותר.
- הגשות באיחור יתבצעו דרך אתר הקורס.
- יש להגיש את התרגיל בקובץ zip לפי ההוראות בעמוד הבא.
- אי עמידה בהוראות אלו תעלה לכם בנקודות יקרות.

הוראות והנחות לפתרון והגשת התרגיל

הוראות כלליות

- התרגיל וכל ההוראות כתובים בלשון זכר/נקבה באופן אקראי אך פונים לגברים ונשים באופן שווה.
- ישנם 5 סעיפים, הפתרון של כל אחד צריך להופיע בקובץ נפרד ex1.asm, ex2.asm וכו' המצורפים לכם להשלמה. ההגשה שלכם צריכה להיות קובץ zip אחד שמכיל את חמשת הקבצים שקיבלתם, בלי שהורדתם מהם כלום וכאשר הוספתם להם רק פקודות אסמבלי ולא שום דבר אחר.

טסטים

- מסופק לכם קובץ בדיקה לתרגיל 3 והוראות כיצד להשתמש בו. הטסט מכסה מקרה בסיסי ומסופק לכם כדי שתדעו כיצד אנו מצפים לקבל את הפתרון (בכל אחד מחמשת התרגילים) וכיצד אנו מתכוונים לבדוק אותם.
- מומלץ ביותר לבצע טסטים נוספים בעצמכם לכל חמשת התרגילים, על בסיס מבנה הטסט הזה.
 - נדגיש כי לא מסופקים טסטים מהסגל עבור תרגילים 1,2,4,5.

כללים לכתיבת קוד

- בקוד שאתם מגישים אין לשנות שום ערך במקטעי (sections) הזיכרון, מלבד אלו שנאמר לכם מפורשות.
- עליכם לבצע חישובים ברגיסטרים בלבד (ככל שניתן). ביצוע חישובים על ערכים בזכרון הוא לא יעיל.
- בכל סעיף הניחו כי בתוויות הפלט הוקצה מספיק מקום.
- על כל תווית שתשמשו בה לצורך מימוש להסתיים בארבעת התווים "_HW1". תחילת התווית היא לבחירתכם וכדאי כמובן שיהיה קשר למטרת התווית, לתיעוד וקריאות של הקוד. לדוגמא –
loop: #this is a bad label name
loop_HW1: #this is a good label name

אי עמידה בכל אחת מההוראות הנ"ל עלול לגרור הורדת ניקוד לתרגיל ולהשפיע על זכותכם לערעור במקרים מסוימים.

אופן בדיקת התרגיל וכתובת ססמים בעצמכם

כאמור, כל חמשת התרגילים יבדקו בצורה דומה. לכן, עקבו אחר ההוראות שיתוארו להלן, כאשר תרצו לבדוק את הקוד שלכם לפני ההגשה. חבל שתגישו קבצים שלא לפי הפורמט ותצטרכו להתעסק עם ערעורים ולאבד נקודות סתם.

בכל תרגיל, תקבלו קובץ asm המכיל text.section. בלבד. עליכם להשלים את הקוד שם, אך לא להוסיף sections נוספים לקובץ בעת ההגשה (ההגשה חייבת להכיל section text בלבד. בפרט, **לא להכיל את section data**).

אז איך בכל זאת תוכלו לבדוק את התרגיל שלכם? זה פשוט. לתרגיל 3 מצורף טסט בודד בתיקייה tests. הבדיקה היא בעזרת הקובץ run_test.sh.

שימו לב שכל הטסטים על קבצי הקוד שלכם ירוצו עם timeout (כפי שגם מופיע ב-run_test.sh) ולכן כתבו אותם ביעילות. קוד שלא ייכתב ביעילות ולא יסיים את ריצתו על טסט מסוים עד ה-timeout, ייחשב כקוד שלא עמד בדרישות. בפרט הקוד ייבדק עם מגבלת הזמן שבטסט שקיבלתם, כך תראו שאין צורך לאלגוריתמים פורצי דרך לעמידה בזמנים.

הריצו את הקובץ run_test.sh באופן הבא:

```
./run_test.sh <path to asm file> <path to test file>
```

לדוגמה, עבור התרגיל השלישי והטסט שלו ומתוך התיקייה שמכילה את קבצי הקוד של הסעיפים ואת תיקיית הטסטים:
./run_test.sh ex3.asm ex3sample_test

הערה:

ייתכן ולפני הרצת קבצי sh על המכונה, תצטרכו להריץ את הפקודה –

```
chmod +x <your .sh file>
```

כתיבת ססמים בעצמכם

לכל תרגיל תוכלו לכתוב טסט, שהמבנה שלו דומה למבנה של ex3sample_test, עם שינוי תוויות ובדיקות בהתאם.

תרגיל 1 (12 נק')

עליכם לממש את ex1 המוגדרת בקובץ ex1.asm.

בתרגיל זה תקבלו תווית num עם מספר (עם סימן - signed) בגודל 8 bytes. עליכם לשים בתווית Bool, שגודלה byte אחד, 1 אם המספר מתחלק ב4, ו0 אחרת.

תרגיל 2 (20 נק')

עליכם לממש את ex2 המוגדרת בקובץ ex2.asm.

בתרגיל זה תקבלו את שלוש התוויות הבאות:

- source, destination – שתי כתובות זיכרון.
- num – מספר עם סימן (signed) בגודל 4 bytes.

עליכם להעתיק <num> bytes המתחילים בכתובת source אל הכתובת destination. הניחו שמוקצים מספיק בתים והגישות חוקיות.

במקרה שבו num חיובי, על התוכנית להתנהג בדיוק כמו [memmove](#).

במקרה ש- num שלילי (לפי שיטת המשלים ל-2), יש להעתיק מהסוף להתחלה. לדוגמה, אם num מכיל -2, הבייט שנמצא בכתובת source יהיה בכתובת destination+1 והבייט שנמצא בכתובת source+1 יהיה בכתובת destination.

תרגיל 3 (20 נק')

עליכם לממש את ex3 המוגדרת בקובץ ex3.asm.

בתרגיל זה תקבלו את שלוש התוויות הבאות:

- array1, array2 – שני מערכים של מילים כפולות (4 bytes) המכילות מספרים חיוביים חסרי סימן (unsigned) ממיינים מהגדול לקטן. מובטח שכל מערך יסתיים במילה כפולה שמכילה את המספר 0, והאיבר שלפני ה-0 הוא האחרון.
- mergedArray – תווית יעד.

מצופה מהקוד שלכם לפעול כך שבסוף ריצתו, mergedArray יכיל מערך שאתם תמלאו (כלומר בכתובת הזו נמצא האיבר הראשון, והשני נמצא בכתובת זו + 4 bytes וכו'), של האיברים משני המערכים, גם הוא ממין מהגדול לקטן (עד כה תיארנו איחוד שני מערכים ממיינים למערך ממין), אך כך שכל איבר יופיע פעם אחת בלבד (אין התחייבות לכמות הפעמים שהאיברים מופיעים בהם במערכי המקור). גם המערך שלכם צריך להסתיים ב-0.

תרגיל 4 (24 נק')

עליכם לממש את ex4 המוגדרת בקובץ ex4.asm.

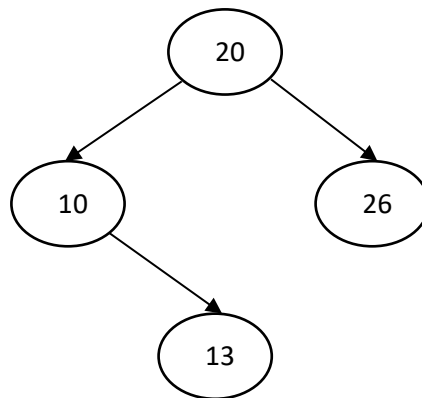
בתרגיל זה עליכם לממש הוספה של איבר לעץ חיפוש בינארי.

בעץ חיפוש בינארי, לכל איבר בעץ מתקיימת התכונה הבאה – כל האיברים בתת העץ השמאלי שלו קטנים ממנו, וכל האיברים בתת העץ הימני שלו גדולים ממנו. שימו לב שתת העץ יכול להיות ריק. כל איבר בעץ בתרגיל זה יורכב מ-3 חלקים, כמתואר ב-struct הבא:

```
struct Node {
    long data;
    struct Node *lson;
    struct Node *rson;
}
```

למשל, נתונה לכן הדוגמה של עץ חיפוש בינארי תקין, כאשר root מצביע על האיבר הראשון בעץ (השורש), והעץ אותו היא מתארת.

```
root:
    .quad    A
A:
    .quad    20
    .quad    B
    .quad    C
B:
    .quad    10
    .quad    0
    .quad    D
C:
    .quad    26
    .quad    0
    .quad    0
D:
    .quad    13
    .quad    0
    .quad    0
```



אז איך תתבצע הוספת איבר?

אתם תקבלו label נוסף, ששמו new_node באופן הבא:

```
new_node:    .quad    ????, 0, 0
```

ועליכם להוסיף אותו לעץ.

במידה והערך של new_node (שמסומן בדוגמה הנ"ל כ-???) כבר קיים בעץ, העץ לא ישתנה.

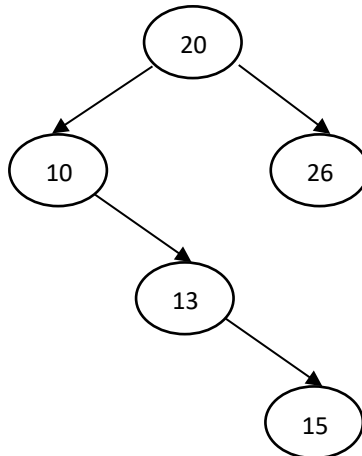
המשך ההסבר על תרגיל 4 בעמוד הבא

למשל, עבור העץ הנתון, נדגים שתי הוספות.

הוספה מהצורה:

new_node: .quad 15, 0, 0

יגרור שהעץ החדש ייראה כך:



לעומת זאת, הוספה מהצורה הבאה:

new_node: .quad 20, 0, 0

יגרור שהעץ החדש ייראה בדיוק כמו העץ המקורי, כי 20 כבר קיים בעץ.

הנחות והערות:

- תזכורת: עץ הוא גרף קשיר ללא מעגלים.
- לכל צומת בעץ יש לכל היותר 2 בנים.
- התוויות שמצביעה על הצומת הראשי של העץ תקרא root, אך אין התחייבות על השמות של שאר התוויות בעץ הנתון.
- אם root הוא NULL (מצביע לכתובת 0), הצומת new_node צריך להיות שורש העץ.
- אין לשנות את מבנה העץ, מלבד הוספת האיבר החדש (בפרט, אין לבצע איזונים למיניהם).

תרגיל 5 (24 נק')

עליכם לממש את ex5 המוגדרת בקובץ ex5.asm.
קלט:

- head - מצביע לתחילת רשימה מקושרת.
- Value – מספר שלם (4 בייטים)

עליכם להחליף בין שני האיברים המכילים את הערך שבתוויות val, אם ורק אם יש בדיוק שני איברים המכילים ערך זה. בכל מקרה אחר אין לבצע שום שינוי.

שימו לב שהרשימה מיוצגת בזכרון בדומה לדוגמה מתרגול 3 (כיתה הפוכה), אך כמובן עם ערך אחד בלבד ובגודל 8 bytes.

הערות אחרונות

איך בונים ומריצים לבד?

כאמור בתחילת התרגיל, נתון לכם קובץ טסט לתרגיל 3 וקובץ `run_test.sh`. אתם יכולים לשנות את הקובץ של תרגיל 3 ולהשתמש במבנה שלו כדי לבדוק תרגילים אחרים באותה הצורה, כפי שהוסבר קודם לכן בהקדמה לתרגיל. כדי להריץ, או לנפות שגיאות:

```
as ex3.asm ex3sample_test -o my_test.o #run assembler (merge the 2 files into one asm before)
```

```
ld my_test.o -o ex3 #run linker
```

```
./ex3 #run the code
```

```
gdb ex3 #enter debugger with the code
```

את הקוד מומלץ לדבג באמצעות `gdb`. לא בטוחים עדיין איך? על השימוש ב-`gdb` תוכלו לקרוא עוד במדריך באתר הקורס

קלי קלות! (ואם לא – אנחנו זמינים בפיאצה)

שימו לב: למכונה הוירטואלית של הקורס מצורפת תוכנת `sasm`, אשר תומכת בכתיבה ודיבוג של קוד אסמבלי וכן יכולה להוות כלי בדיקה בנוסף ל-`gdb`. (פגשתם אותה בתרגיל בית 0).
כתבו ב-`cmd`:

```
sasm <path_to_file>
```

כדי להשתמש ב-SASM לבנייה והרצת קבצי ה-`asm`, עליכם להחליף את שם התווית `_start` בשם `main` (זאת מכיוון ש-`sasm` מזהה את תחילת הריצה על-ידי התווית `main`. **אל תשכחו להחזיר את `start` לפני ההגשה!**)

בדיקות תקינות

בטסט אתם תפגשו את השורות הבאות

```
movq $60, %rax
movq $X, %rdi    # X is 0 or 1 in the real code
syscall
```

שורות אלו יבצעו `exit(X)` כאשר `X` הוא קוד החזרה מהתוכנית – 0 תקין ו-1 מצביע על שגיאה.

בקוד שאתם מגישים, אסור לפקודה `syscall` להופיע. קוד שיכיל פקודה זו, יקבל 0.

ניתן גם לדבג באמצעות מנגנון הדיבוג של SASM במקום עם `gdb`, אך השימוש בו על אחריותכם (**כלומר לא נתמך על ידנו ולא נתחשב בבעיות בערעורים**). שימו לב לשוני בין אופן ההרצה ב-SASM לאופן ההרצה שאנו משתמשים בו בבדיקה שלנו).