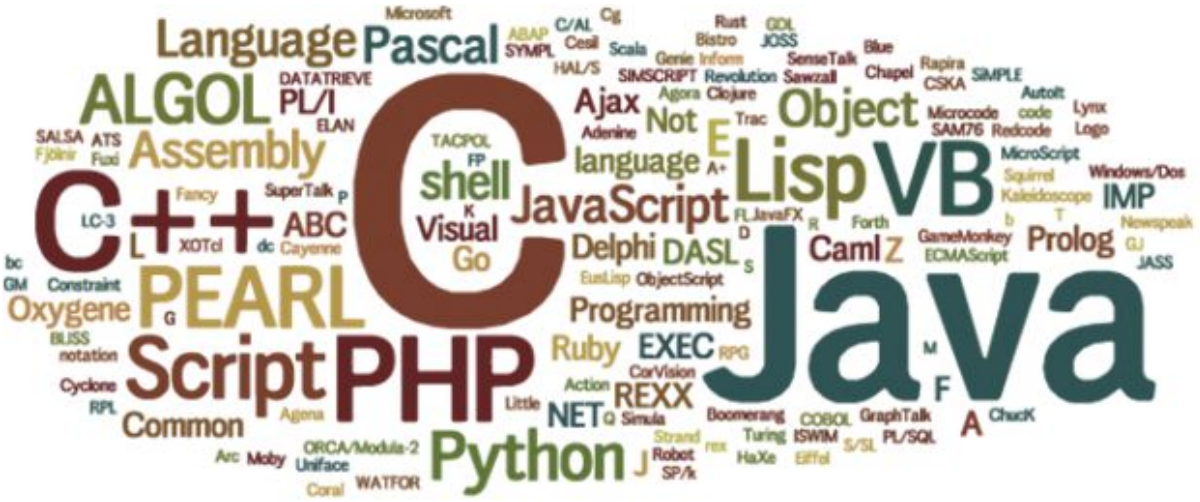
שפות תכנות, 236319

חורף 2018-2019



תרגיל בית 5

תאריך פרסום: 30/12/2018
מועד אחרון להגשה: 10/01/2019
מועד אחרון להגשה מאוחרת: 13/01/2019
מתרגל אחראי: יוסי גורשומוב
אי-מייל: yossigor@campus.technion.ac.il
בפניה בדוא"ל, נושא ההודעה (subject) יהיה "PL-EX5" (ללא המירכאות).

תרגיל בית זה מורכב משני חלקים, חלק יבש וחלק רטוב.
לפני ההגשה, ודאו שההגשה שלכם תואמת את הנחיות ההגשה בסוף התרגיל.
תיקונים והבהרות יפורסמו בסוף מסמך זה, אנא הקפידו להתעדכן לעתים תכופות.

חלק יבש

שאלה 1 RUST:

א)

בשפת Rust קיים מנגנון ownership המסייע בניהול הזיכרון. מתי משתחרר זיכרון שהוקצה בשפת Rust ,ומה בקוד יכול לשנות את זמן השחרור של הזיכרון בזמן ריצת התוכנית? הניחו שבקוד לא נעשה שימוש במילת המפתח unsafe (הערה: על מילת המפתח unsafe לא למדנו ולכן יש להתעלם מאפשרות זו).

<u>תשובה</u>: זיכרון משוחרר ברגע שה-owner שלו יוצא מסקופ. לפעמים נרצה להאריך את זמן החיים של זיכרון מעבר לסקופ שהוא נמצא בו, למשל איזשהו ערך שפונקציה יוצרת ורוצה להחזיר. ניתן לעשות זאת באמצעות המנגנון של lifetime ובעזרת המילה static, שערך עם lifetime כזה חי לאורך כל ריצת התוכנית.

ב) בשפת Rust יש גם מנגנון borrow checker. מתי חוקי בשפת Rust ליצור reference למשתנה, כלומר לעשות borrow?

תשובה: אם הרפרנס הוא מסוג immutable, אז ניתן להצהיר על מספר בלתי-מוגבל של רפרנסים כאלו. אם הרפרנס הוא מסוג mutable, אז אפשרי שיהיה רק אחד כזה בכל רגע נתון. בנוסף, לא ניתן שיהיו רפרנסים לאותו משתנה גם מסוג mutable וגם מסוג immutable באותו פרק זמן. כמובן, אם משתנה הוא מסוג immutable, לא ניתן ליצור רפרנס מסוג mutable אליו.

המוטיבציה לכללים אלה היא שרפרנס מסוג immutable לא יכול לשנות את המשתנה, ולכן אין בעיה עם רפרנסים מרובים מסוג זה. לעומת זאת, אם הרפרנס מסוג mutable, אז הוא יכול לשנות את המשתנה, ולכן אם יהיו יותר מאחד כאלה יכולות לקרות תופעות בלתי-צפויות כמו שני משתנים שכותבים לאותו משתנה. באופן דומה, רפרנסים מסוג mutable לא מצפים שהערך שלהם ישתנה תוך כדי, ולכן Rust לא מאפשרת שיהיו רפרנסים משני הסוגים באותו רגע. Rust אוכפת את כל אלה עוד ב-Compile time כדי להימנע מתופעות בלתי-צפויות ובאגים בשלב מוקדם.

בשני הסעיפים הבאים נתון קוד RUST שאינו מתקמפל. בצעו מספר של תיקונים מינימלי על מנת שהוא יבצע את המטרה שלו. לכל תיקון שהוספתם ציינו:

- מה גרם לבעיה
- כיצד התיקון שלכם פתר אותה.

התייחסו בתשובתכם למושגים שנלמדו בתרגול, תשובה ללא הסבר לא תתקבל.

ג) על הקוד להדפיס את המספרים 1-9 פעמיים.

```
fn main(){
   let mut vec = Vec::new();
   for i in 1..10 {
       vec.push(i);
   }
   for i in &vec{
       println!("{}",i);
   }
   for i in vec{
       println!("{}",i);
   }
}
```

הסיבה לשינוי הראשון היא שבהמשך אנו דוחפים איברים חדשים לווקטור, ולכן הווקטור חייב להיות mutable כדי לאפשר שינויים. השינוי הבא נובע כי לולאת ה-for היא בעצם sugar syntax לכך שהערך של vec מועבר (moved) לתוך הלולאה. מכאן שלאחר הלולאה הווקטור יוצא מסקופ ומת, ולכן לאחר מכן בלולאה הבאה לא ניתן יהיה להשתמש בערך שלו. עקרונית, הגיוני לשים גם & בפעם השנייה שעושים איטרציה על הווקטור, אבל במקרה הנ"ל זה לא נדרש.

ד)

על הקוד להדפיס `"The first element is:1"`

```
fn main(){
   let mut v = vec![1, 2, 3, 4, 5];
   let first = v[0]; // & was deleted
   v.push(6);
   println!("The first element is: {}", first);
}
```

הקוד לא עבד כי בשורה השלישית המשתנה first השאיל את הווקטור בצורה של immutable. כלומר, rust מבטיחה לו שהווקטור לא ישתנה. אבל בשורה 4 מנסים לשנות את הווקטור ולכן תהיה שגיאת קומפילציה. בצורה הזו, מכיוון שאיבר בווקטור הוא מסוג פרימיטיבי, המשתנה first יבצע העתקה של הערך ולא ישאיל.

ה)

נתון קטע הקוד הבא בשפת Rust. הפונקציה fold פועלת באופן דומה ל foldl.List שלמדנו בשפת ML.
הפרמטר השני לפונקציה fold הוא פונקציה אנונימית. החוקים של הפונקציות האנונימיות ב-Rust דומים לחוקים של הפונקציות האנונימיות ב-C++11 . הסבירו מדוע בהגדרת הפונקציה האנונימית מועבר b כרפרנס (מושאל) ו-a לא?

```
fn sum_vec(v: &Vec<i32>) -> i32 {
   v.iter().fold(0, |a, &b| a + b)
}
```

תשובה: האמת שהקוד ירוץ גם אם b לא יהיה מושאל, אבל באופן כללי זה לא הדבר הנכון לעשות.
אם נסמן f(a,b)=a+b. אז בפעם הראשונה קורה f(v[n-1], 0), ובפעם השנייה f(v[n-2], f(v[n-1],0). כלומר, המשתנה b בפונקציה האנונימית מקבל את ערך החזרה של הפונקציה האנונימית. מכיוון שהפונקציה האנונימית עשויה להחזיר משהו שאינה בבעלותה מלכתחילה, היא צריכה להשאיל אותו ממנה והלאה.

שאלה 2 חזרה על איחסון:

מהי שיטת ניהול הזכרון בשפת נים? האם יש איסוף אשפה? אריתמטיקה של מצביעים? משתני מחסנית הגוועים מעצמם? כיצד מיוצגים מערכים? מה קורה בחריגה מגבולות מערך? **קצת חזרה על החומר שנלמד בשיעור, וקצת חיפוש בויקיפדיה ובגוגל. עליכם ל"זהות" את המונחים "אריתמטיקה של מצביעים", ו"משתני מחסנית 'גוועים' מעצמם". לא סיפור גדול.**

<u>תשובה</u>:

ב-Nim יש שני סוגים של רפרנסים. סוג ראשון הוא רפרנס עם מעקב (traced reference) והסוג השני הוא רפרנס ללא מעקב (untraced reference). הסוג הראשון הוא זיכרון המנוהל על ידי אוסף האשפה והסוג השני לא. זאת אומרת, ב-Nim ניהול הזיכרון הוא אוטומטי באמצעות מנגנון איסוף זבל (המשתמש ב-reference counting עם cycle collection, מקור) אלא אם באופן מפורש מבקשים לנהל את הזיכרון בצורה ידנית. בנוסף, כדי למנוע שגיאות, Nim לא מספקת אריתמטיקה של מצביעים. אם מתעקשים לבצע זאת, יש לבצע casting ל-int ולבצע את החישובים.

בנוגע למשתני מחסנית, עבורם נקרא ה-destructor באופן אוטומטי. מתוך המדריך הרשמי:

=destroy(v) will be automatically invoked for every local stack variable v that goes out of scope.

(מקור)

בדומה לרוב השפות, מערכים הם רצף של זיכרון בגודל קבוע שלא ניתן לשנות בזמן ריצה שבו כל "תא" הוא מאותו טיפוס. פרט לכך, ב-Nim חלק מהטיפוס של מערך הוא הגודל שלו, שנשמר יחד איתו (מקור). השפה מתריעה על שגיאות חריגה מגבולות המערך בזמן קומפילציה (אם ניתן לדעת) או בזמן ריצה, אלא אם נאמר לה בצורה מפורשת לא לעשות זאת. מתוך המדריך הרשמי:

Arrays are always bounds checked (at compile-time or at runtime). These checks can be disabled via pragmas or invoking the compiler with the --boundChecks:off command line switch.

(מקור)

שאלה 3 - RTTI

1. מהו מנגנון RTTI? סכמו בשני משפטים.
<u>תשובה</u>: RTTI הינו מנגנון השומר על כל מופע של טיפוס גם מידע עליו הזמין בזמן ריצה.

2. האם בשפת C יש מנגנון RTTI?
<u>תשובה</u>: ב-C לא קיים מנגנון RTTI. הסיבה היא ש-C היא statically typed, ומהעיקרון ב-C שלא מסתירים עבודה שמתבצעת מאחורי הקלעים (או לפחות ממזערים את זה). מהסיבה הזו גם ניתן לפרש כל רצף של בתים ב-C בכל צורה ו-C לא תתריע לנו אם אנחנו עושים דברים לא הגיוניים (קומפיילרים מודרנים עשויים לספק אזהרות, אבל לא תהיה שגיאת קומפילציה).

3. הסבירו: כיצד מנגנון איסוף אשפה משתמש במנגנון RTTI?
<u>תשובה</u>: כדי להמחיש למה אנו צריכים RTTI, נראה מה קורה כאשר מנגנון איסוף האשפה לא חשוף למידע כזה. נניח שיש לנו רפרנס בשם A ומנגנון איסוף האשפה מגלה שהוא כבר לא בשימוש בתוכנית ולכן ניתן לפנות את הזיכרון שלו. כביכול, מספיק לדעת רק את גודל הזיכרון של A ולפנות אותו. אבל נשים לב שאם A מכיל רפרנס אחר בשם B וכל מה שמנגנון איסוף האשפה יעשה יהיה לפנות את בלוק הזיכרון ש-A מצביע אליו, אז יכול להיות שצריך לעדכן את מנגנון איסוף האשפה ש-B כבר לא בשימוש בתוכנית וניתן לפנות גם אותו. אבל כאמור, ללא RTTI לא ניתן לדעת ש-B הוא חלק מ-A.

   במילים אחרות, מנגנון איסוף האשפה משתמש במנגנון RTTI כדי לדעת לפרש הזיכרון ובאמצעותו מידע זה לעשות bookkeeping לרפרנסים.

4. תארו שימוש אחר (שאינו איסוף אשפה) למנגנון RTTI.
<u>תשובה</u>: ב-C++ מנגנון RTTI הוא מוגבל, אבל יש דוגמה לשימוש במנגנון הזה כאשר משתמשים בפולימורפיזם. מחלקה אבסטרקטית יכולה להגדיר פונקציות וירטואליות. מחלקות אחרות יכולות לרשת את המחלקה האבסטרקטית ולממש את אותן פונקציות וירטואליות. ואז, באמצעות מצביע למחלקה האבסטרקטית, ניתן להצביע לכל מחלקה שירשה מהמחלקה אבסטרקטית הזו, ולקרוא לפונקציות הוירטואליות ו-C++ תדע באופן דינמי לאיזה פונקציה היא צריכה לקרוא לפי המידע שמספק לה מנגנון RTTI.

שאלה 4 - פקודות

קראו את השקפים של פרק 6 וענו על השאלות הבאות בנוגע לשפה Rust:

1. מהן הפקודות האטומיות בשפה?

<u>תשובה</u>: באופן אידיאלי אנו מגדירים פקודה כחלק בתוכנית שמטרתו הוא שינוי מצב התוכנית ולא מחזיר ערך. Rust במובן מסויים משלבת תכנות פונקציונלי ואימפרטיבי, ולכן ההבחנה לא תמיד ברורה שכן כמעט כל פקודה ב-Rust היא פקודה-ביטוי. כלומר, גם מחזירה ערך וגם משנה את מצב התוכנית. לכן נוכל להגיד שהפקודות הן:

- פקודה ריקה - כתיבה של ";" בלי שום דבר לפניו.
- פקודת השמה - מתחילה במילה let (וזו אינה מחזירה ערך).
- ביטוי שבא אחריו ";".

2. מהם בנאי הפקודות בשפה?

<u>תשובה</u>: להלן בנאי הפקודות ב-Rust המוגדרים בצורה רקורסיבית.

- בנאי השרשור. מהצורה:

  {C1; C2; C3;...}

  כלומר קודם מתבצעת C1, אחר כך C2 ואחר כך C3 וכך הלאה.

- בנאי תנאי. מהצורה:

  if E C1 else C2

  כלומר משערכים את הביטוי E, ואם הוא מחזיר אמת אז מבצעים את C1 או את C2 אחרת.

- בנאי לולאה. ב-Rust יש שלושה סוגים של בנאי לולאה: for, while ו-loop. הצורה שלהם:

  for s in S C

  כלומר, לכל אלמנט s באוסף S בצע פקודה C.

  while E C

  כלומר, כל עוד הביטוי E משוערך לאמת בצע את C.

  loop C

  כלומר, בצע את C שוב ושוב, אלא אם באופן מפורש מבוצעת פקודת break בתוך C (שקול ל-while (true) בשפת C, למשל).

3. מהם ה Sequencers של השפה?

<u>תשובה</u>:

break, continue, return

(השתמשתי באוסף ה-keywords של השפה כדי לאתר אותם)

חלק רטוב

שאלה 1: רשימות מתחלפות

רקע: הטיפוס 'a list בשפת ML מוגדר כך:

```
datatype 'a list = nil | :: of 'a * 'a list
infixr 5 ::
```

(infixr הוא infix עם אסוציאטיביות ימנית. 5 הוא ספרה שמגדירה את העדיפות בין אופרטורים.)

פרט לכך, ML מגדירה תחביר מיוחד עבור רשימות, בעזרת סוגריים מרובעים:

```
[1,2,3] = 1::2::3::nil
```

החיסרון של רשימות ביחס ל tuple, למשל, היא שכל האיברים של רשימה נתונה הם מאותו טיפוס. בתרגיל זה ננסה לתקן את המצב.

1. הגדירו טיפוס גנרי "רשימה מתחלפת". ערכים מטיפוס זה מייצגים רשימות המחזיקות איברים מטיפוס 'a, אחריו איבר מטיפוס 'b, אחריו איבר מטיפוס 'a וכן הלאה. רשימות עשויות להיות בכל אורך שהוא, כולל 0.
   טיפוס שיאפשר לבצע את המשימות הנדרשות בהמשך השאלה יקיים את הדרישה.

   השלימו את התבנית עבור הטיפוס והוסיפו את ההגדרה לקובץ הפתרון:

```
datatype ('a, 'b) heterolist =
                NIL | :: of 'a * ('b, 'a) heterolist
```

2. בשפת ML לא ניתן להגדיר תחביר מקביל העושה שימוש בסוגריים מרובעים, אך אם היה ניתן להגדיר תחביר כזה היה אפשר ליצור רשימות כגון:

```
val x1y2 : (string, int) heterolist  = ["x",1,"y",2]
```

   הגדירו את הפונקציה:

```
build4 : 'a*'b*'a*'b -> ('a, 'b) heterolist
```

   שמחזירה רשימה מתחלפת, המחזיקה בדיוק את ארבעת הערכים שהועברו בפרמטר.
   הפתרון צריך להינתן בביטוי בודד (אין להגדיר פונקציות עזר פנימיות וכו'):

```
fun build4 (x,one,y,two) = x :: one :: y :: two :: NIL;
```

3. **(לא להגשה)**: השלימו את התבנית בהצהרה להלן כך שכל אחד מהמשתנים x, one, y, two יחזיק את הערך המתאים לשמו, ושתי השורות יתקמפלו וירוצו ללא חריגה (כאשר הן מבוצעות ברצף):

```
val x ++ one ++ y ++ two ++ NIL = build4 ("x",1,"y",2)
val ("x",1,"y",2) = (x,one,y,two)
```